Current Sharing To Minimize Power Losses In Parallel Converters Using Pso

Dan Li

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation
Li, Dan, "Current Sharing To Minimize Power Losses In Parallel Converters Using Pso" (2009). Theses and Dissertations. 1239.
https://scholarsjunction.msstate.edu/td/1239

This Graduate Thesis is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.
CURRENT SHARING TO MINIMIZE POWER LOSSES IN PARALLEL

CONVERTERS USING PSO

By

Dan Li

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2009
Copyright By

Dan Li

2009
CURRENT SHARING TO MINIMIZE POWER LOSSES IN PARALLEL CONVERTERS USING PSO

By

Dan Li

Approved:

Herbert L. Ginn
Associate Professor of Electrical and Computer Engineering
(Director of Thesis)

Noel N. Schulz
Professor of Electrical and Computer Engineering
(Committee Member)

Randolph F. Follett
Assistant Professor of Electrical and Computer Engineering
(Committee Member)

James E. Fowler
Professor and Director of Graduate Studies, Electrical and Computer Engineering

Sarah A. Rajala
Dean of Bagley College of Engineering
The Power Electronic Building Block (PEBB) concept leads to multifunctional converter systems, which provide robustness and flexibility in heavily power electronics based power systems. Systems comprised of flexible modular converters may have multiple possible operation conditions with respect to individual converters that meet the overall system goals. In this thesis, an optimization method for such flexible online power electronic systems is developed to minimize power losses of the overall group of converters in the system. Here the objective is to allocate sharing such that compensation objectives are met while the power loss of the entire parallel group of compensators is minimized. Considering optimization of an online power electronic system, convergence time and running in the feasible region should be taken into account. This thesis is focused on the development of a cost function and of an optimization method that meets these goals.
DEDICATION

I would like to dedicate this research to my God.
ACKNOWLEDGMENTS

The author expresses her sincere gratitude to her Lord Jesus Christ. Without him, this thesis could not have been finished. Sincere thanks are due to my advisor, Dr. Herbert L. Ginn, for his continuous support, guidance, and constant encouragement throughout the research work. Expressed appreciation is also due to my two committee members, Dr. Noel N. Schulz and Dr. Randolph F. Follett, for their invaluable help and willingness to serve on my graduate committee.

I want to thank the Department of Office of Naval Research for financial support during my graduate program.

I would also like to thank my families and friends for their support and love throughout the entire time.
# TABLE OF CONTENTS

DEDICATION .................................................................................................................................................... ii

ACKNOWLEDGMENTS ....................................................................................................................................... iii

LIST OF TABLES ................................................................................................................................................ vii

LIST OF FIGURES ........................................................................................................................................ viii

LIST OF ABBREVIATIONS .............................................................................................................................. x

CHAPTER

I. INTRODUCTION ................................................................................................................................. 1

1.1 Overview of the Problem ......................................................................................................................... 1
1.2 Global Constrained Optimization ............................................................................................................... 3
1.3 Power Electronic System ........................................................................................................................... 4
1.4 Contributions ........................................................................................................................................ 4
1.5 Thesis Outline ......................................................................................................................................... 5

II. SYSTEM WITH PARALLEL COMPENSATORS ................................................................................. 6

2.1 Introduction ........................................................................................................................................... 6
2.2 Voltage Source Converter ......................................................................................................................... 7
    2.2.1 Typical Structures of VSC ................................................................................................................. 7
    2.2.2 Multifunctional VSC Overview ........................................................................................................... 8
2.3 Power Losses .......................................................................................................................................... 10
    2.3.1 VSC Power Losses ............................................................................................................................ 10
    2.3.2 Losses of Compensator with VSC ..................................................................................................... 12
    2.3.3 Difficulty to Evaluate Losses in Compensator .................................................................................. 13
    2.3.4 Replaced Losses through DC Control Loop ....................................................................................... 14
2.4 System with Parallel Converters .............................................................................................................. 15
    2.4.1 Block Diagram of the Converter System ............................................................................................. 15
    2.4.2 Block Diagram of System with Parallel Converters .......................................................................... 16
2.5 Summary ................................................................................................................................................. 17
APPENDIX

A. CODES IN MATLAB .................................................................61

B. CODES IN LABWINDOWS/CVI ..................................................77
LIST OF TABLES

5.1 Some Important Parameters of the Simulation Model
    Shared by All Compensators .................................................................47
5.2 Important Parameters of the Three Compensators in MATLAB That Differ .....47
5.3 Parameters of PSO in MATLAB .......................................................................47
5.4 Final Particles after 450 Iterations .................................................................49
5.5 Final Velocities after 450 Iterations .................................................................50
5.6 Parameters of Three PEBBs in LabWindows .....................................................51
5.7 Parameters of PSO in LabWindows ....................................................................51
5.8 Coefficients of Each Converter after Optimization .............................................52
5.9 Checking Searching Results with Different Particles ........................................52
LIST OF FIGURES

2.1 Diagram of Three-Phase Full Wave VSC .............................................................8
2.2 Functional VSCs .................................................................................................9
2.3 Control Architecture of Shunt Connected VSC .................................................10
2.4 Structure of VSC Valve Units [49] ..................................................................11
2.5 Classification of Loss in a VSC Valve Unit [49] ...............................................12
2.6 Comparison of Real_Pwr and (DC_Cont*10000) .............................................15
2.7 Block Diagram of Flexible Converter System [2] .............................................16
2.8 Block Diagram of System with Parallel Converters ..........................................17
3.1 Flowchart of PSO Algorithm (Part 1) ...............................................................24
3.2 Flowchart of PSO Algorithm (Part 2) ...............................................................25
3.3 Flowchart of PSO Algorithm (Part 3) ...............................................................26
3.4 Max_a Calculated if Violating Constraints ......................................................30
4.1 Flowchart of the Loop of Initialization of Particles ...........................................34
4.2 Flowchart of Iteration Loop .............................................................................35
4.3 Flowchart of the Loop of Updating Local Best and Global Best .......................36
4.4 Test-Bed Configuration .....................................................................................37
4.5 Configuration of PEBB .....................................................................................38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>Diagram of System in Details</td>
<td>39</td>
</tr>
<tr>
<td>4.7</td>
<td>Interface to Connect PEBBs from Computer Supervisory</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>Parameter Control Panel</td>
<td>41</td>
</tr>
<tr>
<td>4.9</td>
<td>Control Panel with PSO Implemented</td>
<td>42</td>
</tr>
<tr>
<td>4.10</td>
<td>Flowchart of PSO Implemented in CVI (Part 1)</td>
<td>43</td>
</tr>
<tr>
<td>4.11</td>
<td>Flowchart of PSO Implemented in CVI (Part 2)</td>
<td>44</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS

GTO  Gate Turn-off Thyristor
HVDC  High-Voltage Direct Current
IEGT  Injection Enhanced Gate Transistor
IGBT  Insulated-Gate Bipolar Transistor
IGCT  Integrated Gate Commutated Thyristor
IPFC  Interline Power Flow Controller
MOSFET  Metal-Oxide-Semiconductor Field-Effect Transistor
PEBB  Power Electronic Building Block
STATCOM  Static Synchronous Compensator
SSSC  Solid State Series Compensator
UPFC  Unified Power Flow Controller
VSC  Voltage Source Converter
CHAPTER I

INTRODUCTION

1.1 Overview of the Problem

There has been much work in recent years toward modular power electronic system leading to the Power Electronic Building Block (PEBB) concept [1, 37-38, 46]. Also, there have been efforts in the area of multifunctional converter system. Both of these help to provide robustness and flexibility in heavily power electronics based power systems. Systems comprised of flexible modular converters may have multiple possible operating conditions with respect to individual converters that meet the overall system goals. However, while these may be equivalent with respect to the overall system goals, they may not be equivalent with respect to the operating efficiency of the converters.

In this thesis, an optimization method for such flexible power electronic systems is developed to minimize power losses of the overall group of converters in the system. Of course, there are a vast number of possible topologies that could be considered. Therefore, in order to bound the problem, only shunt connected voltage source converters (VSCs) are considered. Shunt connected VSCs [7, 14] are a very commonly employed converter topology. The example system considered here consists of shunt VSCs
connected in parallel to form a flexible and scalable compensator system. In order to achieve this goal, flexible active compensators [1-2] are used, where each compensator is appointed with selection coefficients corresponding to compensation of each orthogonal component of current to be compensated. All compensators are shunt connected, sharing the compensation task, thus the PEBB concept and multifunctional control allow flexible compensation that can be scaled by adding parallel VSCs.

Because the compensators may not have identical attributes, sharing may not simply be equally distributed when considering issues such as energy storage constraints and power losses of the group as a whole. Here the objective is to allocate sharing such that compensation objectives are met while the power loss of the entire parallel group of compensators is minimized. Considering the power loss, the scaling coefficients selection determines the power losses of each compensator converter. When there are degrees of freedom for selection of coefficients, the coefficients should be selected to minimize the power loss in the group of converters, while still meeting objectives. Optimization techniques may be used to determine the optimal solution with respect to power losses while meeting system compensation goals and constraints. In order to get a global optimal solution, stochastic algorithms would be appropriate, since it is expected that there will be optima.

The system to be optimized is an online power electronic system, thus the employed optimization algorithm should not be too slow, since running at low efficiency for a long time should be avoided. Also the system must operate within the feasible region at all times during the optimization process. This thesis is focused on the development of a cost function and of an optimization method that meets these goals.
1.2 Global Constrained Optimization

Compared to a local search, global optimization tries to find the global best among multiple local optima. However, there is no way to verify that the result is the global optimum unless it is a special case such as a convex optimization problem. Generally, based on the methods of operations, the optimization algorithms are categorized into deterministic algorithms and stochastic algorithms [12, 48-49]. Deterministic algorithms work based on certain fixed rules and steps. Thus, their outcomes are predictable and repeatable, and they do not vary as long as the algorithm is repeated with the same settings. Stochastic algorithms involve random factors, on which its outcome would depend. With random factors, it is more possible for stochastic algorithms to achieve a global optimum. However, stochastic algorithms may need much more time in order to get a satisfactory optimization result. The conditions of the optimum solution are decided by the problem itself and the applied algorithm as well. Some deterministic algorithms can locate all the global solutions with more CPU time and memory, while stochastic methods do not guarantee the global optimum when the search process ends [49]. However, stochastic algorithms are usually easier to implement and quicker to reach an approximate solution. In practice, most optimization problems have constraints [49]. For constrained optimization, typically it is a function of several variables, to be maximized or minimized, subject possibly to one or more constraints [9]. Considering the constraint techniques, they are dependent on the type of problems, or the optimization model and the requirements of the particular problem.
1.3 Power Electronic Systems

Power electronic systems play an important role in modifying the form of electrical energy, saving energy in power system transmission, motor efficiency, etc. A power electronic converter contains powerful semiconductor devices, which offer not only the advantage of high speed and reliability of switching, but, more importantly, a variety of innovative circuit concepts based on these power devices can enhance the value of electric energy [7]. Compensation is one of the widely used power electronics applications in order to enhance controllability and power transfer capability. Work in the compensation area appears extensively in the literature [1-4, 15-28, 35-39].

1.4 Contributions

With respect to meeting the compensation objectives, online flexible active compensators are utilized that employ a novel reference current generation technique [2]. With flexible active compensators, more options of scaling coefficient selection are available while still satisfying the power quality objective. This thesis aims at minimizing the power losses by selecting the most optimal set of scaling coefficient which yields the minimum power losses while achieving the compensation objective. Without the exact relationship between the power losses and the scaling coefficients, traditional methods of dealing with constraints are not suitable here, such as penalty function, Lagrangian method, etc. Moreover, it is an online optimization, which means the system only can run in the feasible region. In other words, each scaling coefficient selection is assured to satisfy the constraint limitation before they are sent to the system. Based on the above requirements, a novel particle swarm optimization (PSO) is applied, which includes a
proposed method to solve the constraints, making sure that all the particles are in the feasible region.

1.5 Thesis Outline

The structure of this thesis is as follows. Chapter II gives background information about the power electronic system studied in this thesis, including an introduction to voltage source converters, the power electronic system with parallel converters and power losses. Chapter III describes the problem solved in this thesis. It covers optimization methods discussion, the introduction of PSO and its application in the specific optimization problem. Chapter IV presents the PSO implementation, both in the MATLAB environment and in LabWindows/CVI, along with some background knowledge about LabWindows and flowcharts of the implemented algorithm. Chapter V presents the test cases and results. Finally conclusions along with future work are covered in Chapter VI.
CHAPTER II
SYSTEM WITH PARALLEL COMPENSATORS

2.1 Introduction

Voltage source converters (VSCs) are one of the most common topologies used in shunt compensator applications. Shunt compensator control systems combine current reference signal generation with a multiloop control structure that enables the VSC to function as a controlled current source [1-4, 7, 14-24, 26-27]. Depending on the current reference injected, the VSC may perform various functions, such as reactive power compensation, compensation for harmonic distortion, and compensation for unbalanced loads. If the employed reference generation technique is flexible because of the decompositions of the current, the particular function of a VSC would be adjusted based on the power quality requirement and compensation objectives. Such a reference signal generation technique provides flexible scaling coefficients corresponding to components of the current, including the active, reactive, harmonic and unbalanced components. A flexible active compensator allows each component of the current to be adjusted independently by any percentage of the reference current. Obviously, it is desirable that the compensator’s function would be controlled and changed based on the particular compensation needs. This chapter will introduce a system with flexible compensators, a
VSC overview and corresponding power losses discussion for the employed converter and system.

2.2 Voltage Source Converter

2.2.1 Typical Structures of VSC

A voltage source converter generates AC voltage from a DC voltage with particular switching sequences of semi-conductor devices, such as IGBT, GTO, MOSFET, etc [7]. On the DC side, the voltage is unipolar and supported by a capacitor, which is large enough to handle a sustained charge/discharge current keeping the DC voltage around a constant value [7]. When connected to an AC supply system, the interface between AC side voltage source and VSC is usually through a series inductor, transformer, or both, which is to ensure that the DC capacitor is not short-circuited and discharged rapidly into a capacitive load such as a transmission line. Figure 2.1 displays the diagram of a three-phase full wave VSC, comprised of six semiconductor switches $S_1$ to $S_6$ and three interface inductors $L_a$, $L_b$ and $L_c$. 
The switching sequences of those semiconductor switches in Figure 2.1 decide the output of a VSC in the magnitude, the phase angle and the frequency. Although it generates AC voltage from DC voltage, it has the ability to transfer power in either direction, depending on the direction of the current through the capacitor [7]. If the power flows from the AC side to the DC side, it is called a rectifier. It is called an inverter if the power flows the opposite way. Its capability to act as a rectifier or as an inverter with the instantaneous current flow in the positive (AC to DC side) or negative direction, respectively, is basic to VSC concepts [7].

2.2.2 Multifunctional VSC Overview

The VSC is the building block of the STATCOM, SSSC, UPFC, IPFC and other controllers in power electronics applications [7].
A multifunctional VSC would be desirable if the space is limited or for flexible requirements in the applications. These functions include compensation of reactive power, balancing of unbalanced currents, harmonic compensation, AC/DC and DC/AC power conversion for motor drives and other loads, and AC/DC/AC and DC/AC/DC energy conversion and transmission [4] as shown in Figure 2.2. As the previous section describes, series inductors form the interface between a VSC and an AC source voltage. Based on the current components that a VSC injects, it functions as a harmonic compensator, balancing compensator, etc. In addition to VSC, each compensator includes hierarchical control, comprised of current control and switching signal generation at the lower level, and a reference signal generation that includes an outer voltage control loop at the upper level. The control architecture is shown in Figure 2.3.
The compensator’s current, $i_c$, is decided by the reference current signal, which is extracted from the load current components.

2.3 Power Losses

2.3.1 VSC Power Losses

The controllable semiconductor devices (called valves) used in VSCs are generally GTO, IGCT, IEGT and IGBT, etc [49]. The IGBT is the most common choice for high power and high voltage VSC-HVDC transmission system used today because of its excellent characteristic features [49-50]. In this thesis, the IGBT is also employed in the VSC. For such semiconductor devices, the losses include switching losses, conduction losses and off-state losses, which can be neglected, since the leakage current...
during the off-state is negligibly small [50]. However, [49] a complete IGBT valve unit does not only consist of an IGBT but also a gate driver (1), a voltage grading circuit (2 may be used), a monitoring and protecting circuit (3), a current detecting circuit (4) and circuit impedance (5) as shown in Figure 2.4 [49].

![Diagram of VSC Valve Units](image)

**Figure 2.4  Structure of VSC Valve Units [49]**

The evaluation of power losses of VSC valve units (Figure 2.4) must include several parts [49], as shown in Figure 2.5.
In addition to the changing of operation states and structure of valves, the factors influencing the power losses of a VSC consist of topologies of the converter, index and mode of pulse width modulation (PWM), scheme of the valve driver, accessories of the valve unit, types of power load, DC side voltage and temperature of devices, etc [49].

2.3.2 Losses of Compensator in Addition to the VSC

In addition to the VSC, the compensator also includes the interface impedance. Thus, compensator losses should take the series inductors into account. The losses on interface circuits are dependent on the current through the inductor and the equivalent conductance of the inductor.
2.3.3 Difficulty of Evaluation of Losses in Compensator

The most direct way to determine the converter losses is to subtract the measured output power from the measured input power [49]. But the direct method would be not reasonable due to measurement system requirements since the difference between the two large quantities (output power and input power) is too small to provide a sufficiently accurate indication of the actual losses without additional measurement expense if the power electronic system is working at high power and voltage [49]. Thus, the direct subtracting method is only applied well in situations of small power.

Considering mathematical models of losses in converters, calculating the losses of the interface circuits may be not difficult, since they are mainly inductors in series and the losses are dependent on the current and devices themselves. One of the key challenges is evaluating the losses of the VSC’s valve units. However, as depicted in Figure 2.5, the total losses of the converter valve unit are divided into two categories, total IGBT model losses and total losses of accessories. The first category includes turn-on and turn-off losses of IGBTs, transient losses of forwarded diode (FWD) connected with the IGBT, and losses of FWD steady state. The other category are the losses on voltage divider, heat sink, driver unit and other facilities. A lot of research has been performed to propose more appropriate techniques to evaluate converter losses [49-50], which are generally categorized into physical models and mathematical analysis.

In order to obtain a physical model, the detailed information of the interior structure and working mechanism of the semiconductors is necessary, as well as the relationship between the model’s parameters and the actual elements [49]. Due to the difficulty of collecting all the above data, the mathematical analysis method is widely
used instead of the physical model. The mathematical analysis determines the numerical relation between the IGBT’s model loss and factors that affect the loss after measuring a large number of actual devices and forms equivalent power curves by mathematical technique [49]. However, it is also based on a large number of calculations and measurements to obtain the power curve, which is not suitable for an optimization problem. In this thesis, another technique is applied to evaluate the power losses through the DC control loop in a converter.

2.3.4 Replaced Losses through DC Control Loop

In order to make a VSC perform correctly, the DC side voltage across the capacitor should be constant, which is achieved by the DC control loop. Because of the losses in the converter, the voltage across the energy storage capacitor will decline unless these losses are compensated by injecting active power into the compensator. In this thesis, instead of computing the power losses, the output signal of the DC control loop is considered as equivalent losses since it is proportional to the total power losses in the converter. Figure 2.6 displays the comparison between real measured power losses (Real_Pwr) and the scaled output signal of the DC control loop (DC_Cont), where the scaler factor is 10000. It indicates that the exact power losses could be replaced by the output signal of the DC voltage control loop.
2.4 System with Parallel Converters

2.4.1 Block Diagram of the Converter System

Figure 2.7 [2] is the block diagram of the converter system used to develop the proposed loss minimization method, where the converter is a flexible active compensator with scaling coefficients $K_h$, $K_r$ and $K_u$. Those scaling coefficients decide the extracted percentages of current components injected into the system at the point of the converter connection. The VSC is constructed by a six IGBT three-phase bridge and its output inductors. The particular modulation technique is SVPWM with a dead-time of 2µs. The outside DC control loop is a typical PI controller aiming to maintain the DC bus voltage. In the applied compensator, the reference signal generation technique is based on
currents’ physical component (CPC) theory [1-2, 30] and recursive discrete Fourier transform (RDFT) [15-24], which are indicated by CPC based current waveform reconstruction algorithm block and CPC measurement algorithm block.

![Figure 2.7 Block Diagram of Flexible Converter System [2]](image)

2.4.2 Block Diagram of System with Parallel Converters

A prototype of the system is illustrated in Figure 2.8, where each converter is a black box working as a flexible compensator with selecting coefficients Kaj, Khj, Krj and Kuj representing percentage active component, harmonic component, reactive component and unbalanced component of the jth converter current, respectively. The entire test
system is comprised of power electronic system with three converters, which are flexible active compensators paralleled between a three-phase voltage source and nonlinear and unbalanced loads.

![Diagram of System with Parallel Converters](image)

**Figure 2.8** Block Diagram of System with Parallel Converters

### 2.5 Summary

This chapter introduces the power electronic system with parallel converters, which is studied in this thesis, including the typical structures of VSC, multifunctional VSC overview and the discussion of power losses of the converters. Those parallel converters are operated as flexible active compensators with scaling coefficients corresponding to the components of the current. The scaling coefficients of each converter could be adjusted according to the particular compensation mission, making its
current injected into or drawn from the system be equal to any percentage of the measured load current. Namely, for each converter, its reference signal current could be adjusted freely based on its compensation task. VSC is one of the key parts of the parallel flexible active compensator and it is described along with the associated control system. To simplify the analysis, the prototype is represented by paralleled black boxes as flexible active compensators shunted between three-phase voltage source and nonlinear and unbalanced loads.
CHAPTER III
PROBLEM DESCRIPTION

3.1 Introduction

Chapter III discusses the optimization problem, and covers the overview of optimization algorithms for various problems, particularly discussion about global optimization and online optimization. It also introduces particle swarm optimization (PSO) as a potential candidate method for converter loss minimization. It presents the reasons that a PSO method is selected to solve the global online optimization without certainty. Moreover, it specifies the particular optimization with constraints. Different from traditional application of PSO, a method dealing with constraints is proposed in order to make sure the system always runs within the feasible region, since it’s an online optimization problem.

3.2 Optimization Methods

3.2.1 Optimization Overview

Optimization refers to choosing the best element from some set of available alternatives. This means solving problems in which one seeks to minimize or maximize a
function (which is also known as a cost function, objective function or fit function) by systematically choosing the values from within an allowed set [6, 9-13, 31-32]. Based on the presence of constraints, optimization problems are divided into two classes: constrained optimization problems and un-constrained optimization problems. Considering the types of cost function and the constraints, there are many subfields of optimization. If the cost function is linear and the constraints are linear equalities and inequalities, it is called a linear programming problem [13, 32]. Otherwise, the nonlinear programming problem focuses on the case in which the cost function, the constraints, or both contain nonlinear parts [13, 32]. Certainly, there are other subfields which overlap between the linear programming problem and nonlinear programming program, i.e., quadratic programming allows the cost function to have quadratic terms, while the constraints must be specified with linear equalities and inequalities, which are listed in [32].

3.2.2 Global Optimization

Often, there exist multiple local optima of a function of which there is one best or global optimization solution. In terms of randomness, the global optimization algorithms are categorized into deterministic algorithms and stochastic algorithms. A deterministic algorithm is predictable and its result doesn’t change as long as the operations and parameters don’t change during the optimization process. For such reason, deterministic algorithms have been widely used in the past decades. But the deterministic algorithms may not be recommended for large, complex systems when global optimization is needed. In this case, a stochastic algorithm is more desirable since stochastic algorithms
have a better capability to avoid multiple local optima and get the best solution, because of the inserted random parameters or random operations during searching process. Normally, a stochastic algorithm is used along with heuristic methods, where the rules of the optimization process are based on expert experience and intuition. Thus, with heuristic methods, the optimization process consumes less time.

3.2.3 Online Optimization

The goal of online optimization is to provide the optimal set of parameters to an operating system, such as a power plant. Since the optimization process is performed on a running system, one of the key considerations is the tight time restriction. Another consideration is running without violating constraints, for in some cases, the running system must only run in the feasible region. For example, when optimizing the power losses online, if the optimization process takes too much time, the accumulated losses during the search process may be unacceptable.

3.3 Particle Swarm Optimization (PSO)

3.3.1 Introduction to PSO

PSO was developed by Eberhart and Kennedy in 1995 [33, 40-42]. It is a population based stochastic algorithm simulating the social behavior of a flock of birds. By communicating with each other, an individual moves toward another in a socio-cognitive space in order to solve a problem, which PSO simulates [51]. A problem and its corresponding fit function are given, candidate solutions, known as particles are
improved based on the neighbors’ information with an iteration process by certain communication rules and defined instruction.

3.3.2 Principle of PSO

The original global PSO [41] works in the way outlined below [42]:

1. Initialize a population (array) of particles with random positions and velocities on $d$ dimensions in the problem space.
2. For each particle, evaluate the fitness value.
3. Update individual local best by comparing individual’s fitness value with the previous individual local best fitness, fitness$_{i\text{best}}$.
4. Update the global best by comparing all the fitness values with the previous global best.
5. Update the velocity and position of the particle according to (3-1) and (3-2), respectively.
6. Loop to step 2 until a stop criterion is met, usually a maximum number of iterations, or when all the particles are converged to one position.

The following equations are used to determine velocity and position:

$$V[i][j]=w*V[i][j]+C1*R2*(X_{\text{best}[i][j]}–X[i][j])+C2*R2*(X_{\text{gbest}[j]}–X[i][j])$$  \hfill (3-1)

$$X[i][j]=X[i][j]+V[i][j]$$  \hfill (3-2)

where $C1$ and $C2$ are two positive constants, and $R1$ and $R2$ are uniform distribution random numbers. In Eqn3.1 the term associated with $C1$ is the “cognition” part, which represents the private thinking of the particle itself; the term associated with $C2$ is the “social” part, which represents the collaboration among the particles [30]. The
recommended values for these constants are $C1 = C2 = 2$ [30, 33, 40-44]. In (3-1), $w$ is inertia weight, balancing the global search and local search. If $w$ is zero or without the first part in (3-1), the new velocity is only dependent on the individual’s local best experience and the current global best among particles, which means the previous velocities have no influence on the next velocity. If so, it’s not hard to imagine that the particles would converge very fast and become a local search instead of a global search. Thus, the inertia weight $w$ mainly decides the performance of the PSO implementation. This is why many different techniques have been developed in order to better balance the fast convergence and more accurate global best [30, 33, 40-44]. For the PSO implementation in this thesis, more discussion and comparisons about the applied techniques are presented in Chapter V.

The procedure is shown in flowchart form in Figures 3.1 to 3.3. Figure 3.1 is part 1 of the flowchart, displaying initializing particle positions and particle velocities, where $n$ and $m$ are the number of particles and the number of dimensions of particle position, respectively.
In Figure 3.1, the $i^{th}$ particle is represented as:

$$X[I] = (X[I][1], \ldots, X[I][J], \ldots, X[I][m])$$ (3-3)
In the flowchart above, it assumes that $X[I][J]$, each element of particle $X[I]$ is connected with a linear inequality constraint, i.e., $X[I][J] \in [a, b]$, where $a$ and $b$ are the lower limit bound and upper limit bound, respectively. Under such conditions, the initialization always satisfies the linear equality constraint for each particle.

Figure 3.2 Flowchart of PSO Algorithm (Part 2)

Figure 3.2 is the second part of the PSO flowchart, which is to initialize the local best and global best fitness values. The best previous position of its own is called the local best position, whose fitness value is the local best fitness value, fitness$_{lbest}$ in this
case above. The best position among all the particles is called global best, whose fitness value is called global best fitness value, fitness_{gbest}. As seen in Figure 3.2, each particle has its own local best position and its corresponding local best fitness value, but the global best position is the only one which is selected from all particles based on fitness values. Figure 3.3 shows the final part of the PSO process, the searching process.

Figure 3.3  Flowchart of PSO Algorithm (Part 3)
3.3.3 Comparison of PSO and Other Stochastic Algorithms

As one of the stochastic algorithms, PSO has been compared with other stochastic algorithms [6, 10-13, 33, 41, 43]. The flowchart of PSO describes that it is a direct search method, which is usually derivative free, meaning that it depends only on the evaluation of a fitness function [33]. Furthermore, PSO algorithms are very easy to apply, with few parameters to be adjusted during the optimization process. As a population based technique, PSO is similar to genetic algorithms or evolutionary algorithms. Both algorithms initialize a group of population randomly and need fitness values to evaluate. Iteration process or searching procedure takes advantage of random techniques. However, they are different from the standpoint of their communication rules. In evolutionary algorithms, individuals communicate with each other, so the whole population moves like one group towards an optimal area [52]. In PSO, only global best gives out the information to others, so each particle only looks for the best solution. Moreover, in contrast to an evolutionary based algorithm, PSO has fewer operators, as compared to mutation or crossover in genetic algorithms. Instead, each individual particle is appointed with velocity, which is decided by its own current optimum and others’ optimum. Then the particle is updated with the previous particle position and its particular velocity, which always make it fly to the best particle.
3.4 Compensator Losses Minimization Using On-line PSO

3.4.1 Specific Conditions and Assumptions

Based on the data at hand, an accurate model that incorporates converter losses under all conditions is not available. Namely, there is no exact cost function between losses and scaling coefficients. However, a cost function relation between control coefficients and power losses can be determined from the DC bus controllers of the online system of converters, which is shown in Figure 2.7. For simplification, it is assumed that no large scale energy storage is available, thus the corresponding active component scaling coefficient $K_a$ is zero for every converter. It is also assumed that the system goal is total reactive power compensation, total harmonic compensation and total unbalanced current compensation, which means that the sum of scaling coefficients corresponding to the same type of current component equals 1.

3.4.2 Cost Function

From the currents, the losses can be calculated. However, lack of model details makes this very difficult to find a reasonable approximation. As mentioned in Chapter II, the output signal of DC voltage control loop is proportional to the real power losses. Consequently, the cost function of total power losses of these parallel converters can be replaced by the sum of scaled output signal of the DC voltage control loop. With a scalar inserted, the cost function is expressed as:

$$f = floor(100000 \sum_{j=1}^{N} DC\_Cont_j)$$

(3-4)
where $N$ is the number of converters. Since the output signal of the DC control loop is very small, which is far less than 1, a scalar $100000$ is chosen in order to make the fitness value more accurate to the real power losses and integers instead of floats. Also in the implementation of PSO, a function $floor$ is used to improve the convergence speed for the reason that it neglects the tiny difference, such as $999.8888$ and $999.8889$.

3.4.3 Constraints

The scaling coefficients represent the percentages of each component of current present in the reference signal. Thus, all coefficients are in the range between 0 and 1. Along with compensation goal and converter rating, all of the constraints are listed below.

$$(K_{h_j}, K_{r_j}, K_{u_j}) \in [0,1],$$

where $(j = 1, 2, ..., N)$

$$\sum_{j=1}^{N} K_{h_j} = 1;$$

$$\sum_{j=1}^{N} K_{r_j} = 1;$$

$$\sum_{j=1}^{N} K_{u_j} = 1;$$

$$\left\| i_j \right\| \leq I_i, \ (\text{Converter Current Rating}); \text{equally,}$$

$$\left\| i_j \right\|^2 = K_{h_j}^2 * I_{h_j}^2 + K_{r_j}^2 * I_{r_j}^2 + K_{u_j}^2 * I_{u_j}^2 \leq I_i^2$$

$I_h$, $I_r$, and $I_u$ are the CRMS values of harmonic component, reactive component and unbalanced component of the load current, respectively. With a constant load, obviously, $I_h$, $I_r$, and $I_u$ are constant, meaning that the third constraint is a quadric constraint.
3.4.4 Solving Constraints in PSO

Typically, PSO is applied in linear inequality constraints [30, 33, 40-44] or in the situation that running out of feasible region is allowed. Without a precise mathematical model of the cost value and selection coefficients, traditional methods of dealing with constraint, i.e. penalty functions, are not realizable. Moreover, it is an online optimization, thus only particles in the feasible region can be sent to the system. For the $i^{th}$ particle position, the next position is calculated based on the formula $\text{Pos}[i] = \text{Pos}_{\text{old}}[i] + \text{V}[i]$.

Before the new particle is sent to the system, all of the constraints are checked. If constraints are violated, maximum velocity would be computed, meaning how far the particle could move in the right direction in the search space as shown in Figure 3.4. With parameter $\text{Max}_a$ inserted, the linear or quadric inequalities are solved, where $\text{Pos}[i] =$

![Figure 3.4 Max_a Calculated if Violating Constraints](image)
\( \text{Pos}_{\text{old}}[i] + \text{Max}_a \* V[i] \), in order to gain the value of \( \text{Max}_a \). Obviously, \( \text{Max}_a \) is in the range between 0 and 1, since it is a percentage which shows the maximum distance the particular particle could move. Thus, if any of those inequalities are unsolvable, \( \text{Max}_a \) is zero, meaning the particle is already close to edge of the feasible region or even on the edge in the velocity’s direction. For such particular cases, all of the constraints are either linear inequalities or quadric inequalities, which make solving the \( \text{Max}_a \) easy in order to see how far the particle could fly in any direction.

3.5 Summary

This chapter describes the specific optimization problem in this thesis. Relative optimization methods are discussed based on the problem, which covers global optimization and online optimization. According to the properties of this online system, PSO is introduced, which is easily implemented. This chapter illustrates why PSO is selected as an appropriate optimization algorithm to solve this particular global online optimization without a power loss model. With quadratic inequalities, a new method of dealing with constraints is proposed, which is different from that commonly used in the application of PSO.
CHAPTER IV
PSO IMPLEMENTATION

4.1 Introduction

For online optimization, the system is in the running state during the optimization process. In order to perform the PSO, the fitness values of all particles are obtained through system measurements. The implementation of the PSO for the parallel converter system was first performed in simulation and then on a hardware test platform. The power electronic system with parallel flexible active compensator is simulated in MATLAB/Simulink. The PSO algorithm is implemented in a MATLAB M-File program. The precision of the scaling coefficients was selected to be two digits after the decimal point to match the hardware platform context interface. This has the additional benefit of narrowing down the search space. To validate the results, the PSO is also implemented on the experimental test bed in the C programming language, under LabWindows/CVI 8.5, where a supervisory control computer was used to send the compensation scaling coefficients to the flexible active compensators over a TCP/IP Ethernet communication link.
4.2 Implementation in MATLAB

The simulation of the power electronic system is built with Simulink blocks in the Simulink environment. To evaluate the fitness value, the Simulink model is executed by the PSO algorithm using the `sim` command. The program flowchart of the PSO implementation in MATLAB is shown in Figures 4.1, 4.2 and 4.3. Figure 4.1 shows the algorithm for the initialization of particles, including particle positions and velocities and the later part of initializations of local best and global best fitness values, represented as Lbest and Gbest respectively, where Pnum is the number of particles. After all initializations are done, it will begin the iteration loop, where the searching process begins.
Figure 4.1 Flowchart of the Loop of Initialization of Particles

Figure 4.2 displays the way the iteration loop works, updating local best and global best solution and calculating the new particles and velocities. The iteration searching process finishes when all the particles converge to the same position or it arrives at the maximum iteration number Iter_num.
Finally, Figure 4.3 shows the flowchart of updating local best and global best. After the particles are updated, the maximum velocity would be gained by solving a series of linear inequalities and quadratic inequalities. In order to get better performance with more accuracy, the Max_a would be recalculated until the new particles are in the feasible region.
4.3 Implementation in LabWindows/CVI

In order to validate the simulation results, the PSO algorithm is also implemented in the laboratory test-bed, where the supervisory control computer uses LabWindows/CVI to interface to the power electronic system over a TCP/IP Ethernet Communication link.
4.3.1 Laboratory Test-bed

The test-bed consists of a 208 V three-phase voltage source, linear and nonlinear three-phase loads, and parallel converters, composed of power electronic building blocks (PEBB) [1, 5, 37-39, 46]. The linear load can be either resistive or resistive and inductive and may be balanced or unbalanced.

Figure 4.4  Test-Bed Configuration

Figure 4.4 shows the test-bed with supervisory control computer and PEBBs, excluding the three-phase source and loads. Figure 4.5 shows the PEBB-based converter with bi-directional VSC. The current control is performed by a digital signal processor
(DSP), including computation for reference signal generation, i.e., RDFT CPC reconstruction algorithm mentioned in the previous chapter.

Figure 4.5 Configuration of PEBB

Figure 4.6 shows the detailed diagram of the specific system with three flexible converters controlled by computer. The scaling coefficient parameters are sent by the supervisory system control through Ethernet TCP/IP communication. After the parameters are sent to the power electronic system, the output signal of DC voltage
control loop of each converter (DC_Cont1, DC_Cont2 and DC_Cont3) is measured and sent back to the implemented PSO algorithm, also through TCP/IP communication.

Figure 4.6  Diagram of System in Details

4.3.2  LabWindows/CVI

The supervisory control system is developed under LabWindows/CVI, which provides an interface between computer control and the test-bed system, supporting the C language.
Figure 4.7 is the interface to connect the three PEBBs. Figure 4.7 displays the parameter control panel, sending orders or receiving particular data setting for system over TCP/IP Ethernet communication. All the parameters are set through the parameter control panel as Figure 4.8 above, including the IP address settings, the scaling coefficients, etc. The button “Down Load” is a function button, which send the parameters to from the setting panel to the PEBB, similarly, the button “Up Load” is to receive the parameter from the PEBB and display them on the computer supervisory control side. The computer supervisory control communicates with the three PEBB separately according to particular IP.
4.3.3 Implementation in LabWindows/CVI 8.5

As Figure 4.8 shows, each PEBB is controlled through the control panel interface built in LabWindows/CVI8.5, by the download buttons, which call download functions using TCP/IP communication. As shown in Figure 4.9, in order to perform the PSO algorithms, a new button “RunPSO” with callback function is inserted, which starts the searching process in the program.
By clicking the “RunPSO” button, the first particle with scaling coefficients is sent to the three PEBBs. Under LabWindows/CVI, a data request function (SPrtc_Pck_DCControl () ) must be called in order to get DC control loop output back from each PEBB. In the RunPSO function, SPrtc_Pck_DCControl(i) is called, where i denotes the PEBB number. Figures 4.10 and 4.11 are the flowcharts of the search process, which begins with i = 0.
Figure 4.10 Flowchart of PSO Implemented in CVI (Part 1)
Figure 4.11 Flowchart of PSO Implemented in CVI (Part 2)
4.4 Summary

This chapter presents how the PSO algorithm is implemented in both MATLAB environment and LabWindows/CVI8.5 platform. The flowcharts both in MATLAB and LabWindows/CVI 8.5 are described. A brief description is also provided for the PSO implementation in LabWindow/CVI8.5. For the later implementation, based on the successful implementation in MATLAB, the most difficult challenge is to make the whole optimization process keep running until it finishes. The next chapter discusses and analyzes some test cases.
5.1 Introduction

Test cases are applied to both the MATLAB/Simulink simulation of the converter system and to the physical test-bed using LabWindows/CVI in order to validate the performance of the developed converter loss minimization technique. In the MATLAB/Simulink simulation model, the system parameters, such as capacitor, switch resistance, etc. are adjustable, while those parameters are fixed in the real test-bed system. Thus, the simulation platform provides more flexibility to check the algorithm behavior and the test-bed is used for a final validation in a real system.

5.2 MATLAB/Simulink Test Case

In the simulation model, the power electronic system with parallel compensators consists of two loads: one linear load with an unbalanced condition, and a DC converter generating both harmonics and reactive power. Although the three compensators (PEBBs) are different, they share some parameters, such as DC bus reference voltage, sampling frequency, etc. which are listed in Table 5.1.
Table 5.1 Some Important Parameters of the Simulation Model Shared by All Compensators

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply line-to-line voltage</td>
<td>208V</td>
</tr>
<tr>
<td>Supply nominal frequency</td>
<td>60 Hz</td>
</tr>
<tr>
<td>DC bus reference voltage</td>
<td>500V</td>
</tr>
<tr>
<td>Sampling Frequency</td>
<td>12Khz</td>
</tr>
<tr>
<td>Switching Frequency</td>
<td>12Khz</td>
</tr>
</tbody>
</table>

5.2.1 Test Case

The three compensators differ in DC bus capacitor, current rating, coupling inductance Lc, and equivalent resistances for switches, as Table 5.2 displays. Parameters in the PSO algorithm are iteration number, particle number, and inertia weight as listed in Table 5.3.

Table 5.2 Important Parameters of the Three Compensators in MATLAB That Differ

<table>
<thead>
<tr>
<th></th>
<th>Coupling Inductor</th>
<th>Winding Resistance</th>
<th>Ron (Ohms)</th>
<th>DC Bus Capacitor</th>
<th>Current Rating (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEBB1</td>
<td>0.3</td>
<td>0.25</td>
<td>900uF</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>PEBB2</td>
<td>0.1</td>
<td>0.5</td>
<td>900uF</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>PEBB3</td>
<td>0.5</td>
<td>0.75</td>
<td>1800uF</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3 Parameters of PSO in MATLAB

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration Number</td>
<td>1000</td>
</tr>
<tr>
<td>Particle Number</td>
<td>20</td>
</tr>
<tr>
<td>Inertia Weight</td>
<td>Wend*{(Wstart/Wend)^((1+C3*k/Iter_num))}</td>
</tr>
</tbody>
</table>

The particle number is set as 20, which is a typical selection for PSO algorithm.
5.2.2  Test Result

As Chapter IV mentions, there is another stop criteria for the PSO search. When all particles have converged to the same point, it will stop searching. In this case, the searching iteration number is only 450 instead of 1000, for the reason that the simulation is very slow. The best particle is [0.22, 0.3, 0.56, 0.48, 0.51, 0.01], where 0.22, 0.3 and 0.56 are harmonic coefficient Kh, reactive coefficient Kr and unbalanced coefficient of the first converter separately, and 0.48, 0.51 and 0.01 are harmonic coefficient Kh, reactive coefficient Kr and unbalanced coefficient of the second converter separately. For the third converter:

\[
K_h = 1 - 0.22 - 0.48 = 0.3 \quad (5-1)
\]

\[
K_r = 1 - 0.3 - 0.51 = 0.19 \quad (6-2)
\]

\[
K_u = 1 - 0.56 - 0.01 = 0.33 \quad (5-3)
\]

As Table 5.3 describes, the current rating of the first converter is the biggest among the three converters, thus its total contribution for compensation is the most compared to the other two. In addition, the coupling inductance and equivalent resistance of switches decide the losses to certain extent, this is why the third converter contributes the lease to the overall compensation. Table 5.4 describes the final particle after the 450th iteration. Table 5.5 displays the corresponding velocities after 450th iteration. From the two tables, all particles have converged except the 7th particle and 11th particle, which are denoted by red color. It also means the corresponding velocities are zero in all directions.
except the particles are converged to the same point. As long as the local best and global best stay unchanged, the particle would not fly anywhere based on the principles of PSO.

Table 5.4  Final Particles after 450 Iterations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.23</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.29</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.57</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.58</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.46</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>
According to Chapter III, inertia weight is the factor deciding the convergence speed. Based on Eqn3.1,

\[
V[i,j] = w \times V[i,j] + C1 \times R1 \times (X_{lbest[i][j]} - X[i,j]) + C2 \times R2 \times (X_{gbest[j]} - X[i,j]);
\]

\[
w = \text{Wend} \times (\text{Wstart}/\text{Wend})^{(1/(1+C3 \times k/\text{Iter\_num}))}
\]

when the iteration number Iter\_num is smaller, w is bigger, then V[i][j] is bigger. So the particles would converge faster.
5.3 LabWindows/CVI Test Case

5.3.1 Test Case

The experimental test-bed is the same system as that in the simulation model including loads and those parameters in Table 5.1. However, the three compensators have the same IGBTs and coupling inductors, except for DC bus capacitors and assumed ratings as given in Table 5.6. Table 5.7 shows the parameters of the PSO algorithm in LabWindows/CVI, which is the same as those in MATLAB simulation.

Table 5.6 Parameters of Three PEBBs in LabWindows

<table>
<thead>
<tr>
<th></th>
<th>DC Bus Capacitor</th>
<th>Current Rating (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEBB1</td>
<td>1800uF</td>
<td>36</td>
</tr>
<tr>
<td>PEBB2</td>
<td>900uF</td>
<td>18</td>
</tr>
<tr>
<td>PEBB3</td>
<td>900uF</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5.7 Parameters of PSO in LabWindows

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration Number</td>
<td>1000</td>
</tr>
<tr>
<td>Particle Number</td>
<td>20</td>
</tr>
<tr>
<td>Inertia Weight</td>
<td>( W_{end} \times \left( \frac{W_{start}}{W_{end}} \right)^{\frac{1}{1+C_{3}*k/Iter_num}} )</td>
</tr>
</tbody>
</table>

5.3.2 Test Result

The searching results are different, as expected since the actual IGBT losses and inductor winding resistances are unknown. The best particle is \([0.97, 0.67, 0.39, 0, 0.21,\)
0.41], and the corresponding power loss is 960.8W. The optimum coefficients of each converter are listed below in Table 5.8.

Table 5.8  Coefficients of Each Converter after Optimization

<table>
<thead>
<tr>
<th></th>
<th>Kh</th>
<th>Kr</th>
<th>Ku</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEBB1</td>
<td>0.97</td>
<td>0.67</td>
<td>0.39</td>
</tr>
<tr>
<td>PEBB2</td>
<td>0</td>
<td>0.21</td>
<td>0.41</td>
</tr>
<tr>
<td>PEBB3</td>
<td>0.03</td>
<td>0.12</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Note that most of the compensation task was allocated to PEBB1. It is expected that if that PEBB1 is even slightly better than the others, minimizing losses would tend to drive it to accept most of the burden, up to the rating constraint. In order to verify that a minimum was in fact determined by the optimization process, the coefficients were varied around the values provided by the PSO and the power loss was recorded. The results are shown in Table 5.9. Note that the power loss does increase as each coefficient is varied indicating that the values determined by the PSO algorithm are in fact at a minima.

Table 5.9  Checking Searching Results with Different Particles

<table>
<thead>
<tr>
<th>Best Particle</th>
<th>0.97</th>
<th>0.67</th>
<th>0.39</th>
<th>0</th>
<th>0.21</th>
<th>0.41</th>
<th>960.8W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>0.87</td>
<td>0.67</td>
<td>0.39</td>
<td>0</td>
<td>0.21</td>
<td>0.41</td>
<td>963W</td>
</tr>
<tr>
<td>Test2</td>
<td>0.97</td>
<td>0.57</td>
<td>0.39</td>
<td>0</td>
<td>0.21</td>
<td>0.41</td>
<td>994.7W</td>
</tr>
<tr>
<td>Test3</td>
<td>0.97</td>
<td>0.67</td>
<td>0.49</td>
<td>0</td>
<td>0.21</td>
<td>0.31</td>
<td>974W</td>
</tr>
</tbody>
</table>
According to Table 5.9, the searching result after 1000 iterations shows that the proposed PSO works in the on-line system, although all of the particles were not completely converged at the 1000\textsuperscript{th} iteration.

5.4 Summary

This chapter presents both the results in MATLAB/Simulink and Labwindows/CVI. It discusses and analyzes the results as well. In the simulation, the proposed PSO works well, which is predicted to converge in less than 1000 iterations, based on the 450\textsuperscript{th} particles and their corresponding velocities. However, in the real test-bed, the system variation in the system during operation was found to decrease the convergence speed of the PSO algorithm. Within 1000 iterations, the best particle was checked to be the best at least locally.
6.1 Conclusions

A novel method was developed to minimize the power losses of an online power electronic system with parallel converters, while meeting the overall system objectives. Through developing a cost function and an optimization method, the tasks of those parallel compensators are distributed in a way such that the system always operates in the feasible region, the compensation goals are satisfied and the total power losses are minimized.

Considering the typical control strategy of a VSC, an equivalent cost function was developed based on the output of the DC control loop signal, which was measured online. A PSO algorithm with simple operations and parameters was then applied to solve the global optimization problem. Also, a novel method for constraints was proposed, especially for the online optimization using PSO, where all the particles must ensure that the system will run in the feasible region before the particles are sent to the system.

The proposed optimization method was applied in a MATLAB simulation first and then implemented in a hardware test-bed. In the former implementation, the PSO algorithm in MATLAB communicates with the Simulink simulation through the command “sim” directly. However, in the latter implementation, the hardware test-bed is
controlled and supervised by a Labwindows/CVI interface based on TCP/IP communication, where the parameters for each converter were sent one by one. Although this means that the system goals may not be met for the interval between the updating of parameters for the first and last converter, that interval is small and the impact of the sequential update can thus be neglected.

### 6.2 Future Work

The inertia weight is one of the key factors in the performance of the PSO algorithm with respect to convergence time. Better inertia weight can make the particles converge faster and also helps to ensure that the global optimum is found. The iteration number is a factor for convergence speed as well, since it’s involved in the speed update formula with the inertia weight. Thus, future work would try other possible inertia weight techniques in order to get better results.

The system investigated in this work was limited to a compensation application in order to ensure that the DC bus control loop output was only proportional to the system losses. The work should be extended so that developed loss minimization technique can be extended to other applications where active power is handled by the converters. This will require development of a method to separate out the loss related information from other signals in the DC voltage controller.
REFERENCES


APPENDIX A

CODES IN MATLAB
clear all
clc

%======================================================================
% System Parameters Settings of
% System
%
%======================================================================

Lc=1.5e-3;% inductance of the connect inductor
Rc=1e-4;% Resistance of the connect inductor
Ts=1/(60*200*100);% System sampling time for solver
Tc=Ts*100;% Control and PWM sampling time
Sc_Phase=90;% Source phase in degree
DC_Voltage=500;% Desired DC bus voltage
Dead_Time=2e-6;% PWM Dead_Time
DCKp=0.00368;% Kp for DC bus voltage regulator
DCKi=0.325;% Ki for DC bus voltage regulator
iA_Load = 22.01;% RMS value of load current of phase A
iB_Load = 33.91;% RMS value of load current of phase B
iC_Load = 37.53;% RMS value of load current of phase C

% Components of Current
ial_load = 43.16; %Ge=0.2075, U=208. Ge*U
ir_load = 27.7472; % Be=-0.1334
iu_load = 16.1013; % Mag(A)=0.07741
ih_load = sqrt((iA_Load^2+iB_Load^2+iC_Load^2)-(ial_load^2)-(ir_load^2)-(iu_load^2));
irms = 36;% Current Rating of PEBB1
irms1 = 18;% Current Rating of PEBB2
irms2 = 12;% Current Rating of PEBB3

%======================================================================
% Parameters Settings of PSO
% Algorithm
%======================================================================
Pnum = 20;% Number of Particles
Dimen = 6;% Dimensions of particle
Iter_num = 10;% Parameter for Iteration number
V = zeros(Pnum,Dimen);% Particles' velocities
Pos = zeros(Pnum,Dimen);% Particles' positions
PosOld = zeros(Pnum,Dimen);% The previous positions
Pos_Lbest = zeros(Pnum,Dimen);% Local best position for each particle
Pos_Gbest = zeros(1,Dimen);% Global best position
Lbest = zeros(Pnum,1);% Local best value for each particle
Stop = zeros((Pnum-1),Dimen);% Stop Criterion
FitValue = zeros(Pnum,1);% Store fitness value of each particle
%FitValue2BestCompare = zeros(1000,2);
Wstart = 0.95;
Wend = 0.4;
C1 = 2;
C2 = 2;
C3 = 10;
ConsStore = [];
ConsStore_Max = [];
ConsStore_Max_0 = [];
Maxa_k_iStore = [];
Max_a = inf;
%**********************************************************************
*** 
%======================================================================
==

% INITIALIZATION OF Positions and Velocities 
%======================================================================
==
for i = 1:Pnum % Initialize each particle
  %%Initialization for PEBB (Kh,Kr,Ku)
  for j = 1:(Dimen-3)
    Pos(i,j) = rand;
    A1(i,j) = Pos(i,j)*100;
    Pos(i,j) = floor(A1(i,j))/100;
    V(i,j) = 0;
  end %END for j = 1:(Dimen-3)

  %% Initialization for PEBB1 (Kh1,Kr1,Ku1)
  for j = 4:Dimen
    Pos(i,j) = (1-Pos(i,(j-3)))*rand;
    A1(i,j) = Pos(i,j)*100;
    Pos(i,j) = floor(A1(i,j))/100;
    V(i,j) = 0;
  end %END for j = 4:Dimen

%======================================================================
***
%% Check Constraints
Cons_111 = (Pos(i,1) >=0);
Cons_222 = (Pos(i,2) >=0);
Cons_333 = (Pos(i,3) >=0);
Cons_444 = (Pos(i,4) >=0);
Cons_555 = (Pos(i,5) >=0);
Cons_666 = (Pos(i,6) >=0);
Cons_1111 = (Pos(i,1) <=1);
Cons_2222 = (Pos(i,2) <=1);
Cons_3333 = (Pos(i,3) <=1);
Cons_4444 = (Pos(i,4) <=1);
Cons_5555 = (Pos(i,5) <=1);
Cons_6666 = (Pos(i,6) <=1);
Cons1 = ((Pos(i,1) + Pos(i,4)) <= 1);
Cons2 = ((Pos(i,2) + Pos(i,5)) <= 1);
Cons3 = ((Pos(i,3) + Pos(i,6)) <= 1);
Cons4 = (((Pos(i,1))*ih_load)^2+((Pos(i,2))*ir_load)^2+((Pos(i,3))*iu_load)^2) <= ((irms*0.95)^2));
Cons5 = (((Pos(i,4))*ih_load)^2+((Pos(i,5))*ir_load)^2+((Pos(i,6))*iu_load)^2) <= ((irms1*0.95)^2));
Cons6 = (((1-Pos(i,1)-Pos(i,4))*ih_load)^2+((1-Pos(i,2)-Pos(i,5))*ir_load)^2+((1-Pos(i,3)-Pos(i,6))*iu_load)^2) <= ((irms2*0.95)^2));
Cons = (Cons_111 && Cons_222 && Cons_333 && Cons_444 && Cons_555 && Cons_666 && Cons_1111 && Cons_2222 && Cons_3333 && Cons_4444 && Cons_5555 && Cons_6666 && Cons1 && Cons2 && Cons3 && Cons4 && Cons5 && Cons6);

IniCons(i) = Cons;
if (Cons == 1)
    IniCons1(i) = inf;
end

While1 = 0;
%% Regenerate the particle when violate the constraints
while (Cons == 0) % && (While1 <= 2))
    While1 = While1 +1;
    for j = 1:(Dimen-3) % PEBB1
        Pos(i,j) = rand;
        A1(i,j) = Pos(i,j)*100;
        Pos(i,j) = floor(A1(i,j))/100;
    end %END for j = 1:(Dimen-3)

    for j = 4:Dimen % PEBB2
        Pos(i,j) = (1-Pos(i,(j-3)))*rand;
        A1(i,j) = Pos(i,j)*100;
        Pos(i,j) = floor(A1(i,j))/100;
    end %END for j = 4:Dimen

    %% Check Constraints again;
    Cons_111 = (Pos(i,1) >=0);
Cons_222 = (Pos(i,2) >=0);
Cons_333 = (Pos(i,3) >=0);
Cons_444 = (Pos(i,4) >=0);
Cons_555 = (Pos(i,5) >=0);
Cons_666 = (Pos(i,6) >=0);
Cons_1111 = (Pos(i,1) <=1);
Cons_2222 = (Pos(i,2) <=1);
Cons_3333 = (Pos(i,3) <=1);
Cons_4444 = (Pos(i,4) <=1);
Cons_5555 = (Pos(i,5) <=1);
Cons_6666 = (Pos(i,6) <=1);
Cons1 = ((Pos(i,1) + Pos(i,4)) <= 1);
Cons2 = ((Pos(i,2) + Pos(i,5)) <= 1);
Cons3 = ((Pos(i,3) + Pos(i,6)) <= 1);
Cons4 = (((Pos(i,1))*ih_load)^2+((Pos(i,2))*ir_load)^2+((Pos(i,3))*iu_load)^2) <= ((irms*0.95)^2));
Cons5 = (((Pos(i,4))*ih_load)^2+((Pos(i,5))*ir_load)^2+((Pos(i,6))*iu_load)^2) <= ((irms1*0.95)^2));
Cons6 = (((1-Pos(i,1)-Pos(i,4))*ih_load)^2+((1-Pos(i,2)-Pos(i,5))*ir_load)^2+((1-Pos(i,3)-Pos(i,6))*iu_load)^2) <= ((irms2*0.95)^2));
Cons = (Cons_1111 && Cons_2222 && Cons_3333 && Cons_4444 && Cons_5555 && Cons_6666 && Cons_1111 && Cons_2222 && Cons_3333 && Cons_4444 && Cons_5555 && Cons_6666 && Cons1 && Cons2 && Cons3 && Cons4 && Cons5 && Cons6);
if (Cons == 1)
  IniConswhile1(i) = 1;
  TestWhile1(i) = While1;
end
if (While1 >= 10)
  exit;
end
end %END while (Cons == 0) //Initialization
end %END for i = 1:Pnum // Initialization for each particle
%**********************************************************************
***

%狭義バストとグローバルバストの初期化
%GlobalBEST & LocalBEST Initialization
Gbest = inf; % Initialize global best
for i = 1:Pnum % Initialize local best for each particle
  Lbest(i,1) = inf;
end %END for i = 1:Pnum % Initialize local best for each particle
for k = (1+10*0):(Iter_num*1-5)
%*****BEGIN*****FitValue for each Particle & Update Lbest
for i = 1:Pnum % Cost Value for each Particle & Update Lbest
    Kh = Pos(i,1);
    Kr = Pos(i,2);
    Ku = Pos(i,3);
    Kh1 = Pos(i,4);
    Kr1 = Pos(i,5);
    Ku1 = Pos(i,6);
    PosTotal(((k-1)*Pnum+i),1) = Pos(i,1);
    PosTotal(((k-1)*Pnum+i),2) = Pos(i,2);
    PosTotal(((k-1)*Pnum+i),3) = Pos(i,3);
    PosTotal(((k-1)*Pnum+i),4) = Pos(i,4);
    PosTotal(((k-1)*Pnum+i),5) = Pos(i,5);
    PosTotal(((k-1)*Pnum+i),6) = Pos(i,6);
    VTotal(((k-1)*Pnum+i),1) = V(i,1);
    VTotal(((k-1)*Pnum+i),2) = V(i,2);
    VTotal(((k-1)*Pnum+i),3) = V(i,3);
    VTotal(((k-1)*Pnum+i),4) = V(i,4);
    VTotal(((k-1)*Pnum+i),5) = V(i,5);
    VTotal(((k-1)*Pnum+i),6) = V(i,6);
    k %display which iteration it is
    i %display which particle it is in kth iteration
    display('Kh,Kr,Ku;Kh1,Kr1,Ku1 are sent to system');
    options = simget('May');
    display('begin sim')
    [t,x,y] = sim('May',[0 8/60],options);
    display('Sim is done')
    FitValue(i,1) = floor((y((8/60/Ts+1),2))*100000);
% Update LocalBEST
if (FitValue(i,1) < Lbest(i,1))
    Lbest(i,1) = FitValue(i,1);
    for j = 1:Dimen
        Pos_Lbest(i,j) = Pos(i,j);
% if (FitValue(i,1) < Lbest(i,1))
%end for i = 1:Pnum //Cost Value for each Particle & Update Lbest
%
%*****END*****FitValue for each Particle & Update Lbest
CostTotal = [CostTotal,FitValue];
%
%*****BEGIN**********Update Gbest & Pos_Gbest
[MinFit,MinFitIndex] = min(FitValue); % UPDATE GLOBALBEST AND
if (MinFit < Gbest) % GLOBALBEST POSTION Update
Gbest = MinFit;
Gbest %output Gbest each iteration
for j = 1:Dimen
Pos_Gbest(1,j) = Pos(MinFitIndex,j);
end
Pos_Gbest %output Pos_Gbest each iteration
%
%*******END***** Update Gbest & Pos_Gbest
Gbest %output Gbest each iteration
MinFitIndex
%
%************************BEGIN Update Velocities and Particles
for i = 1:Pnum % update Each Particle V,P
for j = 1:Dimen % PEBl(the first one)
r1(j) = rand;
r2(j) = rand;
W3 = (Wstart-Wend)*((Iter_num-k)/Iter_num)+Wend; %LDIW-
PSO2
% W3 = -(Wstart-Wend)*((k/Iter_num)^2)+Wstart; % parabola
% opens downward WORST
% W3 = (Wstart-Wend)*((k/Iter_num)^2)+(Wend-
Wstart)*(2*k/Iter_num)+Wstart; % parabola opens upward
% W3 = Wend*((Wstart/Wend)^((1/(1+C3*k/Iter_num)))); % power
% W3 = Wend*((Wstart/Wend)^((1/(1+C3*k/1000)))); % power
Iter_num = 1000
%
W3 = 0.9;
V(i,j) = W3*V(i,j)+C1*r1(j)*(Pos_Lbest(i,j) -
Pos(i,j))+C2*r2(j)*(Pos_Gbest(1,j) - Pos(i,j));
V(i,j) = floor(V(i,j)*100)/100;
PosOld(i,j) = Pos(i,j);
Pos(i,j) = PosOld(i,j) +V(i,j);
Pos(i,j) = floor(Pos(i,j)*100)/100;
end %END for j = 1:Dimen // each particle update
%
% Check Constraints
Cons111 = (Pos(i,1) >=0);
Cons222 = (Pos(i,2) >=0);
Cons333 = (Pos(i,3) >=0);
Cons444 = (Pos(i,4) >=0);
Cons555 = (Pos(i,5) >=0);
Cons666 = (Pos(i,6) >=0);
Cons1111 = (Pos(i,1) <=1);
Cons2222 = (Pos(i,2) <=1);
Cons333 = (Pos(i,3) <=1);
Cons4444 = (Pos(i,4) <=1);
Cons5555 = (Pos(i,5) <=1);
Cons6666 = (Pos(i,6) <=1);
Cons11 = ((Pos(i,1) + Pos(i,4)) <= 1);
Cons22 = ((Pos(i,2) + Pos(i,5)) <= 1);
Cons33 = ((Pos(i,3) + Pos(i,6)) <= 1);
Cons44 = (((Pos(i,1))*ih_load)^2+((Pos(i,2))*ir_load)^2+((Pos(i,3))*iu_load)^2) <= ((irms*0.95)^2);
Cons55 = (((Pos(i,4))*ih_load)^2+((Pos(i,5))*ir_load)^2+((Pos(i,6))*iu_load)^2) <= ((irms1*0.95)^2);
Cons66 = (((1-Pos(i,1)-Pos(i,4))*ih_load)^2+((1-Pos(i,2)-Pos(i,5))*ir_load)^2+((1-Pos(i,3)-Pos(i,6))*iu_load)^2) <= ((irms2*0.95)^2);
Const = (Cons111 && Cons222 && Cons333 && Cons444 && Cons555 && Cons666 && Cons1111 && Cons2222 && Cons3333 && Cons4444 && Cons5555 && Cons6666 && Cons111 && Cons222 && Cons333 && Cons444 && Cons555 && Cons666);
IterConst(k,i) = Const; %record every particle constraint
result
if (Const == 0)
    resultCons = [k;i;Pos(i,1);PosOld(i,1);V(i,1);
    Pos(i,2);PosOld(i,2);V(i,2);
    Pos(i,3);PosOld(i,3);V(i,3);
    Pos(i,4);PosOld(i,4);V(i,4);
    Pos(i,5);PosOld(i,5);V(i,5);
    Pos(i,6);PosOld(i,6);V(i,6);
    Cons11;Cons22;Cons33;Cons44;Cons55;Cons66;
    Cons111;Cons222;Cons333;Cons444;Cons555;Cons666;
    Cons1111;Cons2222;Cons3333;Cons4444;Cons5555;Cons6666;
    Cons11111;Cons22222;Cons33333;Cons44444;Cons55555;Cons66666;
    Cons111111;Cons222222;Cons333333;Cons444444;Cons555555;Cons666666;Const];
    ConsStore = [ConsStore,resultCons];
end
While2 = 0;
TestWhile2(k,i) = Inf;
if (Const == 0)
    for g=1:Dimen
        VOld(i,g) = V(i,g);
    end
end

%% Regenerate the particle when violate the constraints
while (Const == 0) && (While2 <= 2))
    While2 = While2+1;
    Max_a = 999;
    x1 = solve('(Pos(i,1)+x1*VOld(i,1))+(Pos(i,4)+x1*VOld(i,4)) = 1');
    x2 = solve('(Pos(i,2)+x2*VOld(i,2))+(Pos(i,5)+x2*VOld(i,5)) = 1');
end
% x3 = 
solve('(Pos(i,3)+x3*VOld(i,3))+(Pos(i,6)+x3*VOld(i,6)) = 1');
% 0<=(Pos(i,1)+x1*VOld(i,1))

% First Each (Kh,Kr,Ku,Kh1,Kr1,Ku >=0)
for m0 = 1: Dimen
    if (VOld(i,m0) >= 0)
        xm1(m0) = 1;
    end
    if (VOld(i,m0) < 0)
        if ((-PosOld(i,m0)/VOld(i,m0)) <=1)
            xm1(m0) = -PosOld(i,m0)/VOld(i,m0);
        else
            xm1(m0) = 1;
        end
    end
end

% First Each (Kh,Kr,Ku,Kh1,Kr1,Ku <=1)
for m0 = 1: Dimen
    if (VOld(i,m0) == 0)
        xm2(m0) = 1;
    end
    if (VOld(i,m0) > 0)
        if (((1-PosOld(i,m0))/VOld(i,m0)) > 1)
            xm2(m0) = 1;
        else
            xm2(m0) = (1-PosOld(i,m0))/VOld(i,m0);
        end
    end
    if (VOld(i,m0) < 0)
        xm2(m0) = 1;
    end
end

% (Kh+Kh1<=1) && (Kr+Kr1<=1) && (Ku+Ku1<=1)
la1 = VOld(i,1)+VOld(i,4); %PosOld(i,1)+Max_a*VOld(i,1) + PosOld(i,4)+Max_a*VOld(i,4) <=1
lb1 = 1-PosOld(i,1)-PosOld(i,4);
la2 = VOld(i,2)+VOld(i,5);
lb2 = 1-PosOld(i,2)-PosOld(i,5);
la3 = VOld(i,3)+VOld(i,6);
lb3 = 1-PosOld(i,3)-PosOld(i,6);

if (la1 == 0) % Kh+Kh1 <=1; la1*x1<=lb1
    x1 = 1;
end
if (la1 >0)
    if ((lb1/la1) > 1);
        x1=1;
    else

69
x1 = lb1/la1;
end
end
if (la1 < 0)
x1 = 1;
end % END Kh+Kh1 <=1; la1*x1<=lb1
if (Max_a ~= 0) % Kr+Kr1 <=1; la2*x2<=lb2
if (la2 == 0)
x2 = 1;
end
if (la2 > 0)
if ((lb2/la2) > 1);
x2=1;
else
x2 = lb2/la2;
end
end
if (la2 < 0)
x2 = 1;
end
end %END if (Max_a ~= 0) % Kr+Kr1 <=1; la2*x2<=lb2
if (Max_a ~= 0) % Ku+Ku1 <=1; la3*x3<=lb3
if (la3 == 0)
x3 = 1;
end
if (la3 > 0)
if ((lb3/la3) > 1);
x3=1;
else
x3 = lb3/la3;
end
end
if (la3 < 0)
x3 = 1;
end
end %END if (Max_a ~= 0) % Ku+Ku1 <=1; la3*x3<=lb3
%%
% x4 = solve('((PosOld(i,1)+x4*VOld(i,1))^2)*(ih_load^2) + ((PosOld(i,2)+x4*VOld(i,2))^2)*(ir_load^2)+((PosOld(i,3)+x4*VOld(i,3))^2)*(iu_load^2)=(irms^2)');
% x5 = solve('((PosOld(i,4)+x5*VOld(i,4))^2)*(ih_load^2) + ((PosOld(i,5)+x5*VOld(i,5))^2)*(ir_load^2)+((PosOld(i,6)+x5*VOld(i,6))^2)*(iu_load^2)=(irms1^2)');
% x6 = solve('((1-(PosOld(i,1)+x6*VOld(i,1))-(PosOld(i,4)+x6*VOld(i,4)))^2)*(ih_load^2) + ((1-(PosOld(i,2)+x6*VOld(i,2))-(PosOld(i,5)+x6*VOld(i,5)))^2)*(ir_load^2)+((1-(PosOld(i,3)+x6*VOld(i,3))-(PosOld(i,6)+x6*VOld(i,6)))^2)*(iu_load^2)=(loadu^2)=(irms2^2)');
if (Max_a ~= 0) % Current Limit PEBB
    A11 = (VOld(i,1)*ih_load)^2 + (VOld(i,2)*ir_load)^2 + 
          (VOld(i,3)*iu_load)^2;
    B11 = 2*PosOld(i,1)*VOld(i,1)*(ih_load^2) + 
          2*PosOld(i,2)*VOld(i,2)*(ir_load^2) + 
          2*PosOld(i,3)*VOld(i,3)*(iu_load^2);
    C11 = (PosOld(i,1)*ih_load)^2 + (PosOld(i,2)*ir_load)^2 + 
          (PosOld(i,3)*iu_load)^2 - ((irms*0.95)^2);
    delta1 = B11^2 - 4*A11*C11;
    if (A11==0) % not quadratic function
        if (B11 == 0)
            x4 = 1;
        elseif (B11 > 0)
            if ( (-C11/B11) > 1)
                x4=1;
            else
                x4 = (-C11/B11);
            end
        elseif (B11 <0)
            x4 = 1;
        end
    elseif (A11>0) % parabola opens upward
        if (delta1<0)
            Max_a = 0;
        elseif (delta1 > 0)
            x41=(-B11-sqrt(delta1))/(2*A11);
            x42=(-B11+sqrt(delta1))/(2*A11);
            if (x41>1)
                Max_a = 0;
            elseif ((x41 <= 1) && (x42 >= 1))
                x4 = 1;
            elseif ((x42 >= 0) && (x42 <= 1))
                x4 = x42;
            elseif (x42 < 0)
                Max_a = 0;
            end
        elseif (delta1 == 0)
            x412 = (-B11+sqrt(delta1))/(2*A11);
            if (x412>1)
                Max_a = 0;
            elseif (x412 < 0)
                Max_a = 0;
            else
                x4 = x412;
            end
        end
    end % END if (A11==0) % Solve Constraints
end % END if (Max_a ~= 0) % Current Limit PEBB

if (Max_a ~= 0) % Current Limit PEBB1
    %
end
\[ A22 = (V_{Old}(i,4) \cdot ih\_load)^2 + (V_{Old}(i,5) \cdot ir\_load)^2 + (V_{Old}(i,6) \cdot iu\_load)^2; \]
\[ B22 = 2 \cdot Pos_{Old}(i,4) \cdot V_{Old}(i,4) \cdot (ih\_load^2) + \\
    2 \cdot Pos_{Old}(i,5) \cdot V_{Old}(i,5) \cdot (ir\_load^2) + \\
    2 \cdot Pos_{Old}(i,6) \cdot V_{Old}(i,6) \cdot (iu\_load^2); \]
\[ C22 = (Pos_{Old}(i,4) \cdot ih\_load)^2 + (Pos_{Old}(i,5) \cdot ir\_load)^2 + \\
    (Pos_{Old}(i,6) \cdot iu\_load)^2 - ((irms1 \cdot 0.95)^2); \]
\[ \delta_2 = B22^2 - 4 \cdot A22 \cdot C22; \]
\[ \text{if (A22==0) % not quadratic function} \]
\[ \text{if (B22 == 0)} \]
\[ x5 = 1; \]
\[ \text{elseif (B22 > 0)} \]
\[ \text{if } \left(\frac{-C22}{B22}\right) > 1 \]
\[ x5 = 1; \]
\[ \text{else} \]
\[ x5 = \left(\frac{-C22}{B22}\right); \]
\[ \text{end} \]
\[ \text{elseif (B22 < 0)} \]
\[ x5 = 1; \]
\[ \text{end} \]
\[ \text{elseif (A22 > 0) % parabola opens upward} \]
\[ \text{if (delta2 < 0)} \]
\[ Max_a = 0; \]
\[ \text{elseif (delta2 > 0)} \]
\[ x51 = \left(\frac{-B22 - \sqrt{\delta_2}}{2 \cdot A22}\right); \]
\[ x52 = \left(\frac{-B22 + \sqrt{\delta_2}}{2 \cdot A22}\right); \]
\[ \text{if (x51 > 1)} \]
\[ Max_a = 0; \]
\[ \text{elseif ((x51 <= 1) \&\& (x52 >= 1))} \]
\[ x5 = x51; \]
\[ \text{elseif ((x52 >= 0) \&\& (x52 <= 1))} \]
\[ x5 = x52; \]
\[ \text{elseif (x52 < 0)} \]
\[ Max_a = 0; \]
\[ \text{end} \]
\[ \text{elseif (delta2 == 0)} \]
\[ x512 = \left(\frac{-B22 + \sqrt{\delta_2}}{2 \cdot A22}\right); \]
\[ \text{if (x512 > 1)} \]
\[ Max_a = 0; \]
\[ \text{elseif (x512 < 0)} \]
\[ Max_a = 0; \]
\[ \text{else} \]
\[ x5 = x512; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end % END if (delta2 < 0)} \]
\[ \text{if (Max_a == 0) % Solve Constraints} \]
\[ \text{end % END if (Max_a ~= 0) % Current Limit PEBB1} \]

%%% 
\[ \text{if (Max_a ~= 0)} \]
\[ A33 = ((-V_{Old}(i,1)-V_{Old}(i,4)) \cdot ih\_load)^2 + ((-V_{Old}(i,2)-V_{Old}(i,5)) \cdot ir\_load)^2 + \\
    ((-V_{Old}(i,3)-V_{Old}(i,6)) \cdot iu\_load)^2; \]
\[ B_{33} = 2*(1-\text{PosOld}(i,1)-\text{PosOld}(i,4))*(\text{VOld}(i,1) - \text{VOld}(i,4))*(\text{ih}_\text{load}^2) + 2*(1-\text{PosOld}(i,2)-\text{PosOld}(i,5))*(\text{VOld}(i,2) - \text{VOld}(i,5))*(\text{ir}_\text{load}^2) + 2*(1-\text{PosOld}(i,3)-\text{PosOld}(i,6))*(\text{VOld}(i,3) - \text{VOld}(i,6))*(\text{iu}_\text{load}^2); \]

\[ C_{33} = ((1-\text{PosOld}(i,1)-\text{PosOld}(i,4))*\text{ih}_\text{load})^2 + ((1-\text{PosOld}(i,2)-\text{PosOld}(i,5))*\text{ir}_\text{load})^2 + ((1-\text{PosOld}(i,3)-\text{PosOld}(i,6))*\text{iu}_\text{load})^2 - ((\text{irms}2*0.95)^2); \]

\[ \delta_{33} = B_{33}^2 - 4*A_{33}*C_{33}; \]

\[
\text{if (A33==0)} \quad \% \text{not quadratic function}
\text{if (B33 == 0)}
\quad x6=1;
\text{elseif (B33 > 0)}
\quad \text{if (}-C_{33}/B_{33}>1)\]
\quad x6=1;
\text{else}
\quad x6 = (C33/B33);
\text{end}
\text{elseif (B33 <0)}
\quad x6=1;
\text{end}
\text{elseif (A33>0)} \quad \% \text{parabola opens upward}
\text{if (delta3 < 0)}
\quad \text{Max_a = 0;}
\text{elseif (delta3 > 0)}
\quad x61=(-B_{33}-\text{sqrt(delta3)})/(2*A_{33});
\quad x62=(-B_{33}+\text{sqrt(delta3)})/(2*A_{33});
\quad \text{if (x61>1)}
\quad \text{Max_a = 0;}
\text{elseif ((x61 <= 1) && (x62 >= 1))}
\quad x6 = 1;
\text{elseif ((x62 >= 0) && (x62 <= 1))}
\quad x6 = x62;
\text{elseif (x62 < 0)}
\quad \text{Max_a = 0;}
\text{end}
\text{elseif (delta3 == 0)}
\quad x612 = (-B_{33}/(2*A_{33}));
\quad \text{if (x612 > 1)}
\quad \text{Max_a = 0;}
\text{elseif (x612 < 0)}
\quad \text{Max_a = 0;}
\text{else}
\quad x6 = x612;
\text{end}
\text{end}
\text{END if (A33==0)} \quad \% \text{Solve Constraints}
\text{end \% END if (Max_a ~= 0)} \quad \% \text{Current Limit PEBB2}

\%
\% Decide Max_a,if all the constraints have roots, With
\text{Max_a~==0}
\text{if (Max_a ~== 0)}
Minxm1 = min(xm1); % All K>=0;
Minxm2 = min(xm2); % All K<=1;
Minxm = min(Minxm1, Minxm2);

Min1 = min(Minxm, x1);
Min2 = min(Min1, x2); % Maximum from Kr+Kr1<=1
Min3 = min(Min2, x3);
Min4 = min(Min3, x4);
Min5 = min(Min4, x5); % Maximum from Current Limit
Max_a = min(Min5, x6);
A44 = Max_a*10;
A44_floor = floor(A44);
Max_a = A44_floor/10;
end % END if (Max_a ~= 0) % Get Max_a
if (Max_a < 0.0001)
    Max_a = 0;
else
    Max_a = Max_a - 0.0001;
end

for g=1:Dimen
    V(i,g) = Max_a * VOld(i,g);
    Pos(i,g) = PosOld(i,g) + V(i,g);
end

%% Check Constraints again;
Cons111 = (Pos(i,1) >=0);
Cons222 = (Pos(i,2) >=0);
Cons333 = (Pos(i,3) >=0);
Cons444 = (Pos(i,4) >=0);
Cons555 = (Pos(i,5) >=0);
Cons666 = (Pos(i,6) >=0);
Cons1111 = (Pos(i,1) <=1);
Cons2222 = (Pos(i,2) <=1);
Cons3333 = (Pos(i,3) <=1);
Cons4444 = (Pos(i,4) <=1);
Cons5555 = (Pos(i,5) <=1);
Cons6666 = (Pos(i,6) <=1);
Cons11 = ((Pos(i,1) + Pos(i,4)) <= 1);
Cons22 = ((Pos(i,2) + Pos(i,5)) <= 1);
Cons33 = ((Pos(i,3) + Pos(i,6)) <= 1);
Cons44 = (((Pos(i,1))*ih_load)^2+((Pos(i,2))*ir_load)^2+((Pos(i,3))*iu_load)^2) <= ((irms*0.95)^2));
Cons55 = (((Pos(i,4))*ih_load)^2+((Pos(i,5))*ir_load)^2+((Pos(i,6))*iu_load)^2) <= ((irms1*0.95)^2));
Cons66 = (((1-Pos(i,1)-Pos(i,4))*ih_load)^2+((1-Pos(i,2) -Pos(i,5))*ir_load)^2+((1-Pos(i,3)-Pos(i,6))*iu_load)^2) <= ((irms2*0.95)^2));
Const = (Cons111 && Cons222 && Cons333 && Cons444 && Cons555 && Cons666 && Cons1111 && Cons2222 && Cons3333 && Cons4444 &&
\begin{align*}
\text{resultCons\_Max} &= [k; i; While2; \text{Max}_a; \text{PosOld}(i,1); \text{VOld}(i,1); \text{PosOld}(i,2); \text{VOld}(i,2); \text{PosOld}(i,3); \text{VOld}(i,3); \\
&\quad \text{PosOld}(i,4); \text{VOld}(i,4); \text{PosOld}(i,5); \text{VOld}(i,5); \text{PosOld}(i,6); \text{VOld}(i,6); \\
&\quad \text{Cons1}; \text{Cons2}; \text{Cons3}; \text{Cons4}; \text{Cons5}; \text{Cons6};] \\
\text{ConsStore\_Max} &= [\text{ConsStore\_Max}, \text{resultCons\_Max}]; \\
\text{TestWhile2}(k, i) &= While2; \\
\text{if} (\text{Max}_a \neq 0) \\
\text{resultCons\_Max}_0 &= [k; i; While2; Max_a; \text{xml}(1); \text{xml}(2); \text{xml}(3); \text{xml}(4); \text{xml}(5); \text{xml}(6); \\
&\quad \text{xml}(2); \text{xml}(2); \text{xml}(3); \text{xml}(4); \text{xml}(5); \text{xml}(6); \\
&\quad x1; x2; x3; x4; x5; x6; \text{Pos}(i,1); \text{Pos}(i,2); \text{Pos}(i,3); \text{Pos}(i,4); \text{Pos}(i,5); \text{Pos}(i,6); \\
&\quad \text{Cons1; Cons2; Cons3; Cons4; Cons5; Cons6;};] \\
\text{ConsStore\_Max}_0 &= [\text{ConsStore\_Max}_0, \text{resultCons\_Max}_0]; \\
\text{end}
\end{align*}

\% Test if Max \_a calculated more than one time
\text{if} (\text{Const} == 1) \\
\text{if} (\text{While2} == 1) \\
\text{aal} &= [k; i]; \\
\text{Awhile1} &= [\text{Awhile1}, \text{aal}]; \\
\text{end} \\
\text{if} (\text{While2} == 2) \\
\text{aa2} &= [k; i]; \\
\text{Awhile2} &= [\text{Awhile2}, \text{aa2}]; \\
\text{end} \\
\text{if} (\text{While2} == 3) \\
\text{aa3} &= [k; i]; \\
\text{Awhile3} &= [\text{Awhile3}, \text{aa3}]; \\
\text{end} \\
\text{if} (\text{While2} == 4) \\
\text{aa34} &= [k; i]; \\
\text{Awhile34} &= [\text{Awhile34}, \text{aa34}]; \\
\text{end} \\
\text{end}
\text{if} (\text{While2} >= 10) \\
\text{break}; \\
\text{end} % END if (While2 >= 10)
\text{end} %END while (\text{Const} == 0) //Update Particles
\text{end} %END for i = 1:Pnum // update Each Particle V,P
\text{\% Another Stop Condition}
\text{for i = 1:Pnum}
for j = 1: Dimen
    if (Pos(i,j) == Pos(1,j))
        if ((abs(Pos(i,j) - Pos(1,j))) <= 0.001)
            Stop(i,j) = 1;
        else Stop(i,j) = 0;
    end
end
end % Calculate Stop(i,j)
M = sum(Stop);
Crit = sum(M);
if (Crit == (Pnum * 6))
    break;
end
end %END for k = 1:Iter_num //Iteration finished

countwhile1 = 0;
countwhile2 = 0;
countwhile3 = 0;
countwhile34 = 0;
countwhile10 = 0;
for mk = 1 : k
    for mk1 = 1 : Pnum
        if (TestWhile2(mk,mk1) == 1)
            countwhile1 = countwhile1+1;
        end
        if (TestWhile2(mk,mk1) == 2)
            countwhile2 = countwhile2+1;
        end
        if (TestWhile2(mk,mk1) == 3)
            countwhile3 = countwhile3+1;
        end
        if (TestWhile2(mk,mk1) == 10)
            countwhile10 = countwhile10+1;
        end
        if ((TestWhile2(mk,mk1) > 3) & (TestWhile2(mk,mk1) < 10))
            countwhile34 = countwhile34+1;
        end
    end
end
end % examine how while2 runs
CountWhile2Total = countwhile2 + countwhile3 + countwhile34;
save Simulink_Result.mat; % save the workspace
APPENDIX B

CODES IN LABWINDOWS/CVI
int CVICALLBACK AlgorithmCallback (int panel, int control, int event, 
void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
    int i1, j;
    int Cons_111, Cons_222, Cons_333, Cons_444, Cons_555, Cons_666, Cons_1111, Cons_2222, Cons_3333, Cons_4444, Cons_5555, Cons_6666;
    int Cons1, Cons2, Cons3, Cons4, Cons5, Cons6, Cons, While1;
    case EVENT_COMMIT:
    {
        // Initialization of PSO
        for (il=0; il<Pnum; il++)
        {
            Pos[i1].Pkh = (float)(Random (0, 1));
            Pos[i1].Pkh = ((floor(Pos[i1].Pkh * 1000))/1000);
            Pos[i1].Pkr = (float)(Random (0, 1));
            Pos[i1].Pkr = ((floor(Pos[i1].Pkr * 1000))/1000);
            Pos[i1].Pku = (float)(Random (0, 1));
            Pos[i1].Pku = ((floor(Pos[i1].Pku * 1000))/1000);
            Pos[i1].Pkh1 = (float)(Random (0, (1-Pos[i1].Pkh)));
            Pos[i1].Pkh1 = ((floor(Pos[i1].Pkh1 * 1000))/1000);
            Pos[i1].Pkr1 = (float)(Random (0, (1-Pos[i1].Pkr)));
            Pos[i1].Pkr1 = ((floor(Pos[i1].Pkr1 * 1000))/1000);
            Pos[i1].Pku1 = (float)(Random (0, (1-Pos[i1].Pku)));
            Pos[i1].Pku1 = ((floor(Pos[i1].Pku1 * 1000))/1000);
            Velocity[i1].Vkh = 0;
            Velocity[i1].Vkr = 0;
            Velocity[i1].Vku = 0;
            Velocity[i1].Vkh1 = 0;
            Velocity[i1].Vkr1 = 0;
            Velocity[i1].Vku1 = 0;
            // Check Constraints
            Cons_111 = (Pos[i1].Pkh >= 0);
            Cons_222 = (Pos[i1].Pkr >= 0);
            Cons_333 = (Pos[i1].Pku >= 0);
            Cons_444 = (Pos[i1].Pkh1 >= 0);
            Cons_555 = (Pos[i1].Pkr1 >= 0);
            Cons_666 = (Pos[i1].Pku1 >= 0);
            Cons_1111 = (Pos[i1].Pkh <= 1.0);
            Cons_2222 = (Pos[i1].Pkr <= 1.0);
            Cons_3333 = (Pos[i1].Pku <= 1.0);
            Cons_4444 = (Pos[i1].Pkh1 <= 1.0);
            Cons_5555 = (Pos[i1].Pkr1 <= 1.0);
            Cons_6666 = (Pos[i1].Pku1 <= 1.0);
            Cons1 = ((Pos[i1].Pkh + Pos[i1].Pkh1) <= 1.0);
            Cons2 = ((Pos[i1].Pkr + Pos[i1].Pkr1) <= 1.0);
            Cons3 = ((Pos[i1].Pku + Pos[i1].Pku1) <= 1.0);
Cons4 =
((pow(((Pos[i1].Pkh)*ih_load),2)+pow(((Pos[i1].Pkr)*ir_load),2)+pow(((Pos[i1].Pku)*iu_load),2)) <= (pow((irms*0.95),2)));
Cons5 =
((pow(((Pos[i1].Pkh1)*ih_load),2)+pow(((Pos[i1].Pkr1)*ir_load),2)+pow(((Pos[i1].Pku1)*iu_load),2)) <= (pow((irms1*0.95),2)));
Cons6 = ((pow(((1-Pos[i1].Pkh-Pos[i1].Pkh1)*ih_load),2)+pow(((1-Pos[i1].Pkr-Pos[i1].Pkr1)*ir_load),2)+pow(((1-Pos[i1].Pku-Pos[i1].Pku1)*iu_load),2)) <= (pow((irms2*0.95),2)));
Cons = (Cons_111 && Cons_222 && Cons_333 && Cons_444 && Cons_555 && Cons_666 && Cons_1111 && Cons_2222 && Cons_3333 && Cons_4444 && Cons_5555 && Cons_6666 && Cons1 && Cons2 && Cons3 && Cons4 && Cons5 && Cons6);//It must satisfy all the constraints
IniCons[i1] = Cons;//Record of initialization
if (Cons == 1)
{
 IniCons[i1] = 777;
}
if (Cons == 0)
{
 INIequalzero = INIequalzero +1;
}
While1= 0;
// Regenerate the particle when violate the constraints
while (Cons == 0) //&& (While1 <= 2))
{
 While1 = While1 +1;
 Pos[i1].Pkh = (float)(Random (0, 1));
 Pos[i1].Pkh = ((floor(Pos[i1].Pkh * 1000))/1000);
 Pos[i1].Pkr = (float)(Random (0, 1));
 Pos[i1].Pkr = ((floor(Pos[i1].Pkr * 1000))/1000);
 Pos[i1].Pku = (float)(Random (0, 1));
 Pos[i1].Pku = ((floor(Pos[i1].Pku * 1000))/1000);
 Pos[i1].Pkh1 = (float)(Random (0, (1-
 Pos[i1].Pkh)));
 Pos[i1].Pkh1 = ((floor(Pos[i1].Pkh1 * 1000))/1000);
 Pos[i1].Pkr1 = (float)(Random (0, (1-
 Pos[i1].Pkr)));
 Pos[i1].Pkr1 = ((floor(Pos[i1].Pkr1 * 1000))/1000);
 Pos[i1].Pku1 = (float)(Random (0, (1-
 Pos[i1].Pku)));
 Pos[i1].Pku1 = ((floor(Pos[i1].Pku1 * 1000))/1000);

// Check Constraints again;
Cons_111 = (Pos[i1].Pkh >=0);
Cons_222 = (Pos[i1].Pkr >=0);
Cons_333 = (Pos[i1].Pku >=0);
Cons_444 = (Pos[i1].Pkh1 >=0);
Cons_555 = (Pos[i1].Pkr1 >=0);
Cons_666 = (Pos[i1].Pku1 >= 0);
Cons_1111 = (Pos[i1].Pkh <= 1);
Cons_2222 = (Pos[i1].Pkr <= 1);
Cons_3333 = (Pos[i1].Pku <= 1);
Cons_4444 = (Pos[i1].Pkh1 <= 1);
Cons_5555 = (Pos[i1].Pkr1 <= 1);
Cons_6666 = (Pos[i1].Pku1 <= 1);
Cons1 = ((Pos[i1].Pkh + Pos[i1].Pkh1) <= 1);
Cons2 = ((Pos[i1].Pkr + Pos[i1].Pkr1) <= 1);
Cons3 = ((Pos[i1].Pku + Pos[i1].Pku1) <= 1);
Cons4 =
((pow(((Pos[i1].Pkh)*ih_load),2)+pow(((Pos[i1].Pkr)*ir_load),2)+pow(((Pos[i1].Pku)*iu_load),2)) <= (pow((irms*0.95),2)));
Cons5 =
((pow(((Pos[i1].Pkh1)*ih_load),2)+pow(((Pos[i1].Pkr1)*ir_load),2)+pow(((Pos[i1].Pku1)*iu_load),2)) <= (pow((irms1*0.95),2)));
Cons6 = ((pow(((1-Pos[i1].Pkh-Pos[i1].Pkh1)*ih_load),2)+pow(((1-Pos[i1].Pkr-Pos[i1].Pkr1)*ir_load),2)+pow(((1-Pos[i1].Pku-Pos[i1].Pku1)*iu_load),2)) <= (pow((irms2*0.95),2)));
Cons = (Cons_111 && Cons_222 && Cons_333 && Cons_444 && Cons_555 && Cons_666 && Cons_1111 && Cons_2222 && Cons_3333 && Cons_4444 && Cons_5555 && Cons_6666 && Cons1 && Cons2 && Cons3 && Cons4 && Cons5 && Cons6); //It must satisfy all the constraints
if (While1 == 10) {
    break;
}
} //END while (Cons == 0) //Initialization
if (While1 == 1) {
    INIOneTimeMaxA = INIOneTimeMaxA + 1;
} if (While1 == 2) {
    INITwoTimeMaxA = INITwoTimeMaxA + 1;
} if (While1 == 3) {
    INIThreeTimeMaxA = INITthreeTimeMaxA + 1;
} if (While1 == 4) {
    INIFourTimeMaxA = INIFourTimeMaxA + 1;
} if (While1 == 5) {
    INIFiveTimeMaxA = INIFiveTimeMaxA + 1;
} if (While1 == 6) {
    INISixTimeMaxA = INISixTimeMaxA + 1;
}
if (While1 == 7)
{
    INISevenTimeMaxA = INISevenTimeMaxA + 1;
}
if (While1 == 8)
{
    INIEightTimeMaxA = INIEightTimeMaxA + 1;
}
if (While1 == 9)
{
    ININineTimeMaxA = ININineTimeMaxA + 1;
}
if (While1 == 10)
{
    INITenTimeMaxA = INITenTimeMaxA + 1;
}
} //END for (i=0; i<Pnum; i++) // Initialization for each particle

//GlobalBest & LocalBest Initialization
Gbest = 1000000; //Initialize global best
for (i1=0; i1<Pnum; i1++)
{
    Lbest[i1].Fvalue = 1000000;
} //END of for (i=0; i<Pnum, i++) //Initialize global best

Cn_Param[0].Kh=(unsigned short)(Pos[0].Pkh*Kh_To_Fix);
Cn_Param[0].Kr=(unsigned short)(Pos[0].Pkr*Kr_To_Fix);
Cn_Param[0].Ku=(unsigned short)(Pos[0].Pku*Ku_To_Fix);
Cn_Param[1].Kh=(unsigned short)(Pos[0].Pkh1*Kh_To_Fix);
Cn_Param[1].Kr=(unsigned short)(Pos[0].Pkr1*Kr_To_Fix);
Cn_Param[1].Ku=(unsigned short)(Pos[0].Pku1*Ku_To_Fix);
Cn_Param[2].Kh=(unsigned short)((1-Pos[0].Pkh-Pos[0].Pkh1)*Kh_To_Fix);
Cn_Param[2].Kr=(unsigned short)((1-Pos[0].Pkr-Pos[0].Pkr1)*Kr_To_Fix);
Cn_Param[2].Ku=(unsigned short)((1-Pos[0].Pku-Pos[0].Pku1)*Ku_To_Fix);

Wk_Mode_PanToStr(); // Get PEBB's working mode from setting panel
PSO_Cn_Para_PanToStr(); // Get PEBB's control parameters from setting panel

//Cn_Para_PanToStr();
for (j=0; j<3; j++)
{
    SPrtc_Pck_Para(j); // Send PEBB Parameters Down Load package to PEBB
} // End of for (j=0; j<3; j++)
SPrtc_Pck_DCControl(0);
} // END of Case EVENT_COMMIT:
break;
} // END of switch (event)
return 0;
}

//====================================================================
========
// Receiving Action: Translate the Received package and act
accordingly //
// Arguments- i i=0, for PEBB_1
//
// i=1, for PEBB_2
//
// i=2, for PEBB_3
//
// - *R_Buffer The place the package saved in
//====================================================================
========
void Rec_Action(int i, char *R_Buffer) //
{
    int k, j;
    int j1;
    unsigned int PSize, TI, Temp1, Temp2;
    union Prtc_PCKU IR_Buffer;

    if(R_Buffer[0] != 0x68) return; // Package got error, then
    return
    TI = (unsigned int)R_Buffer[6]; // Get type identification
    k = TI & 0x00F0;
    if (k!=0x50)PSize = (unsigned int)R_Buffer[1]+2;// get package size
    else PSize = WV_Length*2 + 10;
    for (k=0; k<PSize; k++) IR_Buffer.Bit8[k] = R_Buffer[k];
    //
    switch (TI) {
    case 0x6F: // Sent Working Mode and System parameters by PEBB
        Wk_mode[i].all = IR_Buffer.Bit16[5];
        Cn_Param[i].Kh = IR_Buffer.Bit16[6];
        Cn_Param[i].Kr = IR_Buffer.Bit16[7];
        Cn_Param[i].Ku = IR_Buffer.Bit16[8];
        Cn_Param[i].Ka = IR_Buffer.Bit16[9];
        Cn_Param[i].IKp = IR_Buffer.Bit16[10];
        Cn_Param[i].IKi = IR_Buffer.Bit16[11];
        Cn_Param[i].VKp = IR_Buffer.Bit16[12];
        Cn_Param[i].VKi = IR_Buffer.Bit16[13];
        Cn_Param[i].VKd = IR_Buffer.Bit16[14];


Wk_Mode_StrToPan(i); // modify the working mode on setting panel
Cn_Para_StrToPan(i); // modify the parameters on setting panel
break;
case 0x07: // Sent Working and Error State by PEBB
    Wk_State[i].all = IR_Buffer.Bit16[5]; // working state
    Temp1 = (unsigned int)IR_Buffer.Bit16[6];
    Temp2 = (unsigned int)IR_Buffer.Bit16[7];
    Err_State[i].all = (Temp1 & 0xFFFF)+((Temp2 & 0xFFFF)<<16);
    Wks_To_Panel(i);// modify working state on control panel
    break;
case 0x0B: // Sent Working Analog Values by PEBB
    An_Value[i].DC_Voltage = IR_Buffer.Bit16[5];
    An_Value[i].AN1_Val = IR_Buffer.Bit16[6];
    An_Value[i].AN2_Val = IR_Buffer.Bit16[7];
    An_Value[i].AN3_Val = IR_Buffer.Bit16[8];
    An_Value[i].AN4_Val = IR_Buffer.Bit16[9];
    An_Value[i].AN5_Val = IR_Buffer.Bit16[10];
    An_Value[i].AN6_Val = IR_Buffer.Bit16[11];
    An_Value[i].AN7_Val = IR_Buffer.Bit16[12];
    An_Value[i].AN8_Val = IR_Buffer.Bit16[13];
    Wks_To_Panel(i); // Write from Working state union to panel
    ANs_To_Panel(i); // Write analog Value from structure to panel
    break;
case 0x29: // Sent all by PEBB
    Wk_State[i].all = IR_Buffer.Bit16[5]; // working state
    Temp1 = (unsigned int)IR_Buffer.Bit16[6];
    Temp2 = (unsigned int)IR_Buffer.Bit16[7];
    Err_State[i].all = (Temp1 & 0xFFFF)+((Temp2 & 0xFFFF)<<16);
    An_Value[i].DC_Voltage = IR_Buffer.Bit16[8];
    An_Value[i].AN1_Val = IR_Buffer.Bit16[9];
    An_Value[i].AN2_Val = IR_Buffer.Bit16[10];
    An_Value[i].AN3_Val = IR_Buffer.Bit16[11];
    An_Value[i].AN4_Val = IR_Buffer.Bit16[12];
    An_Value[i].AN5_Val = IR_Buffer.Bit16[13];
    An_Value[i].AN6_Val = IR_Buffer.Bit16[14];
    An_Value[i].AN7_Val = IR_Buffer.Bit16[15];
    An_Value[i].AN8_Val = IR_Buffer.Bit16[16];
    Wks_To_Panel(i); // Write from Working state union to panel
    ANs_To_Panel(i); // Write analog Value from structure to panel
    break;
case 0x71: // Sent DC voltage and control loop output by PEBB
    An_Value[i].DC_Voltage = IR_Buffer.Bit16[5];
    MeasuredCost[i].DCSignal = (((float)IR_Buffer.Bit16[6])/256);
    flagreceive[CountPnum][i] = 1; //Temp2 = (unsigned
    int)IR_Buffer.Bit16[7];
    Wks_To_Panel(i); // modify working state on control panel
    if ((flagreceive[CountPnum][i]==1))
    {
        i++;
    }
if (i<3)
{
    SPrtc_Pck_DCControl(i);
}

if ((i==3) && (flagreceive[CountPnum][2]==1)) //IF 1
{
    CostValue[CountPnum].Fvalue = (MeasuredCost[0].DCSignal +
    MeasuredCost[1].DCSignal + MeasuredCost[2].DCSignal);
    CostValue[CountPnum].Fvalue =
    (floor(100000*CostValue[CountPnum].Fvalue))/10;

    //Update LocalBest
    if (CostValue[CountPnum].Fvalue < Lbest[CountPnum].Fvalue)
        //IF 1.1
        {
            Lbest[CountPnum].Fvalue = CostValue[CountPnum].Fvalue;
            Pos_Lbest[CountPnum].Pkh = Pos[CountPnum].Pkh;
            Pos_Lbest[CountPnum].Pkr = Pos[CountPnum].Pkr;
            Pos_Lbest[CountPnum].Pku = Pos[CountPnum].Pku;
            Pos_Lbest[CountPnum].Pkh1 = Pos[CountPnum].Pkh1;
            Pos_Lbest[CountPnum].Pkr1 = Pos[CountPnum].Pkr1;
            Pos_Lbest[CountPnum].Pku1 = Pos[CountPnum].Pku1;
        } //END of if (CostValue[i].Fvalue < Lbest[i].Fvalue)

    if (CountPnum<(Pnum)) //IF 1.2
        {
            CountPnum++;
            if (CountPnum<Pnum) //IF 1.2.1
                {
                    Cn_Param[0].Kh = (unsigned short)(Pos[CountPnum].Pkh*Kh_To_Fix);
                    Cn_Param[0].Kr = (unsigned short)(Pos[CountPnum].Pkr*Kr_To_Fix);
                    Cn_Param[0].Ku = (unsigned short)(Pos[CountPnum].Pku*Ku_To_Fix);
                    Cn_Param[1].Kh = (unsigned short)(Pos[CountPnum].Pkh1*Kh_To_Fix);
                    Cn_Param[1].Kr = (unsigned short)(Pos[CountPnum].Pkr1*Kr_To_Fix);
                    Cn_Param[1].Ku = (unsigned short)(Pos[CountPnum].Pku1*Ku_To_Fix);
                    Cn_Param[2].Kh = (unsigned short)((1-
                    Pos[CountPnum].Pkh-Pos[CountPnum].Pkh1)*Kh_To_Fix);
                    Cn_Param[2].Kr = (unsigned short)((1-
                    Pos[CountPnum].Pkr-Pos[CountPnum].Pkr1)*Kr_To_Fix);
                    Cn_Param[2].Ku = (unsigned short)((1-
                    Pos[CountPnum].Pku-Pos[CountPnum].Pku1)*Ku_To_Fix);
                    Wk_Mode_PanToStr();// Get PEBB's working mode from
                    setting panel
PSO_Cn_Para_PanToStr(); // Get PEBB's control parameters from setting panel

for (j1=0; j1<3; j1++)
{
   SPrtc_Pck_Para(j1); //Send PEBB Parameters Down
}

Load package to PEBB

} // End of for(j=0; j<3; j++)
SPrtc_Pck_DCControl(0);

} //End of if (CountPnum<Pnum) // IF 1.2.1

} // End of if (CountPnum<(Pnum)) //IF1.2

} //End of if ((i==3) && (flagreceive[CountPnum][2]==1))

//Iteration

if((CountPnum==(Pnum))&&(flagreceive[CountPnum-1][2]==1))
//next iteration IF 2
{

MinCost = CostValue[0].Fvalue;
index = 0;
for (j1=1; j1<Pnum; j1++)
{
   if (CostValue[j1].Fvalue < MinCost) //IF 2.2.1
   {
      MinCost = CostValue[j1].Fvalue;
      index = j1;
   }
}
// END of for (j1=1; j1<Pnum; j1++)

if (MinCost<Gbest) //IF 2.2.2
{

Gbest = MinCost;
Pos_Gbest.Pkh = Pos[index].Pkh;
Pos_Gbest.Pkr = Pos[index].Pkr;
Pos_Gbest.Pku = Pos[index].Pku;
Pos_Gbest.Pkh1 = Pos[index].Pkh1;
Pos_Gbest.Pkr1 = Pos[index].Pkr1;
Pos_Gbest.Pku1 = Pos[index].Pku1;
} //End of if (MinCost<Gbest) Update Gbest, Pos_Gbest

if (iter>IterNum) //IF 2.1
{

Cn_Param[0].Kh=(unsigned short)(Pos_Gbest.Pkh*Kh_To_Fix);
Cn_Param[0].Kr=(unsigned short)(Pos_Gbest.Pkr*Kr_To_Fix);
Cn_Param[0].Ku=(unsigned short)(Pos_Gbest.Pku*Ku_To_Fix);
Cn_Param[1].Kh=(unsigned short)(Pos_Gbest.Pkh1*Kh_To_Fix);
Cn_Param[1].Kr=(unsigned short)(Pos_Gbest.Pkr1*Kr_To_Fix);
Cn_Param[1].Ku=(unsigned short)(Pos_Gbest.Pku1*Ku_To_Fix);
Cn_Param[2].Kh=(unsigned short)((1-Pos_Gbest.Pkh-Pos_Gbest.Pkh1)*Kh_To_Fix);

\[\text{Cn\_Param[2].Kr} = (\text{unsigned short})\left((1 - \text{Pos}\_\text{Gbest}\.Pkr - \text{Pos}\_\text{Gbest}\.Pkr1) \times \text{Kr\_To\_Fix}\right);\]
\[\text{Cn\_Param[2].Ku} = (\text{unsigned short})\left((1 - \text{Pos}\_\text{Gbest}\.Pku - \text{Pos}\_\text{Gbest}\.Pku1) \times \text{Ku\_To\_Fix}\right);\]

\[\text{Wk\_Mode\_PanToStr()};\] // Get PEBB's working mode from setting panel

\[\text{PSO\_Cn\_Para\_PanToStr()};\] // Get PEBB's control parameters from setting panel

\[\text{for (j1=0; j1<3; j1++)}\]
\[\{\text{SPrtc\_Pck\_Para(j1)};\] // Send PEBB Parameters Down

\[\text{Load package to PEBB}\]
\[\} \] // End of for(j=0; j<3; j++)
\[\} \] // End of if (iter=IterNum) // IF2.1

\[\text{else}\]
\[\{\text{iter++;}\]
\[\text{for (j1=0; j1<Pnum; j1++)}\]
\[\{\text{flagreceive[j1][0]=0;}\]
\[\text{flagreceive[j1][1]=0;}\]
\[\text{flagreceive[j1][2]=0;}\]

\[\} \] // Update Velocity and Position

\[\text{for (j=0; j<Pnum; j++)} \] //
\[\{\]
\[\text{//W3} = (\text{Wstart}-\text{Wend})^{(\text{IterNum-iter})/\text{IterNum}} + \text{Wend};\]
\[\text{W3} = \text{Wend}^{(\text{IterNum-iter})/\text{IterNum}};\]
\[\text{r1} = (\text{float})(\text{Random} \ (0, \ 1));\]
\[\text{r2} = (\text{float})(\text{Random} \ (0, \ 1));\]
\[\text{Velocity[j].Vkh} = \text{W3} \times \text{Velocity[j].Vkh} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pkh - \text{Pos}[j].Pkh) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pkh} - \text{Pos}[j].Pkh);\]
\[\text{Velocity[j].Vkr} = \text{W3} \times \text{Velocity[j].Vkr} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pkr - \text{Pos}[j].Pkr) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pkr} - \text{Pos}[j].Pkr);\]
\[\text{Velocity[j].Vku} = \text{W3} \times \text{Velocity[j].Vku} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pku - \text{Pos}[j].Pku) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pku} - \text{Pos}[j].Pku);\]
\[\text{Velocity[j].Vkh1} = \text{W3} \times \text{Velocity[j].Vkh1} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pkh1 - \text{Pos}[j].Pkh1) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pkh1} - \text{Pos}[j].Pkh1);\]
\[\text{Velocity[j].Vkr1} = \text{W3} \times \text{Velocity[j].Vkr1} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pkr1 - \text{Pos}[j].Pkr1) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pkr1} - \text{Pos}[j].Pkr1);\]
\[\text{Velocity[j].Vku1} = \text{W3} \times \text{Velocity[j].Vku1} + \text{c1} \times r1 \times (\text{Pos}_{L\text{best}[j]}.Pku1 - \text{Pos}[j].Pku1) + \text{c2} \times r2 \times (\text{Pos}_{G\text{best}.Pku1} - \text{Pos}[j].Pku1);\]
Velocity[j].Vkh = ((floor(Velocity[j].Vkh * 1000))/1000);
Velocity[j].Vkr = ((floor(Velocity[j].Vkr * 1000))/1000);
Velocity[j].Vku = ((floor(Velocity[j].Vku * 1000))/1000);
Velocity[j].Vkh1 = ((floor(Velocity[j].Vkh1 * 1000))/1000);
Velocity[j].Vkr1 = ((floor(Velocity[j].Vkr1 * 1000))/1000);
Velocity[j].Vku1 = ((floor(Velocity[j].Vku1 * 1000))/1000);
PosOld[j].Pkh = Pos[j].Pkh;
PosOld[j].Pkr = Pos[j].Pkr;
PosOld[j].Pku = Pos[j].Pku;
PosOld[j].Pkh1 = Pos[j].Pkh1;
PosOld[j].Pkr1 = Pos[j].Pkr1;
PosOld[j].Pku1 = Pos[j].Pku1;
Pos[j].Pkh = PosOld[j].Pkh + Velocity[j].Vkh;
Pos[j].Pku = PosOld[j].Pku + Velocity[j].Vku;
Pos[j].Pkh1 = PosOld[j].Pkh1 + Velocity[j].Vkh1;
Pos[j].Pkr1 = PosOld[j].Pkr1 + Velocity[j].Vkr1;
Pos[j].Pku1 = PosOld[j].Pku1 + Velocity[j].Vku1;

// Check Constraints
Cons11 = (Pos[j].Pkh >=0);
Cons22 = (Pos[j].Pkr >=0);
Cons33 = (Pos[j].Pku >=0);
Cons44 = (Pos[j].Pkh1 >=0);
Cons55 = (Pos[j].Pkr1 >=0);
Cons66 = (Pos[j].Pku1 >=0);
Cons111 = (Pos[j].Pkh <=1.0);
Cons222 = (Pos[j].Pkr <=1.0);
Cons333 = (Pos[j].Pku <=1.0);
Cons444 = (Pos[j].Pkh1 <=1.0);
Cons555 = (Pos[j].Pkr1 <=1.0);
Cons666 = (Pos[j].Pku1 <=1.0);
Cons11Value = Pos[j].Pkh + Pos[j].Pkh1;
Cons11 = (Cons11Value <= 1.0);
Cons22Value = Pos[j].Pkr + Pos[j].Pkr1;
Cons22 = (Cons22Value <= 1.0);
Cons33Value = Pos[j].Pku + Pos[j].Pku1;
Cons33 = (Cons33Value <= 1.0);
Cons44Value = (pow(((Pos[j].Pkh)*ih_load),2)+pow(((Pos[j].Pkr)*ir_load),2)+pow(((Pos[j].Pku)*iu_load),2));
Cons44 = (Cons44Value <= (pow((irms*0.95),2)));
Cons55Value = (pow(((Pos[j].Pkh1)*ih_load),2)+pow(((Pos[j].Pkr1)*ir_load),2)+pow(((Pos[j].Pku1)*iu_load),2));
Cons55 = (Cons55Value <= (pow((irms1*0.95),2)));
Cons66Value = (pow(((1-Pos[j].Pkh-Pos[j].Pkh1)*ih_load),2)+pow(((1-Pos[j].Pkr-Pos[j].Pkr1)*ir_load),2)+pow(((1-Pos[j].Pku-Pos[j].Pku1)*iu_load),2));
Cons66 = (Cons66Value <= (pow((irms2*0.95),2)));  
const = (Cons111 && Cons222 && Cons333 && Cons444  
&& Cons555 && Cons666 && Cons1111 && Cons2222 && Cons3333 && Cons4444  
&& Cons5555 && Cons6666 && Cons11 && Cons22 && Cons33 && Cons44 &&  
Cons55 && Cons66); //It must satisfy all the constraints
While2 = 0;
if (const == 0)
{
    VelocityOld[j].Vkh = Velocity[j].Vkh;
    VelocityOld[j].Vkr = Velocity[j].Vkr;
    VelocityOld[j].Vku = Velocity[j].Vku;
    VelocityOld[j].Vkh1 = Velocity[j].Vkh1;
    VelocityOld[j].Vkr1 = Velocity[j].Vkr1;
    VelocityOld[j].Vku1 = Velocity[j].Vku1;
    equalzero = equalzero +1;
}
while (const == 0)
{
    float Max_a;
    float x1,x2,x3,x4,x5,x6,la1,la2,lb1,lb2,lb3;
    float A11,B11,C11,delta1,x41,x42,x412;
    float A22,B22,C22,delta2,x51,x52,x512;
    float A33,B33,C33,delta3,x61,x62,x612;
    While2 = While2+1;
    Max_a = 1;
    // First Each (Kh,Kr,Ku,Kh1,Kr1,Ku >=0)
    if (VelocityOld[j].Vkh >= 0) //Kh>=0
    {
        xm1.Vkh = 1.0;
    }
    if (VelocityOld[j].Vkh < 0)
    {
        if ((-PosOld[j].Pkh/VelocityOld[j].Vkh) <=1)
        {
            xm1.Vkh = -PosOld[j].Pkh/VelocityOld[j].Vkh;
        }
        else
        {
            xm1.Vkh = 1;
        }
    } // END of Kh>=0
    // Kr>=0
    if (VelocityOld[j].Vkr >= 0)
    {
        xm1.Vkr = 1.0;
    }
    if (VelocityOld[j].Vkr < 0)
    {
        if ((-PosOld[j].Pkr/VelocityOld[j].Vkr) <=1)
\{ 
    xml.Vkr = -
    PosOld[j].Pkr/VelocityOld[j].Vkr;
    } 
else 
    { 
    xml.Vkr = 1;
    } 
} // END of Kr>=0

//Ku>=0
if (VelocityOld[j].Vku >= 0) 
{ 
    xml.Vku = 1.0;
} 
if (VelocityOld[j].Vku < 0)
{ 
    if ((-PosOld[j].Pku/VelocityOld[j].Vku) <=1) 
    { 
    xml.Vku = -
    PosOld[j].Pku/VelocityOld[j].Vku;
    } 
else 
    { 
    xml.Vku = 1;
    } 
} // END of Ku>=0

//Kh1>=0
if (VelocityOld[j].Vkh1 >= 0) 
{ 
    xml.Vkh1 = 1.0;
} 
else 
{ 
    if ((-PosOld[j].Pkh1/VelocityOld[j].Vkh1) <=1) 
    { 
    xml.Vkh1 = -
    PosOld[j].Pkh1/VelocityOld[j].Vkh1;
    } 
else 
    { 
    xml.Vkh1 = 1;
    } 
} // END of Kh1>=0

//Kr1>=0
if (VelocityOld[j].Vkr1 >= 0) 
{ 
    xml.Vkr1 = 1.0;
} 
else 
89
if ((-PosOld[j].Pkr1/VelocityOld[j].Vkr1) <= 1)
{
  x1m.Vkr1 = -PosOld[j].Pkr1/VelocityOld[j].Vkr1;
} else
{
  x1m.Vkr1 = 1;
} // END of Kr1>=0

if (VelocityOld[j].Vku1 >= 0)
{
  x1m.Vku1 = 1.0;
} else
{
  if ((-PosOld[j].Pku1/VelocityOld[j].Vku1) <= 1)
  {
    x1m.Vku1 = -PosOld[j].Pku1/VelocityOld[j].Vku1;
  } else
  {
    x1m.Vku1 = 1;
  }
} // END of Ku1>=0

// Second Each (Kh,Kr,Ku,Kh1,Kr1,Ku <= 1)
if (VelocityOld[j].Vkh == 0)
{
  x2m.Vkh = 1.0;
} if (VelocityOld[j].Vkh > 0)
{
  if (((1 - PosOld[j].Pkh)/VelocityOld[j].Vkh) > 1)
  {
    x2m.Vkh = 1;
  } else
  {
    x2m.Vkh = (1 - PosOld[j].Pkh)/VelocityOld[j].Vkh;
  }
} if (VelocityOld[j].Vkh < 0)
{
  x2m.Vkh = 1;
}
if (VelocityOld[j].Vkr == 0) {
    xm2.Vkr = 1.0;
} if (VelocityOld[j].Vkr > 0) {
    if (((1 - PosOld[j].Pkr)/VelocityOld[j].Vkr) > 1) {
        xm2.Vkr = 1;
    } else {
        xm2.Vkr = (1 - PosOld[j].Pkr)/VelocityOld[j].Vkr;
    }
} if (VelocityOld[j].Vkr < 0) {
    // END of Kr<=1
}

if (VelocityOld[j].Vku == 0) {
    xm2.Vku = 1.0;
} if (VelocityOld[j].Vku > 0) {
    if (((1 - PosOld[j].Pku)/VelocityOld[j].Vku) > 1) {
        xm2.Vku = 1;
    } else {
        xm2.Vku = (1 - PosOld[j].Pku)/VelocityOld[j].Vku;
    }
} if (VelocityOld[j].Vku < 0) {
    // END of Ku<=1
}

if (VelocityOld[j].Vkh1 == 0) {
    xm2.Vkh1 = 1.0;
} if (VelocityOld[j].Vkh1 > 0)
\[
\begin{align*}
\text{if } ((1 - \text{PosOld}[j].Pkh1)/\text{VelocityOld}[j].Vkh1) > 1) \\
\quad \text{ \quad xm2.Vkh1 = 1;} \\
\text{else} \\
\quad \text{ \quad xm2.Vkh1 = (1 - \text{PosOld}[j].Pkh1)/\text{VelocityOld}[j].Vkh1;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{VelocityOld}[j].Vkh1 < 0) \\
\quad \text{ \quad xm2.Vkh1 = 1;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{VelocityOld}[j].Vkr1 == 0) \\
\quad \text{ \quad xm2.Vkr1 = 1.0;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{VelocityOld}[j].Vkr1 > 0) \\
\quad \text{ \quad if } ((1 - \text{PosOld}[j].Pkr1)/\text{VelocityOld}[j].Vkr1) > 1) \\
\quad \quad \text{ \quad xm2.Vkr1 = 1;} \\
\quad \text{else} \\
\quad \quad \text{ \quad xm2.Vkr1 = (1 - \text{PosOld}[j].Pkr1)/\text{VelocityOld}[j].Vkr1;} \\
\text{if } \text{VelocityOld}[j].Vkr1 < 0) \\
\quad \text{ \quad xm2.Vkr1 = 1;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{VelocityOld}[j].Vku1 == 0) \\
\quad \text{ \quad xm2.Vku1 = 1.0;} \\
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{VelocityOld}[j].Vku1 > 0) \\
\quad \text{ \quad if } ((1 - \text{PosOld}[j].Pku1)/\text{VelocityOld}[j].Vku1) > 1) \\
\quad \quad \text{ \quad xm2.Vku1 = 1;} \\
\quad \text{else} \\
\quad \quad \text{ \quad xm2.Vku1 = 1;} \\
\end{align*}
\]
xm2.Vku1 = (1 - PosOld[j].Pku1)/VelocityOld[j].Vku1;
}
if (VelocityOld[j].Vku1 < 0)
{
    xm2.Vku1 = 1;
}// END of Ku1<=1

// (Kh+Kh1<=1) && (Kr+Kr1<=1) && (Ku+Ku1<=1)
la1 = VelocityOld[j].Vkh + VelocityOld[j].Vkh1;
//PosOld(i,1)+Max_a*V(i,1) + PosOld(i,4)+Max_a*V(i,4) <=1
lb1 = 1-PosOld[j].Pkh-PosOld[j].Pkh1;
la2 = VelocityOld[j].Vkr+VelocityOld[j].Vkr1;
lb2 = 1-PosOld[j].Pkr-PosOld[j].Pkr1;
la3 = VelocityOld[j].Vku+VelocityOld[j].Vku1;
lb3 = 1-PosOld[j].Pku-PosOld[j].Pku1;

if (la1 == 0)//Kh+Kh1 <=1; la1*x1<=lb1
{
    x1 = 1;
}
if (la1 > 0)
{
    if ((lb1/la1) > 1)
    {
        x1 = 1;
    }
    else
    {
        x1 = lb1/la1;
    }
}
if (la1 < 0)
{
    x1 = 1;
}// END Kh+Kh1 <=1; la1*x1<=lb1

if (Max_a != 0) // Kr+Kr1 <=1; la2*x2<=lb2
{
    if (la2 == 0)
    {
        x2 = 1;
    }
    if (la2 > 0)
    {
        if ((lb2/la2) > 1)
        {
            x2 = 1;
        }
        else
        {
            x2 = lb2/la2;
        }
    }
}
if (la2 < 0)
{
    x2 = 1;
} // END Kr+Kr1 <=1; la2*x2<=lb2

if (Max_a != 0) // Ku+Kul <=1; la3*x3<=lb3
{
    if (la3 == 0)
    {
        x3 = 1;
    }
    if (la3 > 0)
    {
        if ((lb3/la3) > 1)
            x3 = 1;
        else
        {
            x3 = lb3/la3;
        }
    }
    if (la3 < 0)
    {
        x3 = 1;
    } // END of Ku+Kul <=1; la3*x3<=lb3

    // x4 =
    solve('((PosOld(i,1)+x4*V(i,1))^2)*(ih_load^2) +
    ((PosOld(i,2)+x4*V(i,2))^2)*(ir_load^2)+((PosOld(i,3)+x4*V(i,3))^2)*(iu_load^2)=(irms^2)');

    // x5 =
    solve('((PosOld(i,4)+x5*V(i,4))^2)*(ih_load^2) +
    ((PosOld(i,5)+x5*V(i,5))^2)*(ir_load^2)+((PosOld(i,6)+x5*V(i,6))^2)*(iu_load^2)=(irms1^2)');

    // x6 = solve('((1-(PosOld(i,1)+x6*V(i,1))- (PosOld(i,4)+x6*V(i,4)))^2)*(ih_load^2) + ((1-(PosOld(i,2)+x6*V(i,2))- (PosOld(i,5)+x6*V(i,5)))^2)*(ir_load^2)+((1-(PosOld(i,3)+x6*V(i,3))- (PosOld(i,6)+x6*V(i,6)))^2)*(iu_load^2)=(irms2^2)');
    if (Max_a != 0)
    {
        A11 = pow((VelocityOld[j].Vkh*ih_load),2) +
        pow((VelocityOld[j].Vkr*ir_load),2) +
        pow((VelocityOld[j].Vku*iw_load),2);
        B11 =
        2*PosOld[j].Pkh*VelocityOld[j].Vkh*(pow(ih_load,2)) +
        2*PosOld[j].Pkr*VelocityOld[j].Vkr*(pow(ir_load,2)) +
        2*PosOld[j].Pku*VelocityOld[j].Vku*(pow(iw_load,2));
    }
\[ C_{11} = \text{pow}((\text{PosOld}[j].Pkh \cdot \text{ih}_\text{load}), 2) + \text{pow}((\text{PosOld}[j].Pkr \cdot \text{ir}_\text{load}), 2) + \text{pow}((\text{PosOld}[j].Pku \cdot \text{iu}_\text{load}), 2) - \text{pow}(\text{irms} \cdot 0.95, 2); \]

\[ \text{delta}_1 = \text{pow}(B_{11}, 2) - 4 \cdot A_{11} \cdot C_{11}; \]

//DAN DAN DAN

if (A_{11} == 0) // not quadric function
{
    if (B_{11} == 0)
    {
        x_4 = 1;
    }
    if (B_{11} > 0)
    {
        if ((-C_{11}/B_{11}) > 1)
        {
            x_4 = 1;
        }
        else
        {
            x_4 = (-C_{11}/B_{11});
        }
    }
    if (B_{11} < 0)
    {
        x_4 = 1;
    }
} //END if (A_{11} == 0) % Solve Constraints

if (A_{11} > 0) // parabola opens upward
{
    if (\text{delta}_1 < 0)
    {
        \text{Max}_a = 0;
    }
    if (\text{delta}_1 > 0)
    {
        \text{x}_41 = (-B_{11} - \text{sqrt(\text{delta}_1)}) / (2 \cdot A_{11});
        \text{x}_42 = (-B_{11} + \text{sqrt(\text{delta}_1)}) / (2 \cdot A_{11});
        if (\text{x}_41 > 1)
        {
            \text{Max}_a = 0;
        }
        else
        {
            if ((\text{x}_41 <= 1) && (\text{x}_42 >= 1))
            {
                \text{x}_4 = 1;
            }
            else
            {
                if ((\text{x}_42 >= 0) && (\text{x}_42 <= 1))
                {
                    \text{x}_4 = \text{x}_42;
                }
            }  
        }  
    }  
}
```c
else
{
    Max_a = 0; //if (x42 < 0)
}
}

if (delta1 == 0)
{
    x412 = (-B11+sqrt(delta1))/(2*A11);
    if (x412>1)
    {
        Max_a = 0;
    }
    else
    {
        if (x412 < 0)
        {
            Max_a = 0;
        }
        else
        {
            x4 = x412;
        }
    }
}
} //END if (A11>0) % Solve Constraints
} //END if (Max_a ~= 0) % Current Limit PEBB

// PEBB1 current limitation (the second PEBB)

if (Max_a != 0)
{
    A22 = pow((VelocityOld[j].Vkh1*ih_load),2) +
         pow((VelocityOld[j].Vkr1*ir_load),2) +
         pow((VelocityOld[j].Vku1*iu_load),2);
    B22 =
        2*PosOld[j].Pkh1*VelocityOld[j].Vkh1*(pow(ih_load,2)) +
        2*PosOld[j].Pkr1*VelocityOld[j].Vkr1*(pow(ir_load,2)) +
        2*PosOld[j].Pku1*VelocityOld[j].Vku1*(pow(iu_load,2));
    C22 = pow((PosOld[j].Pkh1*ih_load),2) +
         pow((PosOld[j].Pkr1*ir_load),2) +
         pow((PosOld[j].Pku1*iu_load),2) -
         pow((irms1*0.95),2);
    delta2 = pow(B22,2) - 4*A22*C22;

    if (A22==0) //not quadric function
    {
        if (B22 == 0)
        {
            x5 = 1;
        }
    }
```
if (B22 > 0)
{
    if ((-C22/B22) > 1)
    {
        x5 = 1;
    }
    else
    {
        x5 = (-C22/B22);
    }
}
if (B22 < 0)
{
    x5 = 1;
}
} //END if (A22==0) % Solve Constraints

if (A22>0) //parabola opens upward
{
    if (delta2<0)
    {
        Max_a = 0;
    }
    if (delta2 > 0)
    {
        x51=(-B22-sqrt(delta2))/(2*A22);
        x52=(-B22+sqrt(delta2))/(2*A22);
        if (x51>1)
        {
            Max_a = 0;
        }
        else
        {
            if ((x51 <= 1) && (x52 >= 1))
            {
                x5 = 1;
            }
            else
            {
                if ((x52 >= 0) && (x52 <= 1))
                {
                    x5 = x52;
                }
                else
                {
                    Max_a = 0; //if (x52 < 0)
                }
            }
        }
    }
}
if (delta1 == 0)
\[
x_{512} = \frac{(-B_{22} + \sqrt{\delta_{22}})}{2A_{22}};
\]

if \( x_{512} > 1 \)

\[
\text{Max}_a = 0;
\]

else

if \( x_{512} < 0 \)

\[
\text{Max}_a = 0;
\]

else

\[
x_5 = x_{512};
\]
// A33 = pow((VOldi_h*ih_load),2) +
pow((VOldi_r*ir_load),2) + pow((VOldi_u*iu_load),2);
// B33 = 2*PosOld_h*VOldi_h*(pow(ih_load,2)) +
2*PosOld_r*VOldi_r*(pow(ir_load,2)) +
2*PosOld_u*VOldi_u*(pow(iu_load,2));
// C33 = pow((PosOld_h*ih_load),2) +
pow((PosOld_r*ir_load),2) + pow((PosOld_u*iw_load),2) -
(pow((irms2*0.95),2));
// delta3 = pow(B33,2) - 4*A33*C33;

if (A33==0) //not quadric function
{
    if (B33 == 0)
    {
        x6 = 1;
    }
    if (B33 > 0)
    {
        if ((-C33/B33) > 1)
        {
            x6=1;
        }
        else
        {
            x6 = (-C33/B33);
        }
    }
    if (B33 <0)
    {
        x6 = 1;
    }
}
//END if (A33==0) % Solve Constraints

if (A33>0) //parabola opens upward
{
    if (delta3<0)
    {
        Max_a = 0;
    }
    if (delta3 > 0)
    {
        x61=(-B33-sqrt(delta3))/(2*A33);
        x62=(-B33+sqrt(delta3))/(2*A33);
        if (x61>1)
        {
            Max_a = 0;
        }
        else
        {
            if ((x61 <= 1) && (x62 >= 1))
            {
                x6 = 1;
            }
            else
            {
                // Other conditions...
            }
        }
    }
}

if (A33>0)
if (delta3 > 0)
{
    if ((x62 >= 0) && (x62 <= 1))
    {
        x6 = x62;
    }
    else
    {
        Max_a = 0; //if (x62 < 0)
    }
}

} // End of if (delta3 > 0)
if (delta3 == 0)
{
    x612 = (-B33+sqrt(delta3))/(2*A33);

    if (x612 > 1)
    {
        Max_a = 0;
    }
    else
    {
        if (x612 < 0)
        {
            Max_a = 0;
        }
        else
        {
            x6 = x612;
        }
    }
}

} //END if (A33>0) % Solve Constraints
} //END if (Max_a ~= 0) % Current Limit PEBB2

// Decide Max_a, if all the constraints have roots, With Max_a!=0
if (Max_a != 0)
{
    float A_array[18];
    A_array[0] = xm1.Vkh;
    A_array[3] = xm1.Vkh1;
    A_array[8] = xm2.Vku;
    A_array[9] = xm2.Vkh1;
    A_array[10] = xm2.Vkr1;
}
A_array[12] = x1;
A_array[13] = x2;
A_array[14] = x3;
A_array[15] = x4;
A_array[16] = x5;
A_array[17] = x6;

Max_a = A_array[0];
for (j1=1; j1<18; j1++)
{
    if (A_array[j1] < Max_a) {
        Max_a = A_array[j1];
    }
} // END of
Max_a = ((floor(Max_a) * 10))/10;

} //% END if (Max_a != 0) % Get Max_a

if (While2 == 1)
{
    OneTimeMaxA = OneTimeMaxA +1;
}
if (While2 == 2 )
{
    TwoTimeMaxA = TwoTimeMaxA + 1;
}
//DAN DAN
if (While2 == 3)
{
    ThreeTimeMaxA = ThreeTimeMaxA + 1;
}
if (While2 == 4)
{
    FourTimeMaxA = FourTimeMaxA + 1;
}
if (While2 == 5)
{
    FiveTimeMaxA = FiveTimeMaxA + 1;
}
if (While2 == 6)
{
    SixTimeMaxA = SixTimeMaxA + 1;
}
if (While2 == 7)
{
    SevenTimeMaxA = SevenTimeMaxA + 1;
}
if (While2 == 8)
{
    EightTimeMaxA = EightTimeMaxA + 1;
}
if (While2 == 9)
{  NineTimeMaxA = NineTimeMaxA + 1; }
if (While2 == 10)
{
    TenTimeMaxA = TenTimeMaxA + 1;
}
Velocity[j].Vkh = Max_a * VelocityOld[j].Vkh;
Velocity[j].Vkr = Max_a * VelocityOld[j].Vkr;
Velocity[j].Vku = Max_a * VelocityOld[j].Vku;
Velocity[j].Vkh1 = Max_a * VelocityOld[j].Vkh1;
Velocity[j].Vkr1 = Max_a * VelocityOld[j].Vkr1;
Velocity[j].Vku1 = Max_a * VelocityOld[j].Vku1;
Pos[j].Pkh = PosOld[j].Pkh + Velocity[j].Vkh;
Pos[j].Pku = PosOld[j].Pku + Velocity[j].Vku;
Pos[j].Pkh1 = PosOld[j].Pkh1 +
Velocity[j].Vkh1;
Pos[j].Pkr1 = PosOld[j].Pkr1 +
Velocity[j].Vkr1;
Pos[j].Pku1 = PosOld[j].Pku1 +
Velocity[j].Vku1;

// Check Constraints
Cons11 = (Pos[j].Pkh >=0);
Cons22 = (Pos[j].Pkr >=0);
Cons33 = (Pos[j].Pku >=0);
Cons44 = (Pos[j].Pkh1 >=0);
Cons55 = (Pos[j].Pkr1 >=0);
Cons66 = (Pos[j].Pku1 >=0);
Cons111 = (Pos[j].Pkh <=1.0);
Cons222 = (Pos[j].Pkr <=1.0);
Cons333 = (Pos[j].Pku <=1.0);
Cons444 = (Pos[j].Pkh1 <=1.0);
Cons555 = (Pos[j].Pkr1 <=1.0);
Cons666 = (Pos[j].Pku1 <=1.0);
Cons11Value = Pos[j].Pkh + Pos[j].Pkh1;
Cons11 = (Cons11Value <= 1.0);
Cons22Value = Pos[j].Pkr + Pos[j].Pkr1;
Cons22 = (Cons22Value <= 1.0);
Cons33Value = Pos[j].Pku + Pos[j].Pku1;
Cons33 = (Cons33Value <= 1.0);
Cons44Value =
(pow(((Pos[j].Pkh)*ih_load),2)+pow(((Pos[j].Pkr)*ir_load),2)+pow(((Pos[j].Pku)*iu_load),2));
Cons44 = (Cons44Value <= (pow((irms*0.95),2)));
Cons55Value =
(pow(((Pos[j].Pkh1)*ih_load),2)+pow(((Pos[j].Pkr1)*ir_load),2)+pow(((Pos[j].Pku1)*iu_load),2));
Cons55 = (Cons55Value <=
(pow((irms1*0.95),2)));
Cons66Value = (pow(((1-Pos[j].Pkh-
Pos[j].Pkh1)*ih_load),2)+pow(((1-Pos[j].Pkr-
Pos[j].Pkr1)*ir_load),2)+pow(((1-Pos[j].Pku-
Pos[j].Pku1)*iu_load),2));
Cons66 = (Cons66Value <=
(pow((irms2*0.95),2)));  
Const = (Cons111 && Cons222 && Cons333 &&
Cons444 && Cons555 && Cons666 && Cons1111 && Cons2222 && Cons3333 &&
Cons4444 && Cons5555 && Cons6666 && Cons11 && Cons22 && Cons33 && Cons44
&& Cons55 && Cons66); //It must satisfy all the constraints
if (While2 >=3)
{
    break;
}
} //End of while(Const==0) Cal. Max_a
} // END for (j=0; j<Pnum; i++) // Update Velocity and
Position
for (j=0; j<Pnum; j++)
{
    if (((unsigned short)(Pos[j].Pkh*Kh_To_Fix)) ==
((unsigned short)(Pos[0].Pkh*Kh_To_Fix)))
    {
        StopCond.Stop[j][0] = 1;
    }
    else
    {
        StopCond.Stop[j][0] = 0;
    }
if (((unsigned short)(Pos[j].Pkr*Kr_To_Fix)) ==
((unsigned short)(Pos[0].Pkr*Kr_To_Fix)))
    {
        StopCond.Stop[j][1] = 1;
    }
    else
    {
        StopCond.Stop[j][1] = 0;
    }
if (((unsigned short)(Pos[j].Pku*Ku_To_Fix)) ==
((unsigned short)(Pos[0].Pku*Ku_To_Fix)))
    {
        StopCond.Stop[j][2] = 1;
    }
    else
    {
        StopCond.Stop[j][2] = 0;
    }
if (((unsigned short)(Pos[j].Pkh1*Kh_To_Fix)) ==
((unsigned short)(Pos[0].Pkh1*Kh_To_Fix)))
    {
        StopCond.Stop[j][3] = 1;
    }
    else
    {
        StopCond.Stop[j][3] = 0;
    }

    if (((unsigned short)(Pos[j].Pkr1*Kr_To_Fix)) ==
((unsigned short)(Pos[0].Pkr1*Kr_To_Fix)))

    if (((unsigned short)(Pos[j].Pku1*Ku_To_Fix)) ==
((unsigned short)(Pos[0].Pku1*Ku_To_Fix)))

    if (((unsigned short)(Pos[j].Pkh1*Kh_To_Fix)) ==
((unsigned short)(Pos[0].Pkh1*Kh_To_Fix)))

    if (((unsigned short)(Pos[j].Pkr1*Kr_To_Fix)) ==
((unsigned short)(Pos[0].Pkr1*Kr_To_Fix)))

    if (((unsigned short)(Pos[j].Pku1*Ku_To_Fix)) ==
((unsigned short)(Pos[0].Pku1*Ku_To_Fix)))
{  
  StopCond.Stop[j][4] = 1;
}
else
{
  StopCond.Stop[j][4] = 0;
}
if (((unsigned short)(Pos[j].Pku*Ku_To_Fix)) ==
((unsigned short)(Pos[0].Pku*Ku_To_Fix)))
{
  StopCond.Stop[j][5] = 1;
}
else
{
  StopCond.Stop[j][5] = 0;
}
}// Calculate StopCond.Stop[i][j]
SumStop = 0;
for (j=0; j<Pnum; j++)
{
  for(j1=0; j1<Dimen; j1++)
  {
    SumStop = SumStop+StopCond.Stop[j][j1];
  }
}
if (SumStop == (Pnum*6))
{
  Cn_Param[0].Kh=(unsigned short)(Pos[0].Pkh*Kh_To_Fix);
  Cn_Param[0].Kr=(unsigned short)(Pos[0].Pkr*Kr_To_Fix);
  Cn_Param[0].Ku=(unsigned short)(Pos[0].Pku*Ku_To_Fix);
  Cn_Param[1].Kh=(unsigned short)(Pos[0].Pkh1*Kh_To_Fix);
  Cn_Param[1].Kr=(unsigned short)(Pos[0].Pkr1*Kr_To_Fix);
  Cn_Param[1].Ku=(unsigned short)(Pos[0].Pku1*Ku_To_Fix);
  Cn_Param[2].Kh=(unsigned short)((1-Pos[0].Pkh-
Pos[0].Pkh1)*Kh_To_Fix);
  Cn_Param[2].Kr=(unsigned short)((1-Pos[0].Pkr-
Pos[0].Pkr1)*Kr_To_Fix);
  Cn_Param[2].Ku=(unsigned short)((1-Pos[0].Pku-
Pos[0].Pku1)*Ku_To_Fix);
  Wk_Mode_PanToStr();// Get PEBB's working mode from
  setting panel
  PSO_Cn_Param_PanToStr();// Get PEBB's control
  parameters from setting panel
  //Cn_Param_PanToStr();
  for {j1=0; j1<3; j1++}
  {
    SPrtc_Pck_Para(j1);//Send PEBB Parameters Down
} // End of for(j=0; j<3; j++)
break;

} // Calculate SumStop
else
{
    Cn_Param[0].Kh=(unsigned short)(Pos[0].Pkh*Kh_To_Fix);
    Cn_Param[0].Kr=(unsigned short)(Pos[0].Pkr*Kr_To_Fix);
    Cn_Param[0].Ku=(unsigned short)(Pos[0].Pku*Ku_To_Fix);
    Cn_Param[1].Kh=(unsigned short)(Pos[0].Pkh1*Kh_To_Fix);
    Cn_Param[1].Kr=(unsigned short)(Pos[0].Pkr1*Kr_To_Fix);
    Cn_Param[1].Ku=(unsigned short)(Pos[0].Pku1*Ku_To_Fix);
    Cn_Param[2].Kh=(unsigned short)((1-Pos[0].Pkh-
Pos[0].Pkh1)*Kh_To_Fix);
    Cn_Param[2].Kr=(unsigned short)((1-Pos[0].Pkr-
Pos[0].Pkr1)*Kr_To_Fix);
    Cn_Param[2].Ku=(unsigned short)((1-Pos[0].Pku-
Pos[0].Pku1)*Ku_To_Fix);
    Wk_Mode_PanToStr();// Get PEBB's working mode from
setting panel
    Pso_Cn_Para_PanToStr(); // Get PEBB's control
parameters from setting panel
    //Cn_Para_PanToStr();
    for (j1=0; j1<3; j1++)
    {
        SPrtc_Pck_Para(j1);//Send PEBB Parameters Down
    }
    Load package to PEBB
} // End of for(j=0; j<3; j++)
    CountPnum = 0;

    //Delay(1); // DAN DAN DELAY
    //SPrtc_Pck_DCControl(2);
    SPrtc_Pck_DCControl(0);
    } //End of if (SumStop == (Pnum*6)) else{ }
} //End of if (iter>IterNum) else{ }

} //End of Case 0x71
break;

case 0x50:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_0[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 0, Cop_GraphNo); // Display appointed waveform
break;

case 0x51:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_1[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 1, Cop_GraphNo); // Display appointed waveform
    break;

case 0x52:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_2[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 2, Cop_GraphNo); // Display appointed waveform
    break;

case 0x53:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_3[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 3, Cop_GraphNo);
    break;

case 0x54:
    for(j=0; j< WV_LENGTH; j++)
    {
        Wave_Form[i].Waveform_4[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 4, Cop_GraphNo);
    break;

case 0x55:
    for(j=0; j< WV_LEN; j++)
    {
        Wave_Form[i].Waveform_5[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 5, Cop_GraphNo);
    break;

case 0x56:
    for(j=0; j< WV_LEN; j++)
    {
        Wave_Form[i].Waveform_6[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 6, Cop_GraphNo);
    break;

case 0x57:
    for(j=0; j< WV_LEN; j++)
    {
        Wave_Form[i].Waveform_7[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_Dis (i, 7, Cop_GraphNo);
    break;

case 0x58:
    for(j=0; j< WV_LEN; j++)
    {
        Wave_Form[i].Waveform_8[j] = (float)IR_Buffer.Bit16[j+5];
    }
WaveFm_D (i, 8, Cop_GraphNo);
break;

case 0x59:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_9[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_D (i, 9, Cop_GraphNo);
    break;

case 0x5A:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_10[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_D (i, 10, Cop_GraphNo);
    break;

case 0x5B:
    for(j=0; j< WV_Length; j++)
    {
        Wave_Form[i].Waveform_11[j] = (float)IR_Buffer.Bit16[j+5];
    }
    WaveFm_D (i, 11, Cop_GraphNo);
    break;

default:
    break;
    } // End of switch (TI)

} // End of this subroutine: ec_Action(int i)