CHAPTER 4

CONCEPT OF LabVIEW PROGRAMMING AND ARTIFICIAL NEURAL NETWORKS

In this chapter, concept of LabVIEW programming software and artificial neural networks with back propagation training algorithm is discussed in detail for the effective implementation of fault identification of multilevel inverters.

4.1 MERITS OF LabVIEW SOFTWARE

Programmers develop software applications every day in order to increase efficiency and productivity in various situations. LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a programming language and powerful tool. LabVIEW is a graphically-based programming language developed by National Instruments. Its graphical nature makes it ideal for test and measurement, automation, instrument control, data acquisition and data analysis applications (Swain et al. 2003; Teng et al. 2000). This results in significant productivity improvements over conventional programming languages.

LabVIEW is a programming environment, in which one can create programs with graphics. Other programming languages such as C++, Pascal, and basic all use text in order to program while LabVIEW uses a graphical language called G. LabVIEW works on PCs running Microsoft Windows, Mac OS, Sun SPARC stations and HP 9000/700 series workstations running HP-UX.
LabVIEW plays major role in automation and control applications. LabVIEW has built in functions that simplify communication over a network (Fang et al. 2011). Communication VIs allow for use of programs in networked applications using TCP/IP which has the following merits over other protocols:

- Connection of computers from multiple platforms (Windows, Mac, Sun, HP).
- Communication with multiple computers simultaneously.
- Great geographical distances can separate networks.

These merits can facilitate a situation where a person at the remote location can have a VI running on a PC, passing results to an analysis VI in another PC, while someone observes the whole process in real time on some other PC.

4.2 FRONT PANEL IN LabVIEW

The three main parts of a LabVIEW VI are front panel, block diagram and the “sub-VI”. Users can input data using the mouse and the keyboard as well as view results such as error warnings on the same screen. A main goal is to directly simulate the front panel of the instrument using knob, buttons, graphs and various other controls. An icon and a connector can represent each individual VI. Any VI that is used within another VI is called a sub-VI. When a VI is used as a sub-VI, the connector is the mechanism used to wire data into it. The icon is used as an object in the block diagram of another VI.

Figure 4.1 illustrates the front panel of a LabVIEW VI. It contains a knob for selecting the number of measurements per average, a control for selecting the measurement type, a digital indicator to display the output value,
and a stop button. An elaborate front panel can be created without much effort to serve as the user interface for an application.

![Figure 4.1 A typical front panel of LabVIEW VI](image)

The front panel is used to display controls and indicators for the user, and the block diagram contains the code for the VI. The icon, which is a visual representation of the VI, has connectors for program inputs and outputs. Programming languages such as C with BASIC use functions and subroutines are used as important programming elements. The front panel of a VI handles the function inputs and outputs, and the code diagram performs the work of the VI. The Multiple VIs can be used to create large scale applications. A VI may be used as the user interface or as a subroutine in an application. User interface elements such as graphs are easily accessed, as drag and drop units in LabVIEW.
4.3 BLOCK DIAGRAM OF LabVIEW

Figure 4.2 depicts the block diagram or source code that accompanies the front panel in Figure 4.1. The outer rectangular structure represents a While loop and the inner one is a case structure. The icon in the center is a VI or subroutine that takes the number of measurements per average as input and returns the frequency value as the output. The orange line or wire represents the data being passed from the control into the VI. The selection for the measurement type is connected or wired to the case statement to determine which case is executed. When the stop button is pressed, the While loop stops execution. This example demonstrates the graphical nature of LabVIEW and gives us the first look at the front panel, block diagram and icon that make up a virtual instrument.

![Block diagram of the front panel](Image)

LabVIEW is not an interpreted language; it is compiled behind the scenes by LabVIEW’s execution engine. Similar to Java, the VIs are compiled into an executable code that LabVIEW’s execution engine processes during runtime. Every time a change is made to a VI, LabVIEW constructs a wire table for the VI. This wire table identifies elements in the
block diagram that have inputs needed for that element to run. Elements can
be primitive operators such as addition or more complex such as a subVI. If
LabVIEW successfully constructs all the wire tables, we are presented a solid
arrow indicating that the VIs can be executed. If the wire table cannot be
created, then a broken arrow is presented for the VIs with a problem and also
for each VI loaded in memory that requires that VI for execution. All that we
need to understand now is that the main execution subsystem compiles
diagrams, while we write them. This allows programmers to write code and
test it without needing to wait for a compiling process and programmers do
not need to worry about execution speed because the language is not
interpreted.

The wire diagrams that are constructed do not define an order in
which elements are executed. This is an important concept for advanced
programmers to understand. LabVIEW is a data flow-based language, which
means that elements will be executed in a somewhat arbitrary order.
LabVIEW does not guarantee which order a series of elements is executed, if
they are not dependent on each other. A process called arbitrary interleaving
is used to determine the order elements are executed; we may force an order
of execution by requiring that elements require output from another element
before execution. This is a fairly common practice and most programmers do
not recognize that they are forcing the order of execution. When
programming, it will become obvious that some operations must take place
before others can. It is the programmer’s responsibility to provide a
mechanism to force the order of execution in the code design.

4.4 LabVIEW FILE EXTENSIONS

LabVIEW programs are developed by the .vi extension method. However, multiple VIs can be saved into library format with the .llb extension
method. Libraries are useful for grouping related VIs for file management.
When loading a particular VI that makes calls to other VIs, the system is able
to find them quickly. Using a library has benefits over simply using a
directory to group VIs. It saves disk space by compressing VIs and facilitates
the movement of VIs between directories or computers. When saving single
VIs, we have to remember to add the .vi extension. If we need to create a
library for a VI and its subVIs, we will need to create a source distribution
using the LabVIEW Project. If we want to create a new library starting with
one VI, we can use Save or Save As. Then select New VI Library from the
dialog box. The File Manager can then be used to add or remove VIs from a
library.

4.5 LabVIEW PROJECTS

Among other features in LabVIEW, the important one is project
view. LabVIEW’s new project view provides a convenient interface to access
everything in a LabVIEW project. Historically, locating all the VIs in an
application has required the use of the hierarchy window but that does not
locate some things like LabVIEW libraries and configuration of the
application builder. The project explorer provides a tree-driven list of all of
these. The set of VI sources and libraries are shown in the first major
breakdown: the Source tree. Information related to compilation and
installations of an application are kept in the second branch of the tree.
Applications that use the same operating system as the development platform
will not find the System Definition folder to be of value. If a compile target is
something like a Palm Pilot, then this folder is where definitions specific to a
Palm based target would be configured. The project window is shown in
Figure 4.3. Among other things worth noting on the project explorer window
is the toolbar, which contains buttons to create, save and save all VIs in the
application; compile; the standard cut, copy and paste buttons; buttons to
support compilation of VIs; and buttons to support source code control tools.
In general, most work will be done in the Sources branch which provides a listing of all VIs and variables in the project. The Dependencies section is for VIs, DLLs and project libraries that are called statically by a VI. For beginning users of LabVIEW, there are various sources for assistance to aid in learning the language.

4.6 MENUS AND PALETTES

LabVIEW has two different types of menus that are used during programming. The first set is visible in the window of the front panel and diagram. They are visible along the menu bar when the application is active. These are typical pull-down menus similar to other applications. The second set of menus is called pop-up menus (also referred to as popping up). Pop-up menus are made to appear by right clicking and holding down. The pop-up
menu that appears when the cursor is on a blank part of the front panel or block diagram is the controls palette. Similarly, the Functions palette appears on the block diagram. We can select specific objects on the front panel or block diagram and pop up on them. The menus that appear allow us to customize, modify or perform other actions on the object. These menus can vary depending on the object that we pop up on. Figure 4.4 shows the pop menu that appears for a digital indicator.

Figure 4.4 Pop up menu appearing on the front panel

The tools palette is made visible by selecting 'Show Tools Palette' from the Windows pull-down menu from either the front panel or block diagram. The first tool is known as the Operating tool. This is used for editing numbers and text as well as changing values on controls. The arrow represents the Positioning tool for selecting positioning and resizing objects on the front panel or block diagram. Next is the Labeling tool for editing text and creating labels. The Wiring tool is depicted by the spool and is used for
wiring data terminals. The Object Popup tool is located under the arrow. This is exercised for displaying the pop-up menu as an alternative to clicking the right mouse button and next to this is the tool for scrolling through the window. The tool for setting and clearing break points is located under the wiring tool. The probe tool is used with this when debugging applications. Finally, at the bottom is the paint brush for setting colors and the tool for getting color is right above it.

4.7 FRONT PANEL CONTROLS

Numerous front panel controls are available in LabVIEW for developing a variety of applications. The controls palette (shown in Figure 4.5) appears when we make the appropriate selection in the Windows menu.

![Figure 4.5 Controls palette of the LabVIEW software](image-url)
The controls are grouped into categories in a tree. Categories now include things like the modern control palette, the classic control palette and specific use selections such as express VIs and application control references. The subpalettes have a lock in the top left corner to keep the window visible while we are working with the controls. When creating a VI, controls can be simply dragged from the palettes and dropped on the front panel. A terminal, representing the control on the block diagram, then appears for use according to the program. Controls are basically variables that can be manipulated in the code.

Considering the advantages of LabVIEW software such as graphical user interface and remote monitoring concepts which is very useful for development of fault diagnostic and condition monitoring systems for industrial applications, in the present work, fault diagnosis of multilevel inverter is implemented in this software.

4.8 CONCEPT OF ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) are excellent tools for analyzing pollution severity in polymeric insulators that have many variables and complex interactions. Basically neural network consists of three layers, namely input layer, hidden layer and output layer. Here, we use the neural network to classify the faulty switch of multilevel inverter.

The field of neural networks can be thought of as being related to artificial intelligence, machine learning, parallel processing, statistics and other fields. The attraction of neural networks is that they are best suited for solving the problems that are the most difficult to solve by traditional computational methods. Neurobiologists believe that the brain is similar to a massively parallel analog computer, containing about 10^10 simple processors, which each require a few milliseconds to respond to input. With neural network
technology, we can use parallel processing methods to solve some real-world problems where it is very difficult to define a conventional algorithm.

4.8.1 Justification for using Neural Network

The primary goal of the present work is to identify the faulty switch of cascaded H-Bridge multilevel inverter based on the features extracted from the output voltage signal. Therefore it is necessary to choose a suitable method capable of recognizing all the features obtained from the output voltage signal and identify the faulty switch of the multilevel inverter. Amongst the available pattern recognition methods, Artificial Neural Network (ANN) is a well-established and proven tool for addressing such pattern classification tasks. ANN is a parallel and highly adaptive learning system that can learn a task by generalizing from case studies of the tasks. If a problem can be posed as an input-output mapping problem, an ANN can be used as a black box that learns the mapping from input-output examples from known cases of a task.

The significance of the neural network is the ability to learn from its environment and to improve the performance through further learning. In recent times, artificial neural networks are being applied to an increasing number of real world problems of various degrees of complexity. They are good pattern recognition engines and robust classifiers, with the ability for generalization and decision-making about imprecise and incomplete input data. They offer ideal solutions to a variety of classification problems such as speech, character and signal recognition, as well as functional prediction and system modeling, where the physical processes are not understood or are highly complex. The advantage of artificial neural networks lies in their resilience against distortions in the input data and their capability for learning from examples.
4.8.2 Model of a Neuron

Neural network consists of what are called neurons, information processing units, which process information dynamically in response to external inputs. The model of a neuron is shown in Figure 4.6. These neurons have a set of synapses or connecting links that are characterized by weights, an adder for summing up the input signals and an activation function for linking the amplitude of the output of a neuron. The input and output of the neural network usually are connected through “hidden layers”.

Figure 4.6 Model of a Neuron

The information about the relationship between the input and output is stored in the connecting weights during the training process. Before training, a suitable neural network topology has to be selected. The choice of activation function can change the behaviour of the neural network considerably. There are three basic types of activation functions, namely threshold functions, piece-wise linear function and the sigmoid function. In the present work, sigmoid function is used, because it assumes a continuous range of values from 0 to 1 and it is differentiable at each point, which is an essential requirement for using back propagation training algorithm.
4.8.3 Multilayer Feed-Forward Neural Network

Among the various ANN architectures available in the literature, the multilayer Feed Forward network (as shown in Figure 4.7) with back propagation learning algorithm has been used for the present study because of its simple approach and good generalization capability.

![Schematic of Artificial Neural Network](image)

**Figure 4.7 Schematic of Artificial Neural Network.**

Multilayer feed forward neural network consists of an input layer, at least one hidden layer and an output layer. In this research work, Back propagation algorithm is used for training the multilayer neural network. Each layer is entirely related to next one. The number of neurons in the input and output layer is equal to the number of inputs desired and number of classification patterns required respectively. The training phase consists of the iterative presentation of the inputs with the estimated output, the adjustment of the weights and biases depend on the resultant error.

Stimulation is applied to the inputs of the first layer and signals propagate through the middle (hidden) layer(s) to the output layer. Each link between neurons has a unique weighting value.

Inputs from one or more previous neurons are individually weighted, then summed as shown in Figure 4.8. The result is non-linearly...
scaled between 0 and +1, the output value is passed on to the neurons in the next layer. Since the real uniqueness or 'intelligence' of the network exists in the values of the weights between neurons, we need a method of adjusting the weights to solve a particular problem. For this type of network, the most common learning algorithm is called Back Propagation (BP).

![Diagram of a neuron's processing](image)

**Figure 4.8 Processing of output of each neuron from sigmoid function**

### 4.8.4 Back Propagation Training Algorithm

We must provide a learning set that consists of some input examples and the known-correct output for each case. So, we use these input-output examples to show the network what type of behavior is expected and the BP algorithm allows the network to adapt.

The BP learning process works in small iterative steps: One of the example cases is applied to the network and the network produces some output based on the current state of its synaptic weights (initially, the output will be random). This output is compared to the known-good output and a mean-squared error signal is calculated. The error value is then propagated backwards through the network and small changes are made to the weights in each layer. The weight changes are calculated to reduce the error signal for the case in question. The whole process is repeated for each of the example cases then back to the first case again and so on. The cycle is repeated until the overall error value drops below some pre-determined threshold. At this point we say that the network has learned the problem "well enough" - the
network will never exactly learn the ideal function, but rather it will asymptotically approach the ideal function.

![Figure 4.9 Back propagation of error in multilayer feed forward network](image)

**Figure 4.9 Back propagation of error in multilayer feed forward network**

Figure 4.9 shows the back propagation of error in a multilayer feed forward network. In general, BP algorithm could be broken down to four main steps. After choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

- Feed-forward computation
- Back propagation to the output layer
- Back propagation to the hidden layer
- Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small, i.e. the convergence is reached when the error between the measured and the desired output is less than fixed value (convergence criteria). In this network, error is back propagated, weights and
biases are conveniently readjusted with an algorithm used in order to minimize the Mean Square Error (MSE), which is the average of sum of the errors for all set of inputs and corresponding outputs, which is calculated as follows,

\[ MSE = \frac{1}{m} \sum_{k=1}^{m} (S_k - Y_k)^2 \]  

(4.1)

where \( S_k \) and \( Y_k \) are respectively the desired and measured output for the \( k^{th} \) input set and \( m \) is the total number of input sets.

### 4.8.5 Neural Network Parameters

The important factors governing the convergence and the learning time of the neural network are network topology, size, learning rate, number of training sets, convergence criterion and number of iterations. The learning rate is known to damp out oscillations to some extent, during the training phase. Higher values of learning rate may result in fast convergence, but it may result in oscillation. Higher values of training sets and training cycles will increase the training time of the neural network. It is necessary to arrive at an optimal neural network topology for better classification results. Therefore, in order to arrive at an optimal topology, the performance of the neural network for different values of the learning rate, training sets, convergence criterion, iterations and different number of neurons in the hidden layer are to be studied in detail and the performance is to be evaluated.