

ON-LINE INCIPIENT FAULT DETECTION IN SINGLE-PHASE SQUIRREL
CAGE USING ARTIFICIAL INTELLIGENCE

ALVIN BRYAN HEE CHOON LOONG

This thesis is submitted as partial fulfillment of the requirements for the award of the
Bachelor of Electrical Engineering (Hons.) (Power System)

Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

NOVEMBER, 2009

DECLARATION

“All the trademark and copyrights use here in are property of their respective owner. References of information from other sources are quoted accordingly, otherwise the information presented in this report is solely work of the author”.

Signature : _____
Author : ALVIN BRYAN HEE CHOON LOONG
Date : 29 OCTOBER 2009

DEDICATION

Specially dedicated to

My beloved parents, sisters, brother and friends

Thank you for the endless support and encouragement

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr Ahmed N Abdul Alla for his endless support, invaluable guidance and critical comments throughout the project.

Next, my heartiest thanks to my beloved family especially both of my parents Hee Vui Yong and Arlene R Padua who always pray for my success continuously, giving me all the guidance, support and love that I need all the time.

Not forgotten, all of my friends, thank you for giving me moral support and assistance to finish up this project.

Once again, thank you all very much from the bottom of my heart. Without you, I could never have done it successfully.

ABSTRACT

This project creates and develops an artificial neural network that is capable to determine the condition of a motor whether it is in a healthy state or fault state. All of the data used to train the artificial neural network is obtained by using the result from the simulation of MATLAB Simulink model that represent the real motor. The artificial neural network is trained by using radial basis function neural network method. MATLAB is used to construct and develop Graphical User Interface and interface it with the artificial neural network created. By doing so, the user will be able to test the neural network created with ease of using the Graphical User Interface

ABSTRAK

Projek ini menghasilkan rangkaian neural yang mampu mengenalpasti keadaan motor samada motor yang digunakan sihat atau sebaliknya. Semua data-data digunakan untuk mencipta rangkaian neural dikumpul dari simulasi MATLAB Simulink Model yang mewakili nilai sebenar motor. Setelah mencipta rangkaian neural yang dapat mengenalpasti keadaan motor, Perisian MATLAB akan digunakan untuk menghasilkan pengantaramuka pengguna dalam bentuk grafik dan pengguna dapat mencuba rangkaian neural yang dicipta dengan menggunakan pengantaramuka pengguna bagi memudahkan pengguna.

TABLES OF CONTENTS

CHAPTER	TITLE	PAGE
1	INTRODUCTION	
	1.1 General	1
	1.2 Problem Statement	2
	1.3 Objectives	2
	1.4 Project scope	2
	1.5 Overview	3
2	LITERATURE REVIEW	
	2.1 Condition Monitoring	4
	2.2 Induction Motor Fault	4
	2.2.1 Voltage Drop	6
	2.2.2 Stator Winding Fault	6
	2.3 Sensor Signal	7

2.3.1	Vibration	7
2.3.2	Stator Current	7
2.4	Signal Processing Techniques	8
2.4.1	Root Mean Squared (RMS)	8
2.4.2	Frequency Analysis	8
2.5	Artificial Intelligence	9
2.5.1	Radial Basis Function Network	10
2.5.2	Artificial Intelligence Technique	20

3

METHODOLOGY

3.1	Introduction	21
3.1.1	Software	23
3.2	Collecting Data	24
3.2.1	Creating a Simulink Model	24
3.2.2	Simulation of the Single Phase Asynchronous Machine Model	31
3.3	Training Artificial Intelligence (Radial Basis Function Neural Network)	34
3.3.1	Coding for the ANN created	36
3.4	Creating Graphical User Interface	38
3.4.1	GUI Development Environment	38
3.4.2	Starting Guide	39
3.4.3	The Layout Editor	40
3.4.4	Creating the Visual Aspect of the GUI	42
3.4.5	Programming a GUI	43

4	RESULT AND DISCUSSION	
4.1	Graphical User Interface (GUI)	52
4.1.1	Main Menu GUI	52
4.1.2	Help GUI	53
4.1.3	Start GUI	54
4.2	Testing the ANN	58
5	CONCLUSION AND RECOMMENDATIONS	
5.1	Conclusion	62
5.2.	Suggestion for Future Work	63
	REFERENCE	64
	Appendices A-D	66 - 80

LIST OF TABLES

TABLE NO.	TITLE	PAGE
3.1	Comparison of Matlab GUI with others Software	23
3.2	The block set required to build the Simulink model and the description for each block set	26

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
1.1	The Steps of this Project	3
2.1	Types of induction machine faults.	5
2.2	Radial basis function neural network	10
3.1	The Flow Chart of the Three Stages of this Project	22
3.2	Create a new model	25
3.3	Empty Simulink model	25
3.4	Single Phase Asynchronous Machines Simulink Model(Healthy State)	27
3.5	Block Parameters: Single Phase Asynchronous Machine	28
3.6	Single Phase Asynchronous Machines Simulink	

	Model (Interturn Fault State)	29
3.7	Single Phase Asynchronous Machines Simulink Model (Voltage Drop Fault)	30
3.8	Block Parameters for voltage drop fault Simulink Model.	31
3.9	Source Block Parameters: Step which represent the load condition of the machine runs at 25%	32
3.10	Single Phase Asynchronous Machines Simulink Model (after simulation)	33
3.11	The Neural Network	35
3.12	GUIDE Quick Start	40
3.13	Layout Editor	41
3.14	Creating the Visual Aspect of the Main Menu GUI	42
3.15	The Property Inspector showing the properties of the push button.	43
3.16	Creating the Visual Aspect of the Start Menu GUI	44
3.17	Creating the Visual Aspect of the Help GUI	45
3.18	Show functions button which enable user to select components of the GUI	47
3.19	The code block for 'edit_Ia' edit text components	48
3.20	The code blocks for 'Evaluate_pushbutton_Callback' edit text components	49
3.21	The code blocks for 'clear_Callback' push button	50
3.22	The code blocks for 'Back_Callback' push button	51
4.1	Main Menu which consists of three push button	53
4.2	Help Menu which consist of one push button	53
4.3	Start Menu which consist of edit text component	

	for data to be inserted.	54
4.4	The Message box indicate that the input is not a number	55
4.5	The Message box indicate that the input is not valid.	56
4.6	The Message box indicate that the motor is in a healthy condition	57
4.7	The Message box indicate that a voltage drop Fault Detected	57
4.8	The Message box indicate that interturn fault Detected	57
4.9	The result of the testing for machine 1	58
4.10	The result of the testing for machine 2	59
4.11	The result of the testing for machine 3	60

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Main Menu GUI coding	66
B	Help GUI coding	69
C	Start GUI coding	71
D	Close Dialog Box coding	80

CHAPTER 1

INTRODUCTION

1.1 General

This project is basically a method or another alternative to monitor the condition for single phase squirrel cage induction motors.

For this project artificial intelligence will be used to identify the state of the motor, whether it is in a healthy condition or in a fault condition. Artificial intelligence is used because of its abilities to do analysis where formal analysis would be difficult or impossible, such as pattern recognition and nonlinear system identification and control.

The experimental data for motor current, voltage, electromechanical torque of the motor under running condition was obtained from the simulation in MATLAB. From the data obtained, MATLAB and Neural Network tools will be used to identify the condition of the motors or machine.

The neural network that was created based on the data can be used to detect fault in single-phase squirrel-cage induction motors by inserting the motor's parameter to monitoring the machine's condition.

1.2 Problem Statement

Condition monitoring of electric machinery can significantly reduce the cost of maintenance and the risk of unexpected failures by allowing early detection of potentially catastrophic faults.

1.3 Objectives

The main core objective of the project is to improve and create other alternatives for condition monitoring of single phase squirrel cage induction motors by developing a Neural Network that is capable of detecting faults in single phase squirrel-cage induction motors while gaining knowledge on the use of Neural Network in MATLAB.

1.4 Project Scope

In order to achieve this project, there are several scopes had been outlined:

- i. The Neural Network created based on the data obtained, can only be applied to the same motor.
- ii. This project is use to detect faults in single-phase squirrel-cage induction motors only.
- iii. Limited to internal faults only(Voltage drop and interturn fault)

1.5 Overview

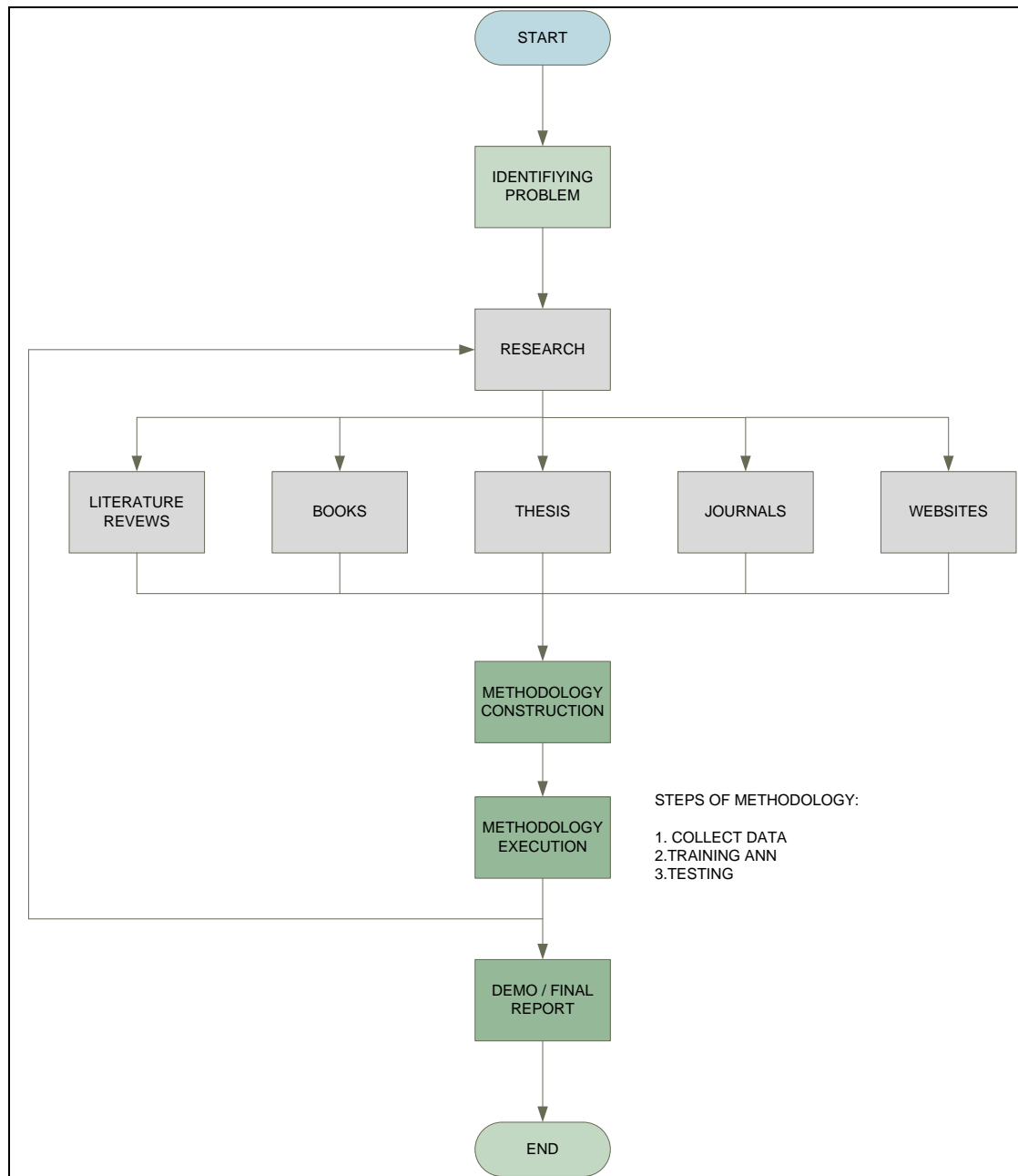


Figure 1.1: The Steps of this Project

CHAPTER 2

LITERATURE REVIEW

2.1 Condition Monitoring

During the past twenty years, there has been a substantial amount of research into the creation of new condition monitoring techniques for electrical machine drives, with new methods being developed and implemented in commercial products for this purpose [15]. On-line condition monitoring involves taking measurements on a machine while it is operating in order to detect faults with the aim of reducing both unexpected failures and maintenance costs. Artificial intelligence is used because of its abilities to do analysis where formal analysis would be difficult or impossible, such as pattern recognition and nonlinear system identification and control. [3]

2.2 Induction Motor Fault

Induction motors play an important role in manufacturing environments, therefore, this type of machine is mainly considered and many diagnostic procedures are proposed both from industry and from academia. [14]

A fault in a component is usually defined as a condition of reduced capability related to specified minimal requirements and is the result of normal

wear, poor specification or design, poor mounting (here also including poor alignment), wrong use, or a combination of these. If a fault is not detected or if it is allowed to develop further it may lead to a failure [13]

The major faults of electrical machines can broadly be classified as the following [14]:

- stator faults resulting in the opening or shorting of one or more of a stator phase winding;
- abnormal connection of the stator windings;
- broken rotor bar or cracked rotor end-rings;
- static and/or dynamic air-gap irregularities;
- bent shaft (akin to dynamic eccentricity) which can result in a rub between the rotor and stator, causing serious damage to stator core and windings;
- shorted rotor field winding;
- bearing and gearbox failures.

Induction machine failure surveys have found the most common failure mechanisms in induction machines[12]. These have been categorized according to the main components of a machine—stator related faults, rotor related faults, bearing related faults and other faults.

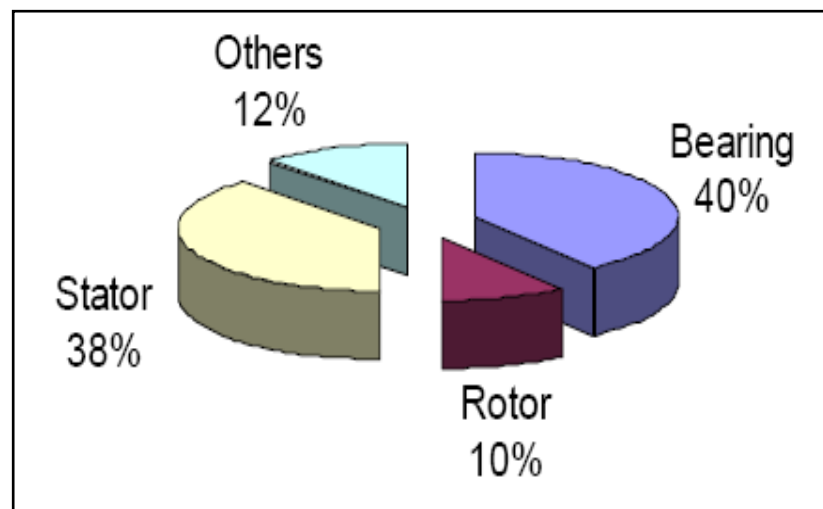


Figure 2.1: Types of induction machine faults.

2.2.1 Voltage Drop

When line voltages applied to a uniphase induction motor are not exactly the same. The effect on the motor can be severe and the motor may overheat to the point of burnout. The voltages should be as closely as can be read on the usually available commercial voltmeter.

2.2.2 Stator Winding Fault

Almost 40% of all reported induction machine failures fall into this category. The stator winding consists of coils of insulated copper wire placed in the stator slots.

Stator winding faults are often caused by insulation failure between two adjacent turns in a coil. This is called a turn-to-turn fault or shorted turn. The resultant induced currents produce extra heating and cause an imbalance in the magnetic field in the machine. If undetected, the local heating will cause further damage to the stator insulation until catastrophic failure occurs. The unbalanced magnetic field can also result in excessive vibration that can cause premature bearing failures [10]

Some of the most frequent causes of stator winding failures are [12]:

- high stator core or winding temperatures,
- slack core lamination, slot wedges, and joints,
- loose bracing for end winding,
- contamination caused by oil, moisture, and dirt,
- short circuits,
- starting stresses,
- electrical discharges,

2.3 Sensor Signal

Different types of sensors can be used to measure signals to detect these faults. Various signal processing techniques can be applied to these sensor signals to extract particular features which are sensitive to the presence of faults.

2.3.1 Vibration

Vibration monitoring is one of the oldest condition monitoring techniques and is widely used to detect mechanical faults such as bearing failures or mechanical imbalance [10]. A piezo-electric transducer providing a voltage signal proportional to acceleration is often used. This acceleration signal can be integrated to give the velocity or position.

2.3.2 Stator Current

The stator current is usually measured using a clip-on Hall-effect current probe. It contains frequency components which can be related to a variety of faults such as mechanical and magnetic asymmetries, broken rotor bars and shorted turns in the stator windings. [10]

2.4 **Signal Processing Techniques**

Signal processing techniques are applied to the measured sensor signals in order to generate features or parameters (e.g. amplitudes of frequency components associated with faults) which are sensitive to the presence or absence of specific faults.[10]

2.4.1 **Root Mean Squared (RMS)**

Calculation of simple statistical parameters such as the overall root mean squared (RMS) value of a signal can give useful information. For instance, the RMS value of the vibration velocity is a convenient measure of the overall vibration severity [4]. In the same way, the RMS value of the stator current provides a rough indication of the motor loading.

2.4.2 **Frequency Analysis**

Frequency analysis using the Fourier transform is the most common signal processing method used for on-line condition monitoring. This is because many mechanical and electrical faults produce signals whose frequencies can be determined from knowledge of motor parameters such as the number of poles. These fault signals appear in a variety of sensor signals including vibration, current and flux [2].

Frequency analysis can thus provide information about a number of faults, though some faults produce similar fault frequencies and so require other

information to differentiate them. It also allows the detection of low-level fault signals in the presence of large “noise” signals at other frequencies.

The use of the frequency analysis of vibration and current signals has been heavily researched to detect bearing, stator, rotor and eccentricity faults. Frequency analysis has also been applied to quantities such as the instantaneous “partial” power and instantaneous torque [2] which can be computed from the measured voltage and current signals.

2.5 Artificial Intelligence

The essence of an expert system is the ability to manage knowledge-based production rules that model the physical system, while it is a main feature of NNs that they are general nonlinear function approximators. This function approximation is achieved by using an appropriate network built up from artificial neurons, which are connected by appropriate weights. However, the exact architecture of a NN is not known in advance; it is usually obtained after a trial-and-error procedure. Fuzzy logic systems are expert, rule-based systems, but they can also be considered to be general nonlinear function approximators. In contrast to NNs, they give a very clear physical description of how the function approximation is performed (since the rules show clearly the function approximation mechanism). On the other hand, fuzzy-NNs are basically NNs with fuzzy features, and it is one main advantage over “pure” NNs that their architecture is well defined [11]. Research trends show that AI techniques will have a greater role in electrical motor diagnostic system with advance practicability, sensitivity, reliability and automation. Diagnostic system based upon fuzzy neural will be very extensively used. Self-repairing electrical drives based upon genetic-algorithm-assisted neural and fuzzy neural systems will also be widely used in the near future [1], [2]. The explored opportunities are to add intelligence to motors, providing a level of communication and diagnostic capability [3], [4].

2.5.1 Radial Basis Function Network

The Basics of Radial Base Function Neural Network and Support vector machine have been demonstrated in this chapter which represents the fundamental tools used in this work. A radial basis function network is an artificial neural network that uses radial basis functions as activation functions. It is a linear combination of radial basis functions. They are used in function approximation, time series prediction, and control. Up to this point we have considered some methods of self-organized learning. Such methods allow a system to organize the data given to it into groups, or clusters, that we call *classes*, from whence comes the term classification. On the other hand, another large category of learning methods is called supervised learning because the systems must be taught, or tutored. The tutoring is done by presenting an input feature vector to the system and then presenting the label, codeword, or output vector that designates the class to which the input belongs. The system then adjusts its parameters according to some algorithm so that it learns the correct output codeword for that input feature vector. After training on examplars (input-output sample pairs) that sufficiently represent a population, the system can then be put online to recognize novel feature vector inputs from the same population.

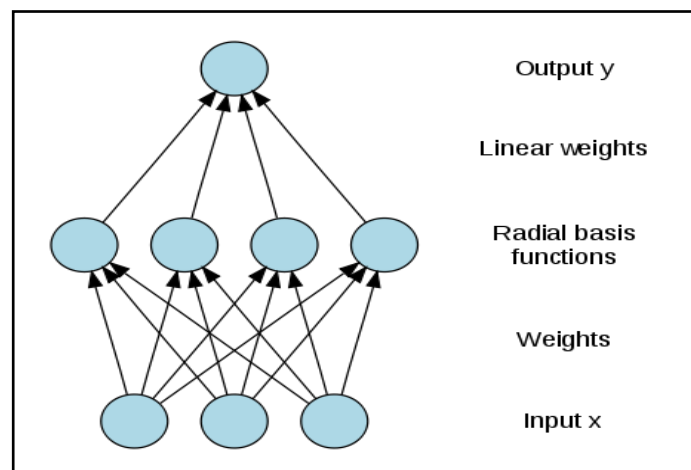


Figure 2.2: Radial Basis Function Neural Network

The diagram in Figure 2.2 is a *radial basis function neural network* (RBFNN) that uses K radial basis functions as the middle layer (columns) of function nodes. The input vector $\mathbf{x} = (x_1, \dots, x_N)$ is a feature vector that is put to the first layer of N nodes, where the n^{th} component of \mathbf{x} goes to the n^{th} input node and then fans out to each to each radial basis function in the middle layer of nodes. Thus each RBF node receives the entire input vector \mathbf{x} .

The network contains 3 layers (array) of nodes from bottom to up. In the usual artificial neural network terminology, the middle layer is called the *hidden layer*. Radial basis functions (RBFs) are shown as Gaussian (bell-shaped) curves inside the nodes in the hidden layer. The input layer of nodes on the bottom is not a layer of *neurodes* (for "neural nodes"), but is only a fanout layer. The hidden and output layers consist of neurodes. Each k^{th} hidden neurode fans out its fuzzy truth value to each of the J neurodes in the *output layer*, so each j^{th} output neurode receives the entire fuzzy truth vector \mathbf{y} .

At the nodes in the output layer we sum up the weighted average of all incoming fuzzy truth values with adjustable weights. More complex schemes have been tried but they yield similar results. The computed output components c_1, \dots, c_J are just real numbers between 0 and 1 if the weights are between 0 and 1 (the fuzzy truth values of $\mathbf{y} = (y_1, \dots, y_K)$ are each between 0 and 1). If we allow any real values as weights, then the outputs can be any positive or negative values, but for the sake of preventing buildup of numerical errors we prefer output values between 0 and 1 in magnitude.

. The output, $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$, of the network is thus

$$\varphi(\mathbf{x}) = \sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

where N is the number of neurons in the hidden layer, \mathbf{c}_i is the center vector for neuron i , and a_i are the weights of the linear output neuron. In the basic form all

inputs are connected to each hidden neuron. The norm is typically taken to be the Euclidean Distance and the basis function is taken to be Gaussian

$$\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2]$$

The Gaussian basis functions are local in the sense that

$$\lim_{\|\mathbf{x}\| \rightarrow \infty} \rho(\|\mathbf{x} - \mathbf{c}_i\|) = 0$$

i.e. changing parameters of one neuron has only a small effect for input values that are far away from the center of that neuron. RBF networks are universal approximators on a compact subset of \mathbb{R}^n . This means that a RBF network with enough hidden neurons can approximate any continuous function with arbitrary precision. The weights a_i , \mathbf{c}_i , and β are determined in a manner that optimizes the fit between φ and the data.

2.5.1.1 Normalized Architecture

In addition to the above *unnormalized* architecture, RBF networks can be *normalized*. In this case the mapping is

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)} = \sum_{i=1}^N a_i u(\|\mathbf{x} - \mathbf{c}_i\|)$$

Where

$$u(\|\mathbf{x} - \mathbf{c}_i\|) \stackrel{\text{def}}{=} \frac{\rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)}$$

is known as a "normalized radial basis function".

2.5.1.2 Theoretical Motivation for Normalization

There is theoretical justification for this architecture in the case of stochastic data flow. Assume a stochastic kernel approximation for the joint probability density

$$P(\mathbf{x} \wedge y) = \frac{1}{N} \sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|) \sigma(|y - e_i|)$$

where the weights \mathbf{c}_i and e_i are exemplars from the data and we require the kernels to be normalized

$$\int \rho(\|\mathbf{x} - \mathbf{c}_i\|) d^n \mathbf{x} = 1$$

And

$$\int \sigma(|y - e_i|) dy = 1$$

The probability densities in the input and output spaces are

$$P(\mathbf{x}) = \int P(\mathbf{x} \wedge y) dy = \frac{1}{N} \sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

and

The expectation of y given an input \mathbf{x} is

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} E(y | \mathbf{x}) = \int y P(y | \mathbf{x}) dy$$

Where

$$P(y | \mathbf{x})$$

is the conditional probability of y given \mathbf{x} . The conditional probability is related to the joint probability through Bayes Theorem.

$$P(y | \mathbf{x}) = \frac{P(\mathbf{x} \wedge y)}{P(\mathbf{x})}$$

which yields

$$\varphi(\mathbf{x}) = \int y \frac{P(\mathbf{x} \wedge y)}{P(\mathbf{x})} dy$$

This becomes

$$\varphi(\mathbf{x}) = \frac{\sum_{i=1}^N e_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)} = \sum_{i=1}^N e_i u(\|\mathbf{x} - \mathbf{c}_i\|)$$

when the integrations are performed.

2.5.1.3 Local Linear Models

It is sometimes convenient to expand the architecture to include local linear models. In that case the architectures become, to first order,

$$\varphi(\mathbf{x}) = \sum_{i=1}^N (a_i + \mathbf{b}_i \cdot (\mathbf{x} - \mathbf{c}_i)) \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

and

$$\varphi(\mathbf{x}) = \sum_{i=1}^N (a_i + \mathbf{b}_i \cdot (\mathbf{x} - \mathbf{c}_i)) u(\|\mathbf{x} - \mathbf{c}_i\|)$$

in the unnormalized and normalized cases, respectively. Here \mathbf{b}_i are weights to be determined. Higher order linear terms are also possible.

This result can be written

$$\varphi(\mathbf{x}) = \sum_{i=1}^{2N} \sum_{j=1}^n e_{ij} v_{ij}(\mathbf{x} - \mathbf{c}_i)$$

where

$$e_{ij} = \begin{cases} a_i, & \text{if } i \in [1, N] \\ b_{ij}, & \text{if } i \in [N + 1, 2N] \end{cases}$$

and

$$v_{ij}(\mathbf{x} - \mathbf{c}_i) \stackrel{\text{def}}{=} \begin{cases} \delta_{ij} \rho(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [1, N] \\ (x_{ij} - c_{ij}) \rho(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [N + 1, 2N] \end{cases}$$

in the unnormalized case and

$$v_{ij}(\mathbf{x} - \mathbf{c}_i) \stackrel{\text{def}}{=} \begin{cases} \delta_{ij} u(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [1, N] \\ (x_{ij} - c_{ij}) u(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [N + 1, 2N] \end{cases}$$

in the normalized case.

Here δ_{ij} is a Kronecker delta function defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

2.5.1.4 Training the Network

In a RBF network there are three types of parameters that need to be chosen to adapt the network for a particular task: the center vectors \mathbf{c}_i , the output weights w_i , and the RBF width parameters β_i . In the sequential training of the weights are updated at each time step as data streams in.

For some tasks it makes sense to define an objective function and select the parameter values that minimize its value. The most common objective function is the least squares function

$$K(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} K_t(\mathbf{w})$$

where

$$K_t(\mathbf{w}) \stackrel{\text{def}}{=} [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})]^2.$$

We have explicitly included the dependence on the weights. Minimization of the least squares objective function by optimal choice of weights optimizes accuracy of fit.

There are occasions in which multiple objectives, such as smoothness as well as accuracy, must be optimized. In that case it is useful to optimize a regularized objective function such as

$$H(\mathbf{w}) \stackrel{\text{def}}{=} K(\mathbf{w}) + \lambda S(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} H_t(\mathbf{w})$$

where

$$S(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} S_t(\mathbf{w})$$

and

$$H_t(\mathbf{w}) \stackrel{\text{def}}{=} K_t(\mathbf{w}) + \lambda S_t(\mathbf{w})$$

where optimization of S maximizes smoothness and λ is known as a regularization parameter.

2.5.1.5 Interpolation

RBF networks can be used to interpolate a function $y : \mathbb{R}^n \rightarrow \mathbb{R}$ when the values of that function are known on finite number of points: $y(\mathbf{x}_i) = b_i, i = 1, \dots, N$. Taking the known points \mathbf{x}_i to be the centers of the radial basis functions and evaluating the values of the basis functions at the same points $g_{ij} = \rho(\|\mathbf{x}_j - \mathbf{x}_i\|)$ the weights can be solved from the equation

$$\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \vdots & & \ddots & \vdots \\ g_{N1} & g_{N2} & \cdots & g_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

It can be shown that the interpolation matrix in the above equation is non-singular, if the points \mathbf{x}_i are distinct, and thus the weights w can be solved by simple linear algebra:

$$\mathbf{w} = \mathbf{G}^{-1}\mathbf{b}$$

2.1.1.6 Function Approximation

If the purpose is not to perform strict interpolation but instead more general function approximation or classification the optimization is somewhat more complex because there is no obvious choice for the centers. The training is typically done in two phases first fixing the width and centers and then the weights. This can be justified by considering the different nature of the non-linear hidden neurons versus the linear output neuron.

2.5.1.7 Training the Basis Function Centres

Basis function centres can be randomly sampled among the input instances or obtained by Orthogonal Least Square Learning Algorithm or found by clustering the samples and choosing the cluster means as the centres.

The RBF widths are usually all fixed to same value which is proportional to the maximum distance between the chosen centres.

2.5.1.8 Pseudo Inverse Solution for the Linear Weights

After the centres c_i have been fixed, the weights that minimize the error at the output are computed with a linear pseudo inverse solution:

$$\mathbf{w} = \mathbf{G}^+ \mathbf{b},$$

where the entries of G are the values of the radial basis functions evaluated at the points x_i : $g_{ji} = \rho(\|x_j - c_i\|)$.

The existence of this linear solution means that unlike Multi-Layer Perceptron (MLP) networks the RBF networks have a unique local minimum (when the centers are fixed).

2.5.1.9 Gradient Descent Training of the Linear Weights

Another possible training algorithm is gradient descent. In gradient descent training, the weights are adjusted at each time step by moving them in a direction opposite from the gradient of the objective function

$$\mathbf{w}(t + 1) = \mathbf{w}(t) - \nu \frac{d}{d\mathbf{w}} H_t(\mathbf{w})$$

where ν is a "learning parameter."

For the case of training the linear weights, a_i , the algorithm becomes

$$a_i(t + 1) = a_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \rho(\|\mathbf{x}(t) - \mathbf{c}_i\|)$$

in the unnormalized case and

$$a_i(t + 1) = a_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] u(\|\mathbf{x}(t) - \mathbf{c}_i\|)$$

in the normalized case.

For local-linear-architectures gradient-descent training is

$$e_{ij}(t + 1) = e_{ij}(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] v_{ij}(\mathbf{x}(t) - \mathbf{c}_i)$$

2.5.1.10 Projection operator training of the linear weights

For the case of training the linear weights, a_i and e_{ij} , the algorithm becomes

$$a_i(t + 1) = a_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{\rho(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

in the unnormalized case and

$$a_i(t + 1) = a_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{u(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N u^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

in the normalized case and

$$e_{ij}(t + 1) = e_{ij}(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{v_{ij}(\mathbf{x}(t) - \mathbf{c}_i)}{\sum_{i=1}^N \sum_{j=1}^n v_{ij}^2(\mathbf{x}(t) - \mathbf{c}_i)}$$

in the local-linear case.

For one basis function, projection operator training reduces to Newton's Method.

2.5.2 Artificial Intelligence Technique

Filippetti et al. (2005) report an induction machine rotor fault diagnosis based on a neural network approach. After the neural network was trained using data achieved through experimental tests on healthy machines and through simulation in case of faulted machines, the diagnostic system was found able to discern between “healthy” and “faulty” machines.

Tallam et al. (2000), present an on-line neural network based diagnostic scheme, for induction machine stator winding turn fault detection. This scheme is claimed to be insensitive to unbalanced supply voltages or asymmetries in the machine and instrumentation. In addition, it is claimed that a turn fault can be detected in the early stage of development.

Huang et al. (2004) propose a scheme to monitor voltage and current space vectors simultaneously in order to monitor the level of air gap eccentricity in an induction motor. For the amplitudes of eccentricity related components that change non-monotonically with the operating conditions, an artificial neural network is used to learn the complicated relationship and estimate corresponding signature amplitudes over a wide range of operating conditions.

Li et al. (2000) use neural networks to perform motor bearing fault diagnosis based on the extracted bearing vibration features. Computer-simulated data were first used to study and design the neural network motor bearing fault diagnosis algorithm. Actual bearing vibration data collected in real-time were then applied to perform initial testing and validation of the approach. The results show that neural networks can be effectively used in the diagnosis of various motor bearing faults through appropriate measurement and interpretation of motor bearing vibration signals

CHAPTER 3

METHODOLOGY

3.1 Introduction

Methodology is several numbers of tasks involved in carry-out the project. Methodology is referred as to job steps or procedure in arranging manner with discipline and smart in making a project successful. This chapter describes the overall methodology in developing On-line incipient fault detection in single-phase squirrel-cage induction motors using AI. This chapter contains three major sections.

The first section is to collect the data by creating and simulating a Simulink model using MATLAB. Three Simulink model will be created where each of the model represent the motor condition. The result of the simulation will then be the data used to create an ANN.

The second sections create and train an ANN. From the data obtained during the first section, a target output will determine the motor condition whether the motor is in a healthy state or fault occurred.

In the third section, the development Graphical User Interface (GUI) is carried out this system. The GUI is developed by using MATLAB for the purpose of evaluating and testing the ANN.

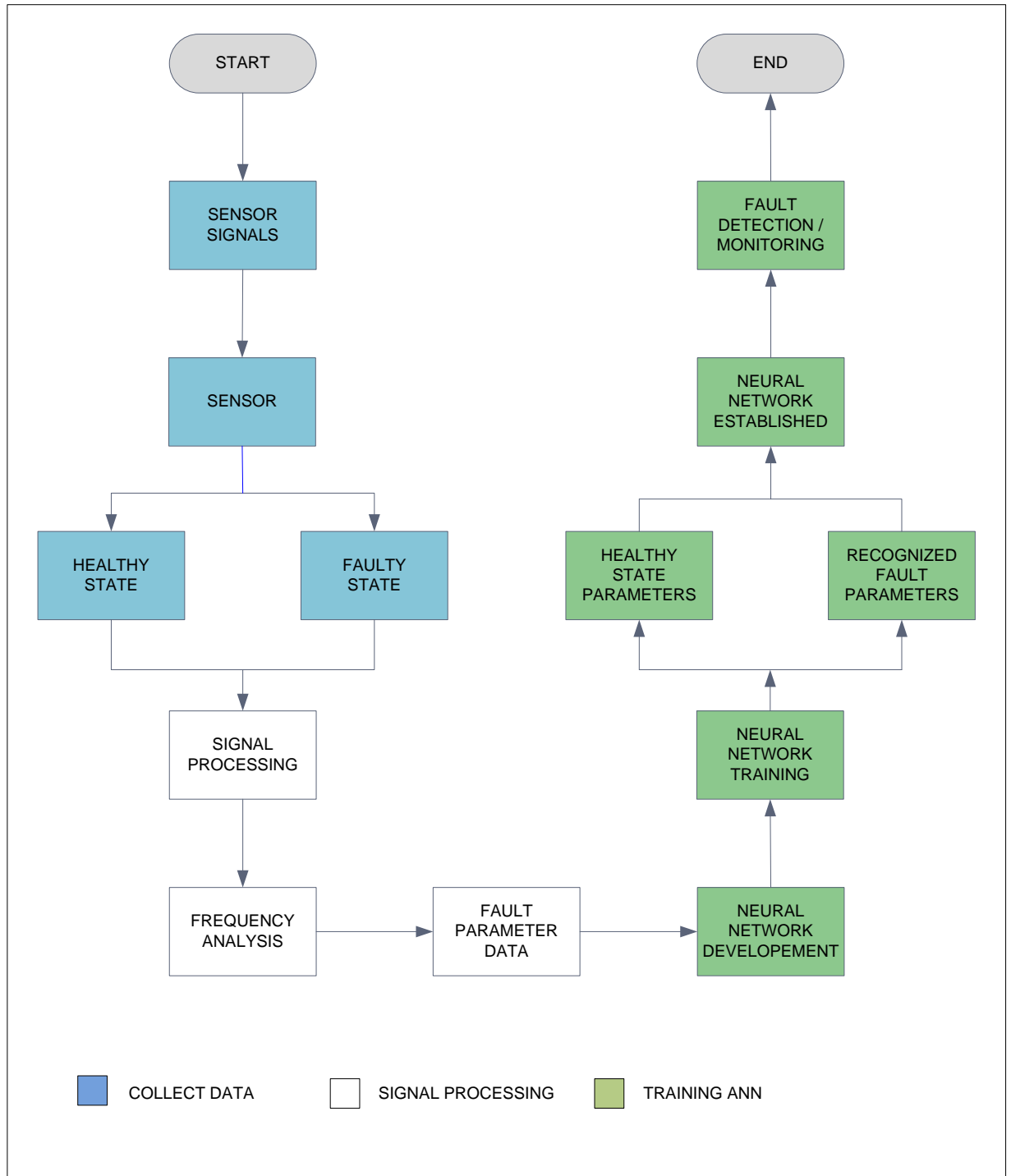


Figure 3.1: The Flow Chart of the Three Stages of this Project

3.1.1 SOFTWARE

For this project a neural network will be created to identify the state of the motor, whether it is in a healthy condition or in a fault condition. Artificial Neural Network is used because of its abilities to do analysis where formal analysis would be difficult or impossible, such as pattern recognition and nonlinear system identification and control.

The MATLAB software is use for this project because it allows one to perform numerical calculations and visualize the result without need for complicated and time consuming programming. This software provides an easy way to go directly from collecting data to deriving informative result. It also accurately solves the problem, to produce graphics easily and create the code efficiently.

A GUI will also be created by using MATLAB. The table shows the different between Matlab GUI and others software that have the same function and the problem with the GUI. The comparison between Matlab GUI and others software like Visual Basic, Labview, and C++ can be the reason why Matlab GUI was selected for this project. In the table also state the problem with the software and can be a guidance to avoid making mistake while working with the software. [5]

Matlab GUI versus others	Problems with GUI
Similar to RAD such as C++ builder and VB	Not as flexible
Can perform most functions as traditional GUI through tricks	Often must use tricks and unfriendly techniques
Can link platform dependent code using MEX programs	MEX code GUI eliminates cross platform operation

Table 3.1: Comparison of Matlab GUI with others software and its problem.

3.2 Collecting Data

For this project, Simulink model will be created and simulated to obtain the data for the three conditions of the motor where the data will be used to create a neural network that is able to determine the state of the motor. Artificial fault will be created by altering the Simulink model's parameters.

3.2.1 Creating a Simulink Model

This procedure explains how to create a simple Simulink model. This model is used to obtain data for three motor conditions. The three motor conditions are motor in healthy state, motor with interturn-fault and motor with voltage drop fault. For each condition, a model will be created and simulated to obtain the data. The Simulink model will be based on a single-phase asynchronous machine in Capacitor-Start mode. The machine is rated 1/4 HP, 110 V, 60 Hz, 1800 rpm and fed by a 110V single phase power supply.

For the basic procedure to create a Simulink model for this project is shown below:

1. Simulink is typed in the MATLAB Command Window. The Simulink Library Browser window is opened as shown in Figure 3.2.
2. From the toolbar, the Create a new model button is clicked.

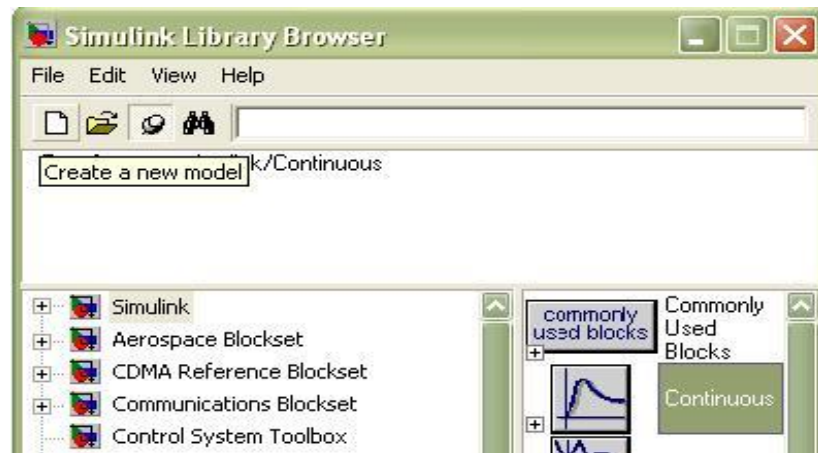


Figure 3.2: Create a new model

3. An empty Simulink window is opened. With the toolbar and status bar disabled, the window looks like following figure 3.3

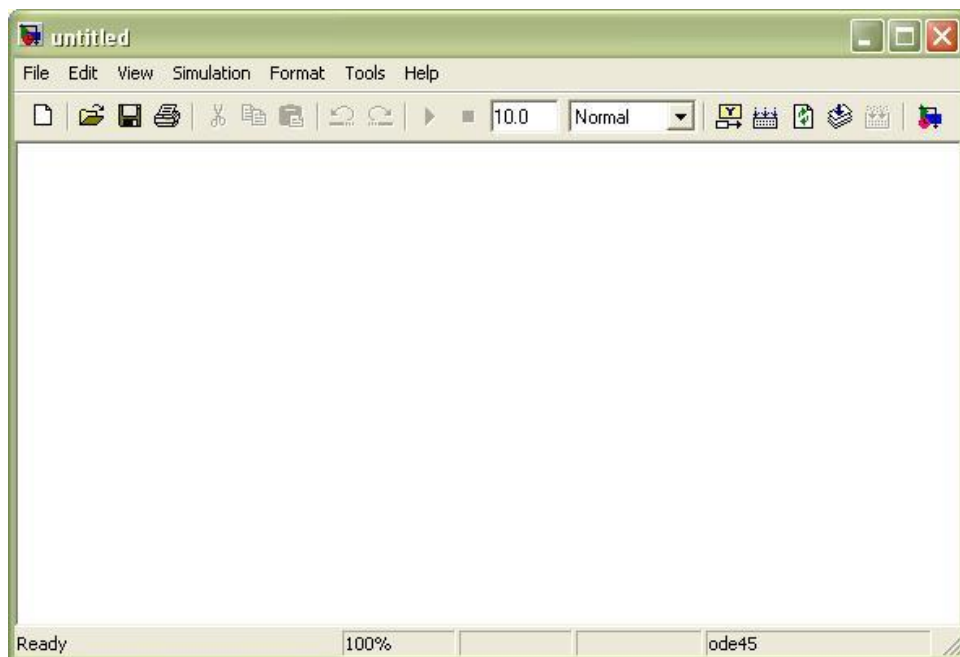


Figure 3.3: Empty Simulink model

4. In the Simulink Library Browser window, select the following:

Block Set	Function
AC Voltage Source	Ideal sinusoidal AC Voltage source
Step	Output a step
Bus Selector	This block accepts a bus as input which can be created from a Bus Creator, Bus Selector or a block that defines its output using a bus object. The left listbox shows the signals in the input bus. Use the Select button to select the output signals. The right listbox shows the selections. Use the Up, Down, or Remove button to reorder the selections. Check 'Output as bus' to output a single bus signal.
Gain	Element-wise gain ($y = K.*u$) or matrix gain ($y = K*u$ or $y = u*K$).
Single Phase Asynchronous Machine	a single phase asynchronous machine (split-phase, capacitor-start, capacitor-start-run, main and auxiliary windings accessible) modeled in the dq stator reference frame. Main and auxiliary windings are in quadrature.
Display	Numeric display of input values.
Voltage Measurement	voltage measurement.

Table 3.2: The block set required to build the Simulink model and the description for each block set.

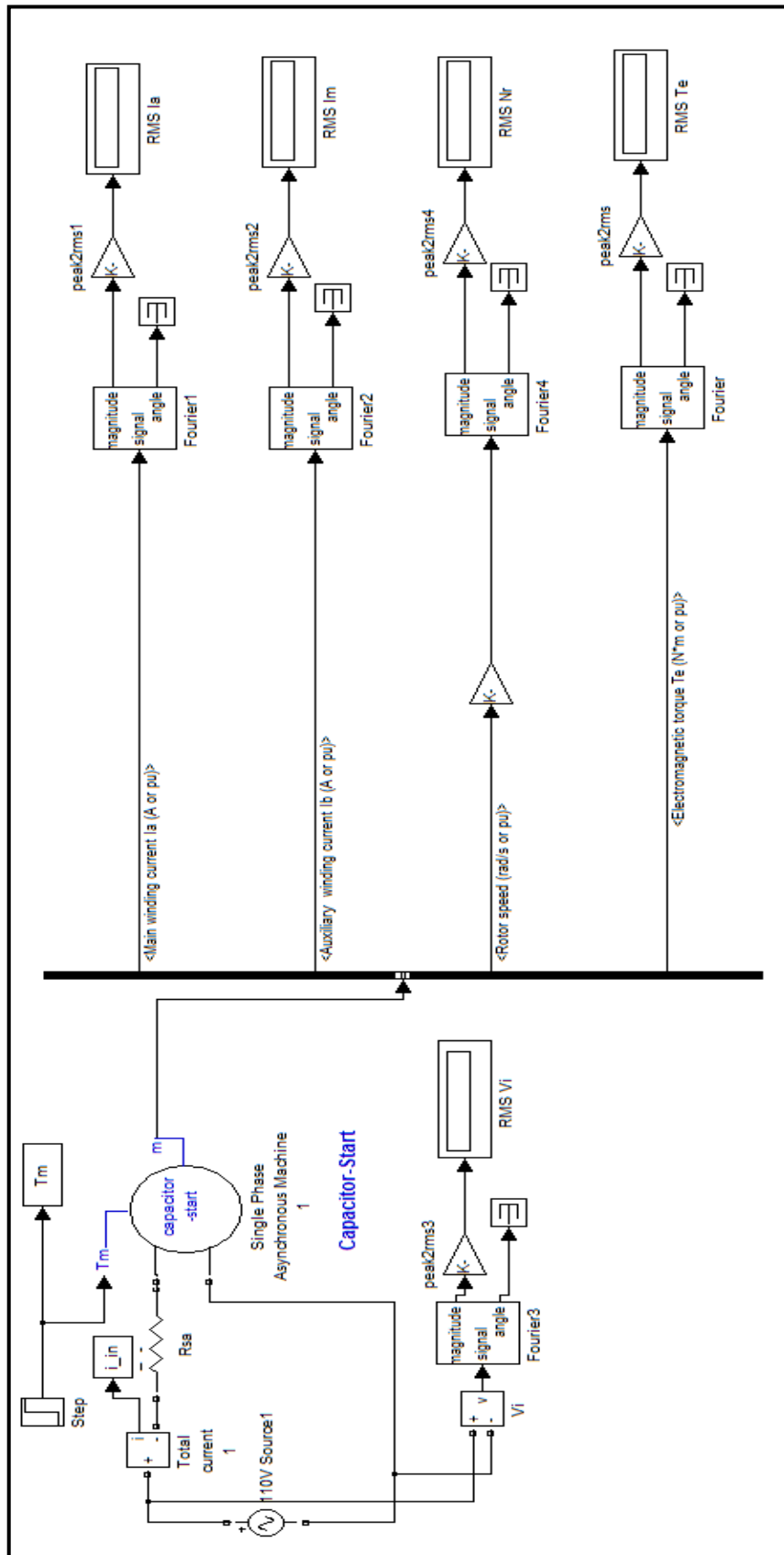


Figure 3.4: Single Phase Asynchronous Machines Simulink Model (Healthy State)

5. After creating the Single Phase Asynchronous Machines Simulink Model (Healthy State), the parameters for each block must be set.

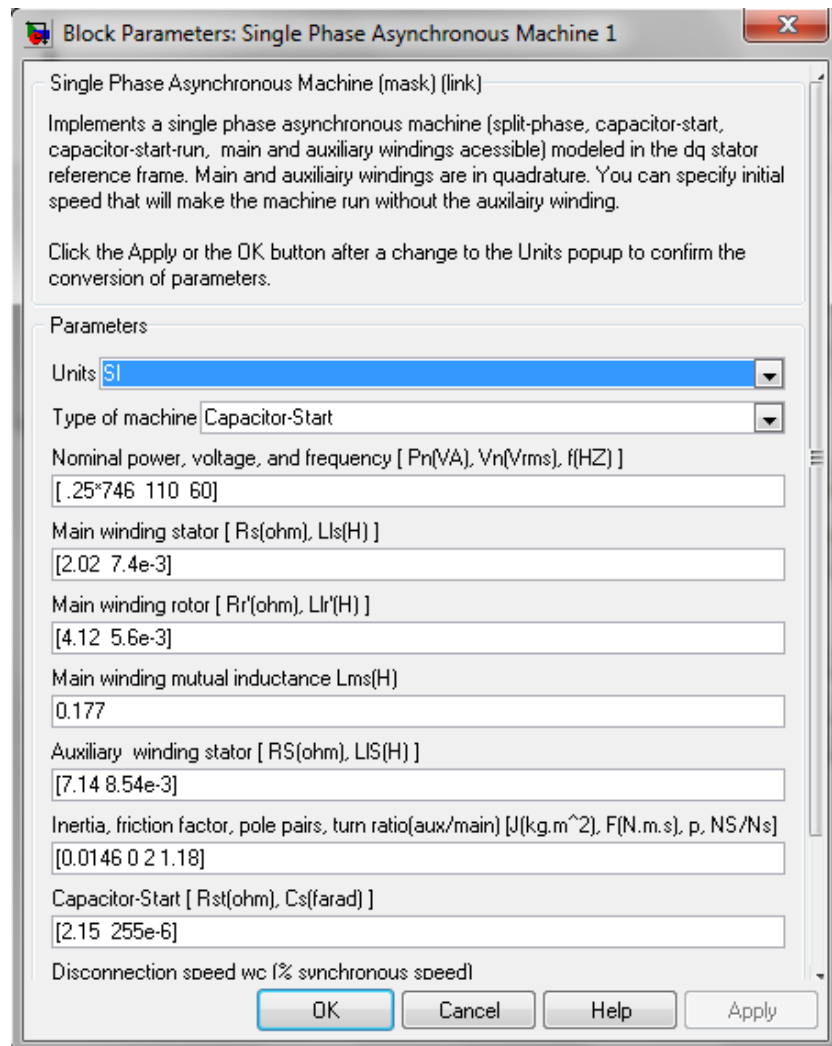


Figure 3.5: Block Parameters: Single Phase Asynchronous Machine

6. For the second motor condition where the motor is in interturn fault, Rsa which represent the stator resistance is not included.

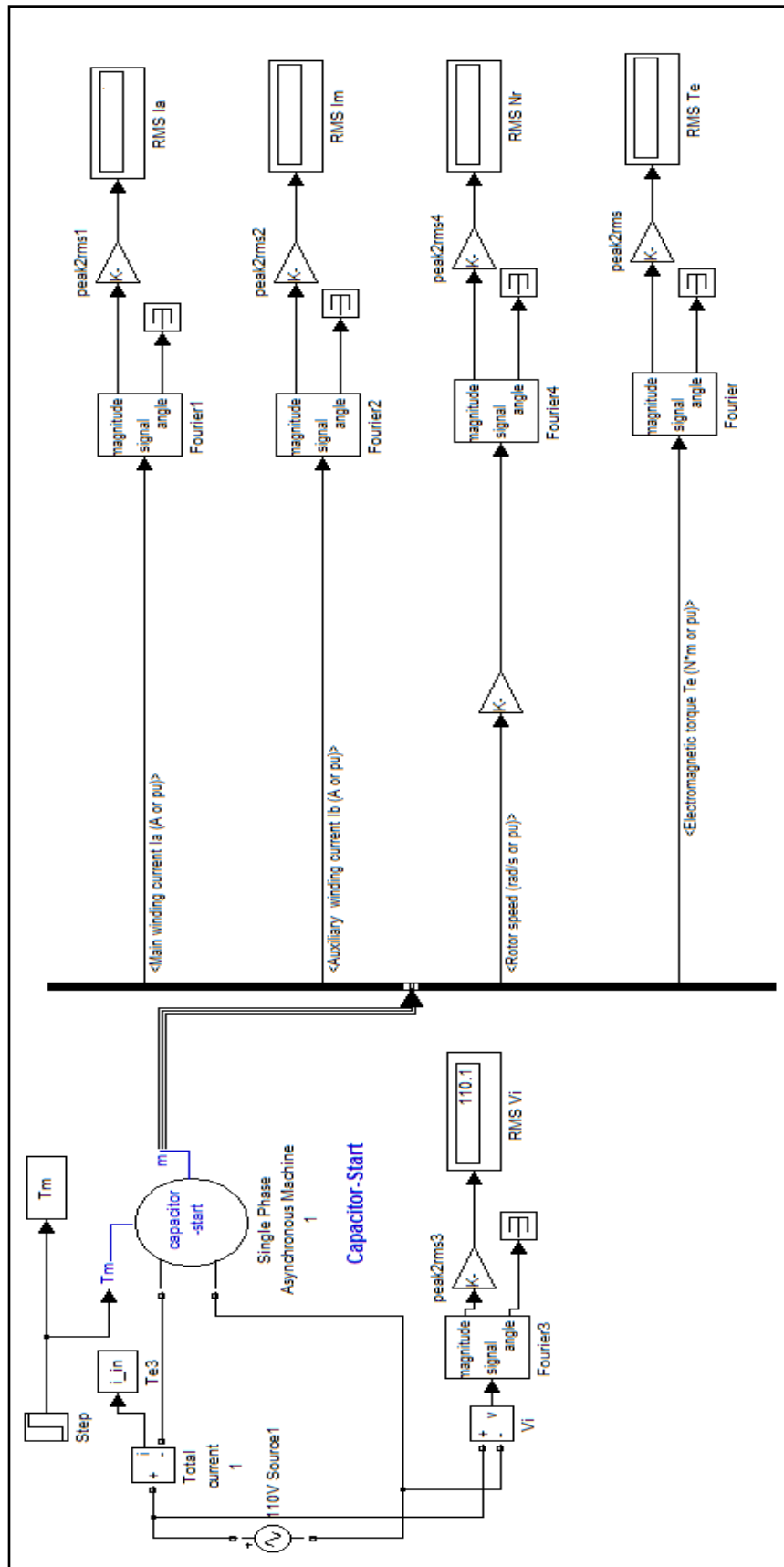


Figure 3.6 Single Phase Asynchronous Machines Simulink Model (Interturn Fault State)

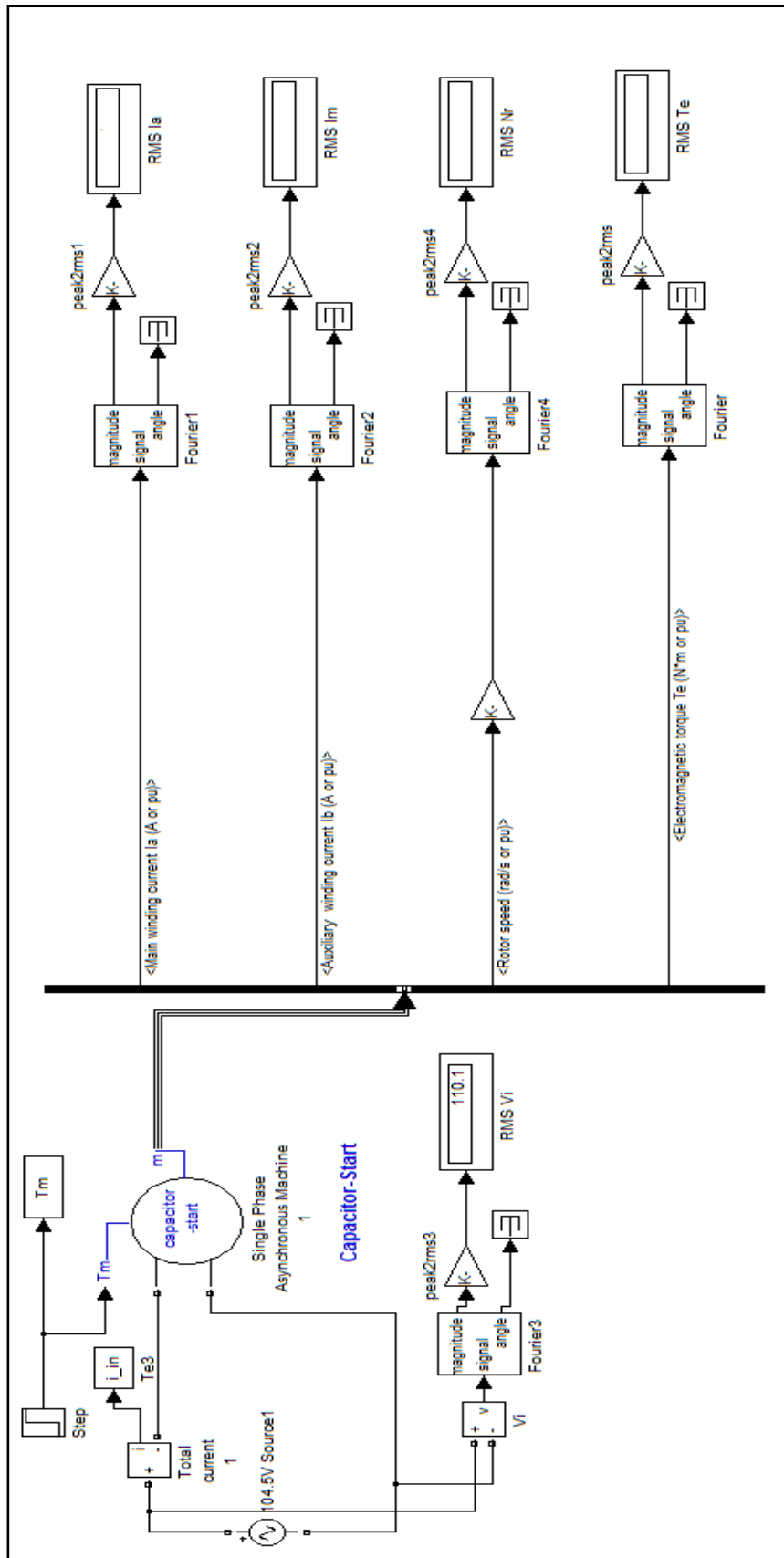


Figure 3.7 :Single Phase Asynchronous Machines Simulink Model (Voltage Drop Fault)

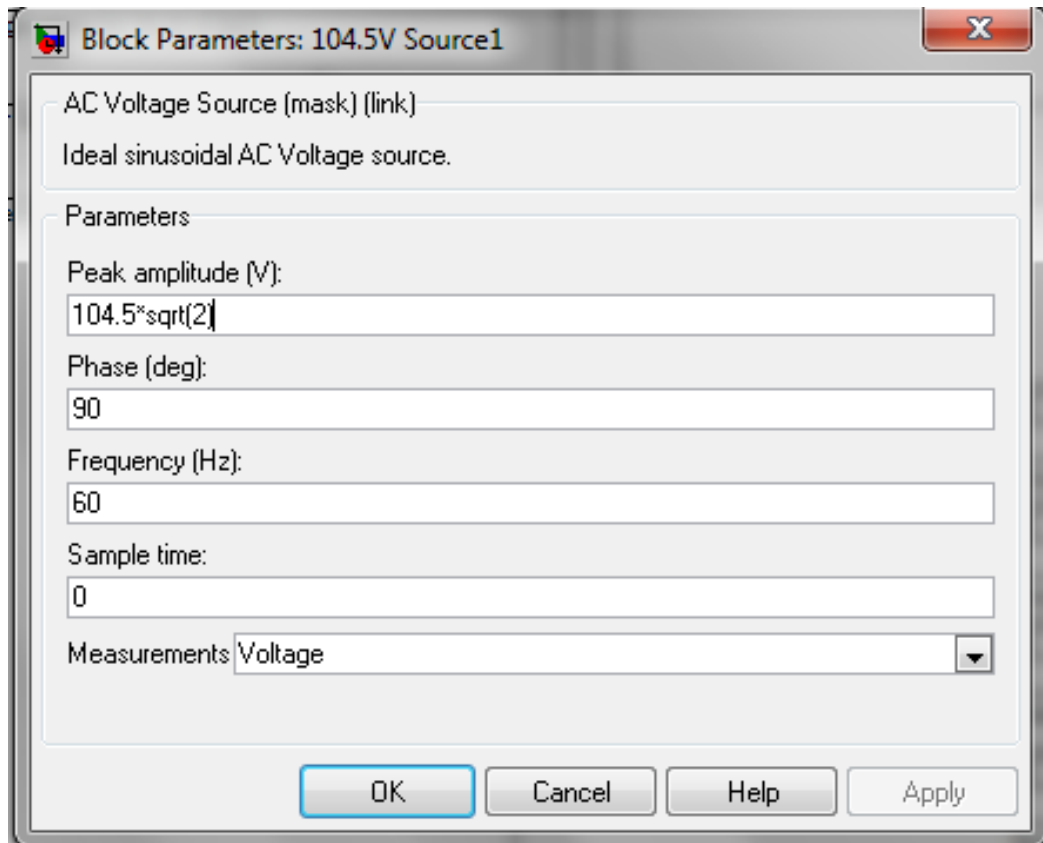


Figure 3.8: Block Parameters for voltage drop fault Simulink Model.

For the voltage drop fault, the voltage source is drop by 5% from 110 V. This fault is artificially created by reducing the voltage supply from 110V to 104.5V.

3.2.2 Simulation of The Single Phase Asynchronous Machine Model

For each motor condition, there are five condition of loads will be used to obtain the simulation result. The five conditions is motor running in 0% (No load), 25% (quarter load), 50 % (half load), 75% (three-quarter load) and 100% (full load).

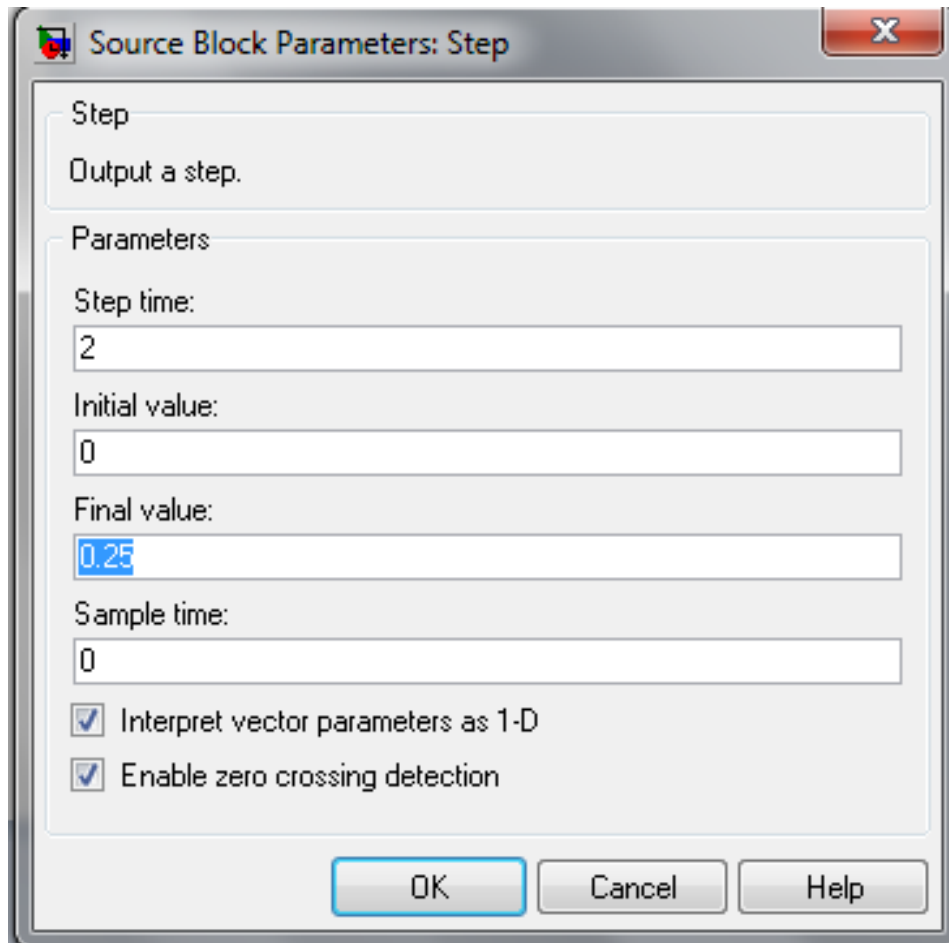


Figure 3.9: Source Block Parameters: Step which represent the load condition of the machine runs at 25% (quarter load)

The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the Step time parameter value, the block's output is the Initial value parameter value. For simulation time greater than or equal to the Step time, the output is the Final value parameter value. To change the load condition, the final value of the step properties is changed. For no load = 0.0, quarter load = 0.25, half load = 0.50, three-quarter load = 0.75 and full load = 1.00.

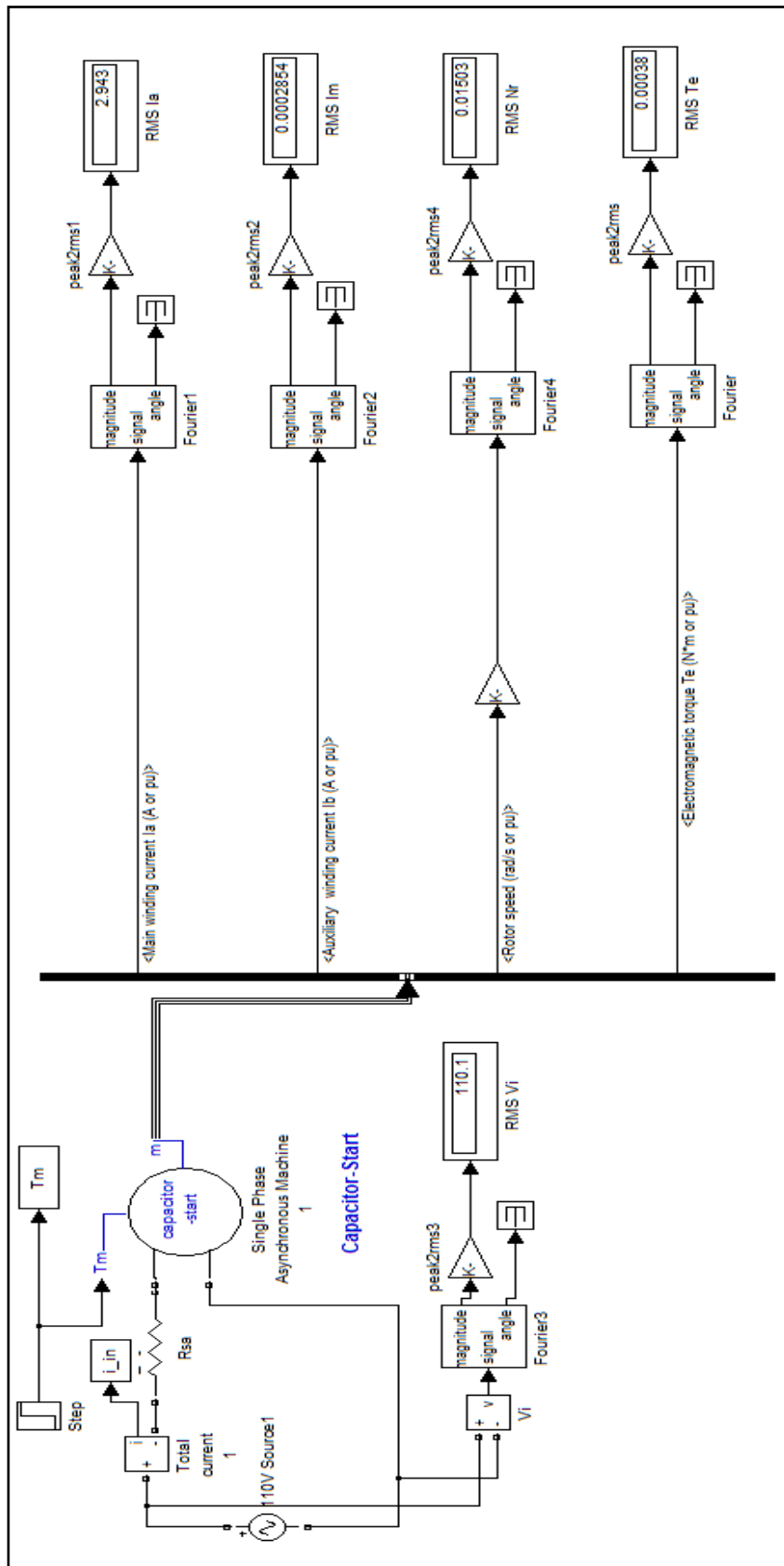


Figure 3.10 : Single Phase Asynchronous Machines Simulink Model (after simulation)

After keying in the load condition, run each machine model will five load conditions to obtain the data. The display will show the RMS value for each parameter. From the data obtained a matrix 6x 15 will be constructed which will be the input for the neural network that will be trained.

	Load	Vi	Ia	Im	Nr	Te
<i>HEALTHY</i>	0.00	107.9	2.965	0.0002305	100.3	0.05529
	0.25	17.26	5.579	$1.184e - 9$	2.335	3.67
	0.50	47.09	4.211	0.0004339	11.64	2.066
	0.75	109.6	3.304	0.0002092	22.12	0.04742
	1.00	109.08	3.555	0.0002095	0.05377	0.05058
<i>VOLTAGE DROP</i>	0.00	104.2	2.789	0.0002106	0.09707	0.03887
	0.25	104.3	2.827	0.0002082	2.422	0.03571
	0.50	49.53	4.232	$3.727e - 5$	37.25	1.898
	0.75	104.1	3.178	0.0002033	2.348	0.04365
	1.00	104.1	3.555	0.0003083	23.31	0.04951
<i>INTERTURN</i>	0.00	10.4	5.186	0.0008248	7.115	3.546
	0.25	24.08	2.644	0	61.94	1.478
	0.50	110.4	3.087	0.0002149	29.15	0.03013
	0.75	109.5	3.264	0.0002153	4.123	0.03934
	1.00	113.3	3.471	0.0002109	69.85	0.09930

3.3 Training Artificial Intelligence (Radial Basis Function Neural Network)

After obtaining the data from the motor, the neural network will be trained to identify the state of the motor, whether it is healthy or in a fault condition.

By using the data, the input and output of the data will be used to train the ANN as shown as the Figure 3.11.

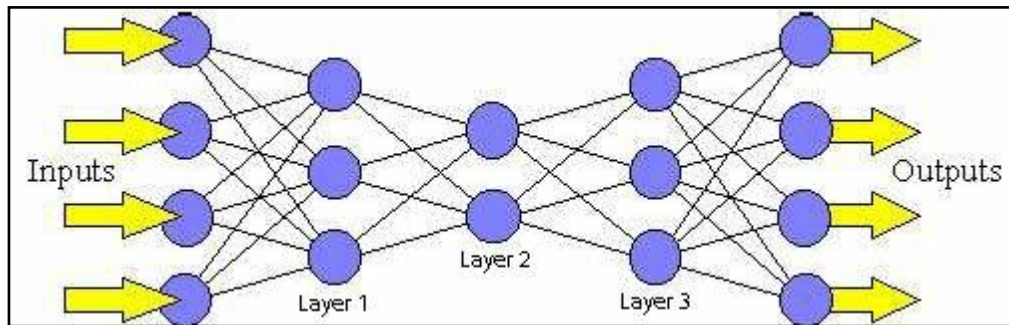


Figure 3.11 : The Neural Network

MOTOR CONDITION	INPUT					OUTPUT	
	Load	V_i	I_a	I_m	N_r	T_e	Target
<i>HEALTHY</i>	0.00	107.9	2.965	0.0002305	100.3	0.05529	0.00
	0.25	17.26	5.579	$1.184e - 9$	2.335	3.67	0.00
	0.50	47.09	4.211	0.0004339	11.64	2.066	0.00
	0.75	109.6	3.304	0.0002092	22.12	0.04742	0.00
	1.00	109.08	3.555	0.0002095	0.05377	0.05058	0.00
<i>VOLTAGE DROP</i>	0.00	104.2	2.789	0.0002106	0.09707	0.03887	0.50
	0.25	104.3	2.827	0.0002082	2.422	0.03571	0.50
	0.50	49.53	4.232	$3.727e - 5$	37.25	1.898	0.50
	0.75	104.1	3.178	0.0002033	2.348	0.04365	0.50
	1.00	104.1	3.555	0.0003083	23.31	0.04951	0.50
<i>INTERTURN</i>	0.00	10.4	5.186	0.0008248	7.115	3.546	1.00
	0.25	24.08	2.644	0	61.94	1.478	1.00
	0.50	110.4	3.087	0.0002149	29.15	0.03013	1.00
	0.75	109.5	3.264	0.0002153	4.123	0.03934	1.00
	1.00	113.3	3.471	0.0002109	69.85	0.09930	1.00

The data obtained from the created Simulink model will be used to train the ANN by using MATLAB. The ANN will train a neural network so that it will get the output (Target). The targets represent the motor condition where 0.00 represent healthy state, 0.5 represent voltage drop fault and 1.0 represent interturn fault.

3.3.1 Coding for the ANN created

Define inputs P from the data collected from the simulation of the Simulink model to train the ANN and associated targets T

```
P = [0.00 107.9 2.965 0.0002305 100.3 0.05529;  
0.25 17.26 5.579 1.184e-009 2.335 3.67;  
0.50 47.09 4.211 0.0004339 11.64 2.066;  
0.75 109.6 3.304 0.0002092 22.12 0.04742;  
1.00 109.08 3.555 0.0002095 0.05377 0.05058;  
0.00 104.2 2.789 0.0002106 0.09707 0.03887;  
0.25 104.3 2.827 0.0002082 2.422 0.03571;  
0.50 49.53 4.232 3.727e-005 37.25 1.898;  
0.75 104.1 3.178 0.0002033 2.348 0.04365;  
1.00 104.1 3.555 0.0003083 23.31 0.04951;  
0.00 10.4 5.186 0.0008248 7.115 3.546;  
0.25 24.08 2.644 0 61.94 1.478;  
0.50 110.4 3.087 0.0002149 29.15 0.03013;  
0.75 109.5 3.264 0.0002153 4.123 0.03934;  
1.00 113.3 3.471 0.0002109 69.85 0.0993];  
T = [0;0;0;0;0;.5;.5;.5;.5;.5;1;1;1;1;1];
```

From the inputs P and the targets T radial basis network is used to determine a function which fits the data points. A radial basis network is a network with layers. A hidden layer of radial basis neurons and an output layer of linear neurons. The weights and biases of each neuron in the hidden layer define the position and width of a radial basis function. Each linear output neuron forms a weighted sum of these radial basis functions. With the correct weight and bias values for each layer, and enough hidden neurons, a radial basis network can fit any function with any desired accuracy.

The function NEWRB quickly creates a radial basis network which approximates the function defined by P and T. In addition to the training set and targets, NEWRB takes two arguments, the sum-squared error goal and the spread constant.

```
%Training  
eg = 0.02; % sum-squared error goal  
sc = 1; % spread constant  
net = newrb(P,T,eg,sc);
```

newrb creates a two-layer network. The first layer has *radbas* neurons, and calculates its weighted inputs with *dist* and its net input with *netprod*. The second layer has *purelin* neurons, and calculates its weighted input with *dotprod* and its net inputs with *netsum*. Both layers have biases.

Initially the *radbas* layer has no neurons. The following steps are repeated until the network's mean squared error falls below goal.

1. The network is simulated.
2. The input vector with the greatest error is found.
3. A *radbas* neuron is added with weights equal to that vector.
4. The *purelin* layer weights are redesigned to minimize error.

The function *newrb* iteratively creates a radial basis network one neuron at a time. Neurons are added to the network until the sum-squared error falls beneath an error goal or a maximum number of neurons has been reached. The call for this function is

$$\text{net} = \text{newrb}(\text{P}, \text{T}, \text{GOAL}, \text{SPREAD})$$

The function *newrb* takes matrices of input and target vectors P and T, and design parameters GOAL and SPREAD, and returns the desired network.

The design method of *newrb* is similar to that of *newrbe*. The difference is that *newrb* creates neurons one at a time. At each iteration the input vector that results in lowering the network error the most is used to create a *radbas* neuron. The error of the new network is checked, and if low enough *newrb* is finished. Otherwise the next neuron is added. This procedure is repeated until the error goal is met or the maximum number of neurons is reached.

3.4 Creating Graphical User Interface

Graphical User Interface (GUI) is the essential part of this project. A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button. MATLAB is used to develop this GUI. All of these tasks are simplified by GUIDE, MATLAB's graphical user interface development environment.

3.4.1 GUI Development Environment

The process of implementing a GUI involves two basic tasks:

- (i) Laying out the GUI components
- (ii) Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the *callbacks* – the functions that execute when users activate components in the GUI.

While it is possible to write an M-file that contains all the commands to lay out a GUI, it is easier to use GUIDE to lay out the components interactively and to

generate two files that save and launch the GUI:

- (i) A FIG-file – contains a complete description of the GUI figure and all of its children (uicontrols and axes), as well as the values of all object properties.
- (ii) An M-file – contains the functions that launch and control the GUI and the callbacks, which are defined as subfunctions. This M-file is referred to as the *application M-file* in this documentation.

3.4.2 Starting Guide

MATLAB GUIs are created using a tool called `guide`, the GUI Development Environment. This tool allows a programmer to layout the GUI, selecting and aligning the GUI components to be placed in it. Once the components are in place, the programmer can edit their properties: name, color, size, font, text to display, and so forth. When `guide` saves the GUI, it creates working program including skeleton functions that the programmer can modify to implement the behavior of the GUI. When `guide` is executed, it creates the Layout Editor, shown in Figure 3.13. The large white area with grid lines is the layout area, where a programmer can layout the GUI. The Layout Editor window has a palette of GUI components along the left side of the layout area. A user can create any number of GUI components by first clicking on the desired component, and then dragging its outline in the layout area. The top of the window has a toolbar with a series of useful tools that allow the user to distribute and align GUI components, modify the properties of GUI components, add menus to GUIs, and so on.

Start `GUIDE` by typing `guide` at the MATLAB command prompt. This displays the `GUIDE` Quick Start dialog, as shown in the following Figure 3.12.

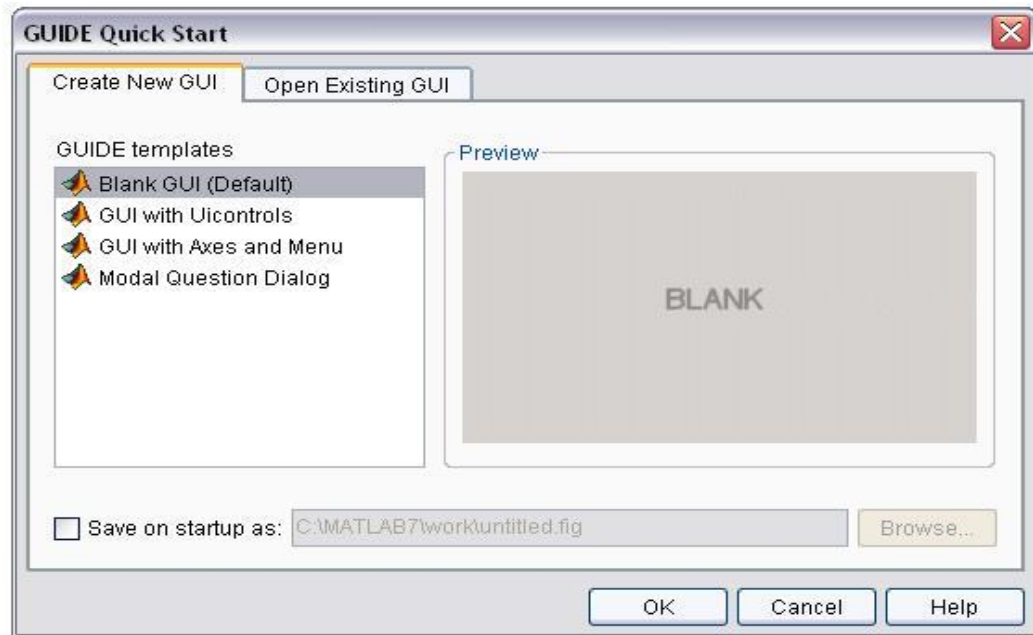


Figure 3.12: GUIDE Quick Start

From the Quick Start dialog, the user can:

- (i) Create a new GUI from one of the GUIDE templates.
- (ii) Open an existing GUI.

3.4.3 The Layout Editor

When the user opened a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. The following Figure 3.24 shows the Layout Editor with a blank GUI template.

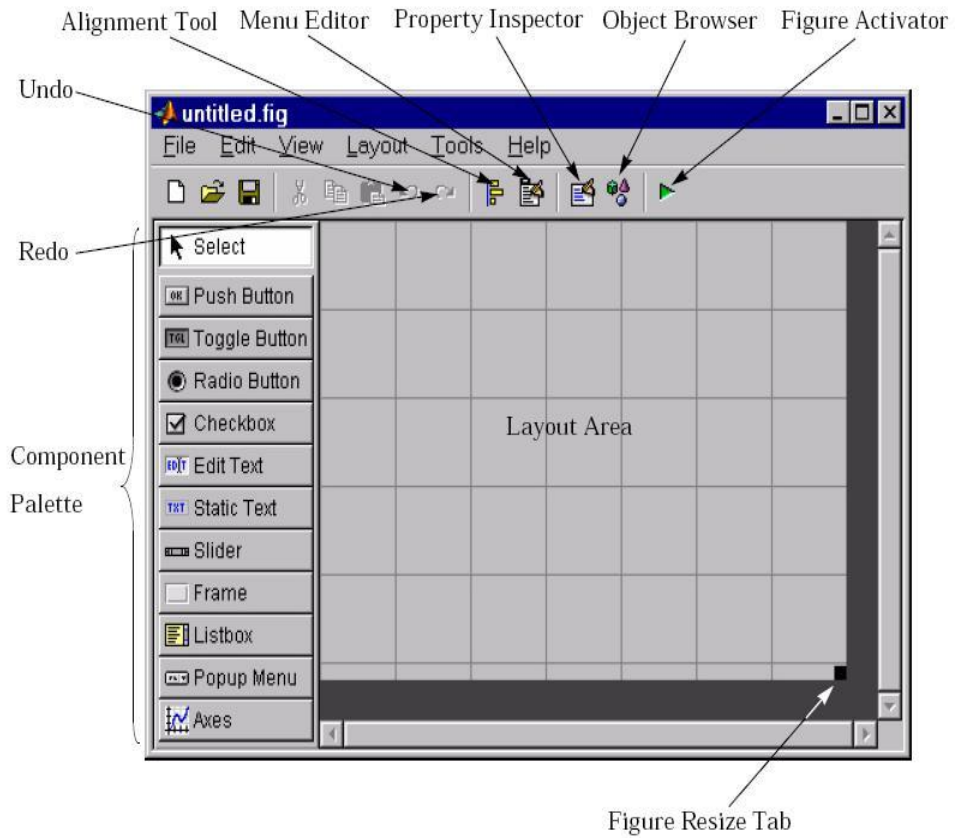


Figure 3.13: Layout Editor

The user can lay out the GUI by dragging components, such as panels, push buttons, pop-up menus, or axes, from the component palette, at the left side of the Layout Editor, into the layout area.

3.4.4 Creating the Visual Aspect of the GUI

3.4.4.1 Main Menu GUI

This GUI determines the next action of the user end. It consist of two static text, three push button and one axes. The objective is to make it easy for first time user and attractive.

MATLAB tool called guide (GUI Development Environment) to layout the Components on a figure. The size of the figure and the alignment and spacing of components on the figure can be adjusted using the tools built into guide.

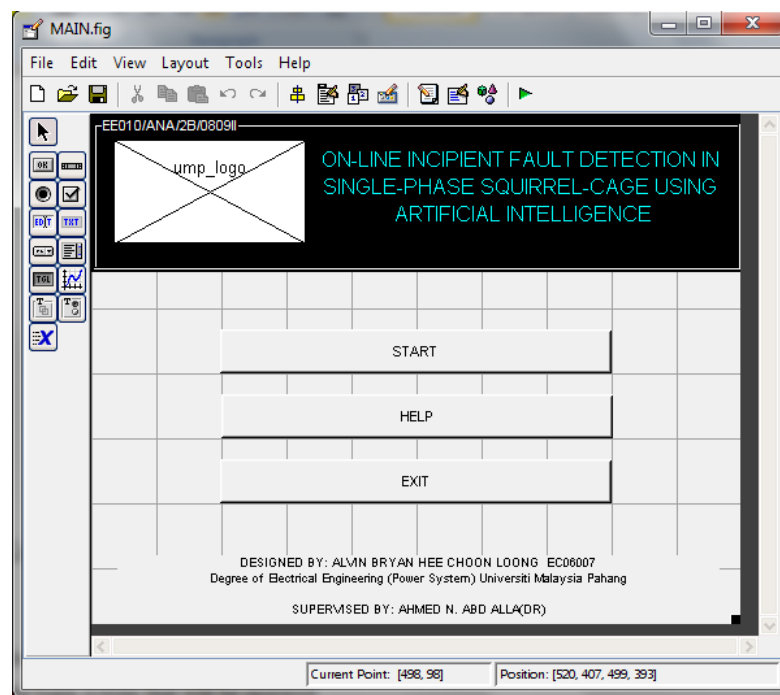


Figure 3.14: Creating the Visual Aspect of the Main Menu GUI

After inserting the components the next step is to edit the properties of these components.

For example start with the Push button. Double click one of the Push button components. The Property Inspector will pop out and allows user to set each component a name (a "tag") and to set the characteristics of each component, such as

its color, the text it displays, and other the properties of a component. Modify the properties of the other component the same way.

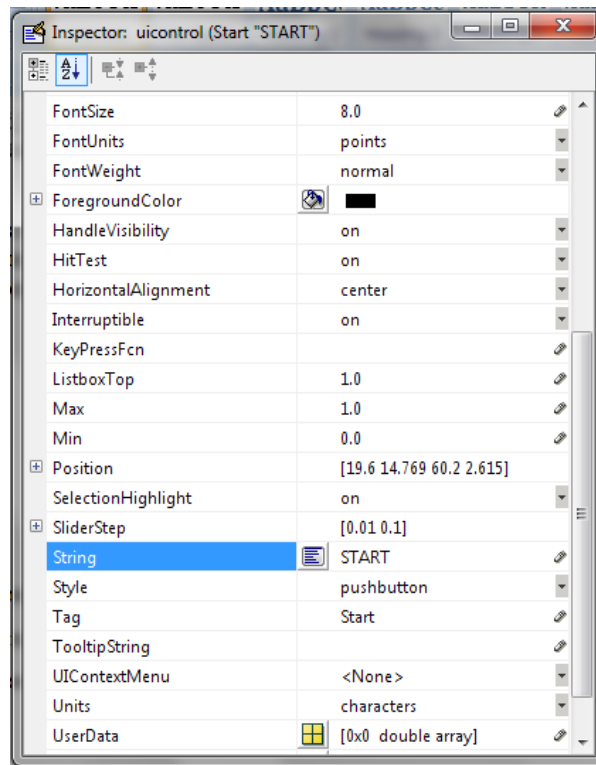


Figure 3.15: The Property Inspector showing the properties of the push button.

In Properties Inspector of a component the two most important properties are the String property, which contains the text to be displayed, and the Tag property, which is the name of the pushbutton. In this case, the String property will be set to 'START', and the Tag property will be set to Start. This name will be needed by the callback function to locate and update the text field.

After setting the Tag property and String property for all of the other components, Save the figure to a file. For this the figure is saved as 'MAIN'. When the figure is saved, two files will be created on disk with the same name but different extents. The MAIN.fig file contains the actual GUI that you have created, and the MAIN.m is an M-file contains the code to load the figure and skeleton call backs for each GUI element.

3.4.4.2 Start Menu GUI

This GUI is the most important part of this project. In this GUI the parameters for the machine that will be evaluated will be inserted into the edit text components. From the data that was inserted in the in the GUI, it will connect with the ANN that was trained previously and work as the inputs of the ANN and the motor condition can be evaluated from the ANN outputs that will be displayed in the GUI.

This GUI consists of six edit text component, eight static text and three push button.

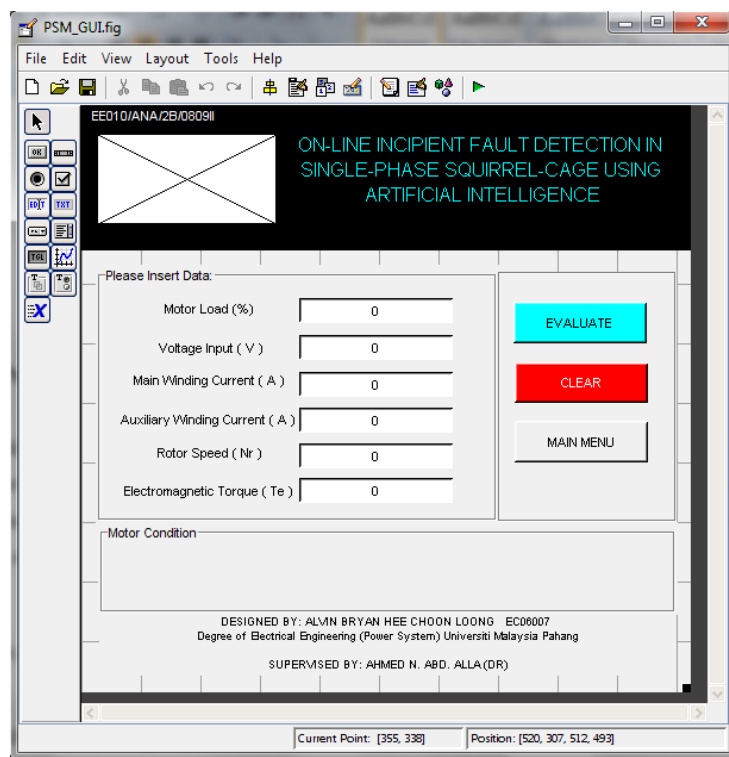


Figure 3.16: Creating the Visual Aspect of the Start Menu GUI

After setting the Tag property and String property for all of the other components, Save the figure to a file. For this the figure is saved as 'PSM_GUI'. When the figure is saved, two files will be created on disk with the same name but different extents. The PSM_GUI.fig file contains the actual GUI that you have

created, and the PSM_GUI.m is an M-file contains the code to load the figure and skeleton call backs for each GUI element.

3.4.4.3 Help Menu GUI

This GUI will give step by step instruction to first time user in order to make this program a user friendly program.

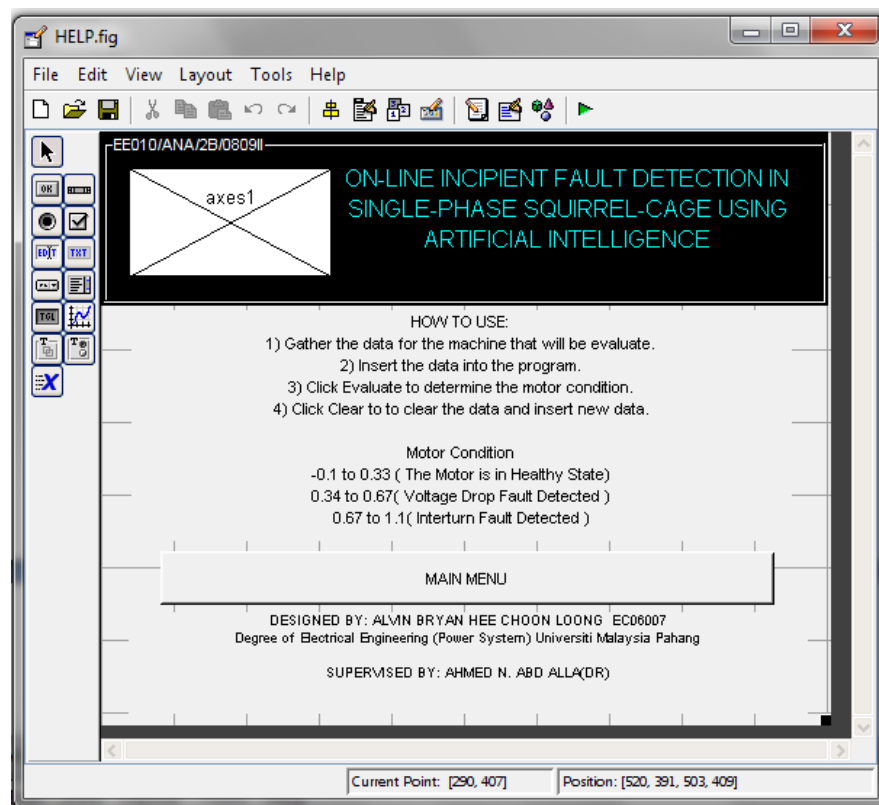


Figure 3.17: Creating the Visual Aspect of the Help GUI

After setting the Tag property and String property for all of the other components, Save the figure to a file. For this the figure is saved as 'HELP'. When the figure is saved, two files will be created on disk with the same name but different extents. The HELP.fig file contains the actual GUI that you have created,

and the HELP.m is an M-file contains the code to load the figure and skeleton callbacks for each GUI element

3.4.5 Programming a GUI

After laying out the GUI and setting component properties, the next step is to program the GUI. The user programs the GUI by coding one or more callbacks for each of its components. Callbacks are functions that execute in response to some action by the user. A typical action is clicking a push button.

A GUI's callbacks are found in an M-file that GUIDE generates automatically. GUIDE adds templates for the most commonly used callbacks to this M-file, but the user may want to add others. M-file Editor is used to edit this file.

For the entire created GUI, it also uses the dialog box, message box and image to make the GUI more attractive for user . All of the coding can be referred on the appendix.

3.4.5.1 Programming the Main Menu

The Main Menu consists of three push button. Start and Help push button is used to call another GUI and Exit push button to close the GUI.

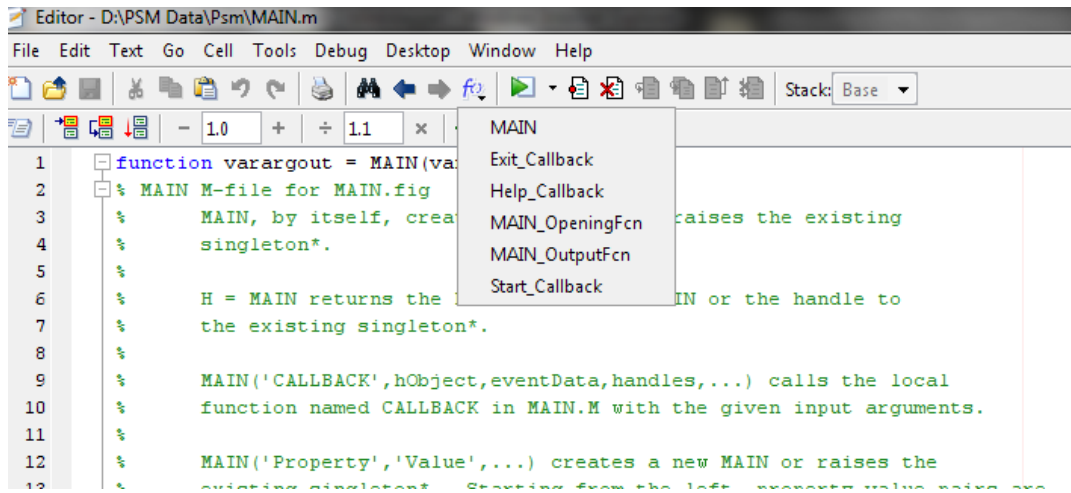


Figure 3.18: Show functions button which enable user to select components of the GUI

Click on the ‘Show functions’ button to which will bring up a list of the functions within the .m file. Select any callback function to add appropriate code to the callback of each component.

When ‘Start_Callback’ is selected, it will show the following code.

```

% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

To determine the function of this ‘START’ push button, appropriate code is added. The code will execute when the ‘START’ push button is being push. It will close the Main Menu and call the PSM_GUI.fig (Evaluate).

```

user_response = PSM_GUI, close MAIN;
```

For the ‘HELP’ push button uses the same concept. By selecting ‘Help_Callback’ it will show the generated code. Add the code to call the HELP.fig. It will close the Main Menu and call the HELP.fig (HELP).

```

user_response = HELP, close MAIN;
```

3.4.5.2 Programming START GUI

For the START GUI, it is a little different from the Main Menu where in this GUI, the user is required to insert data. For the edit text component, certain code is needed in order to obtain the data that being inserted by the user.

The code below is inserted below the generated 'edit_Ia_Callback' code to get the data that the user inserted. Since are the data variables of *String* type, and not *Number* type the input must be converted into number .

```
input = get(handles.edit_Ia,'String'); %get the input from the edit
text field
input = str2num(input); %change from string to number
```

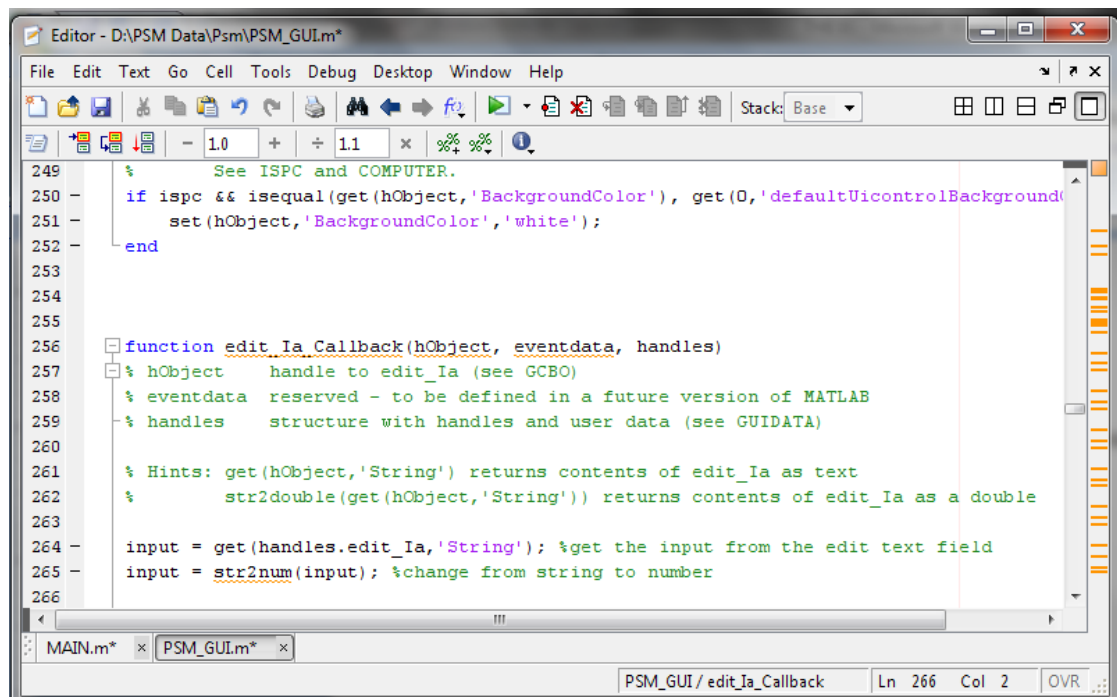


Figure 3.19: The code block for 'edit_Ia' edit text components

For the rest of the edit text, the same procedures need to be done to get the data from the user.

For the 'Evaluate' push button all of the data need to be defined and converted into number type because it will be used as the inputs for the ANN created.

The code below is inserted below the generated code for the 'Evaluate_pushbutton_Callback'

```
a = get(handles.edit_Load, 'String');
b = get(handles.edit_Vi, 'String');
c = get(handles.edit_Ia, 'String');
d = get(handles.edit_Im, 'String');
e = get(handles.edit_Nr, 'String');
f = get(handles.edit_Te, 'String');
```

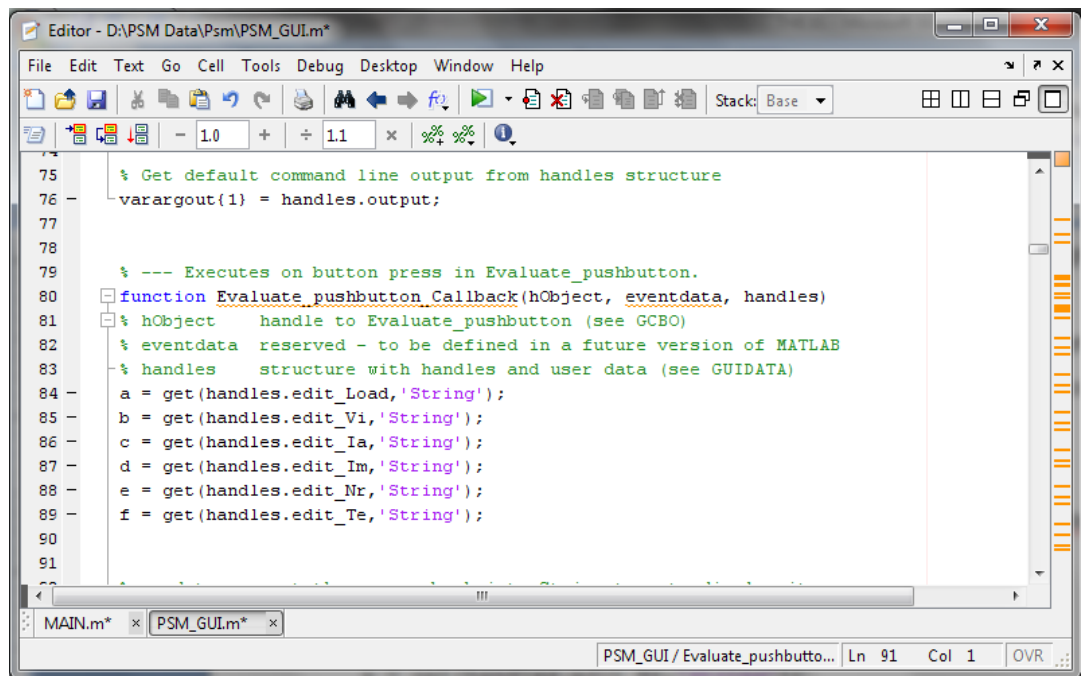


Figure.3.20: The code blocks for 'Evaluate_pushbutton_Callback' edit text components

The data from the edit text will then be the inputs for the ANN and placed into the following code. Y is the output of the ANN and it is used to determine the motor condition of the motor.

The output will be displayed a static text 'answer' on the GUI.

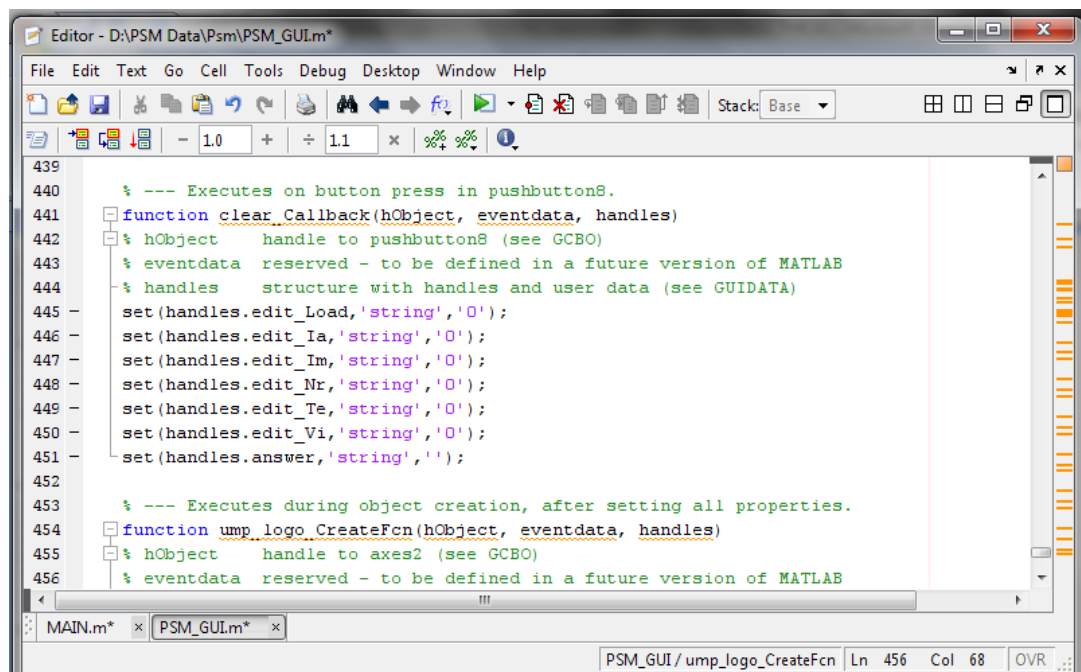
```
X = [str2num(a)    str2num(b)    str2num(c)    str2num(d)    str2num(e)
str2num(f)];
X=X';

Y = sim(net,X)

set(handles.answer,'String',Y);
guidata(hObject, handles);
```

For the 'CLEAR' push button, it is used to reset the data inserted. The following code is inserted below the 'Clear_ Callback'. This will clear all of the inserted data and the result of the ANN. New data can be inserted after clearing the data to evaluate the machine by using another set of data.

```
set(handles.edit_Load,'string','0');
set(handles.edit_Ia,'string','0');
set(handles.edit_Im,'string','0');
set(handles.edit_Nr,'string','0');
set(handles.edit_Te,'string','0');
set(handles.edit_Vi,'string','0');
set(handles.answer,'string','');
```

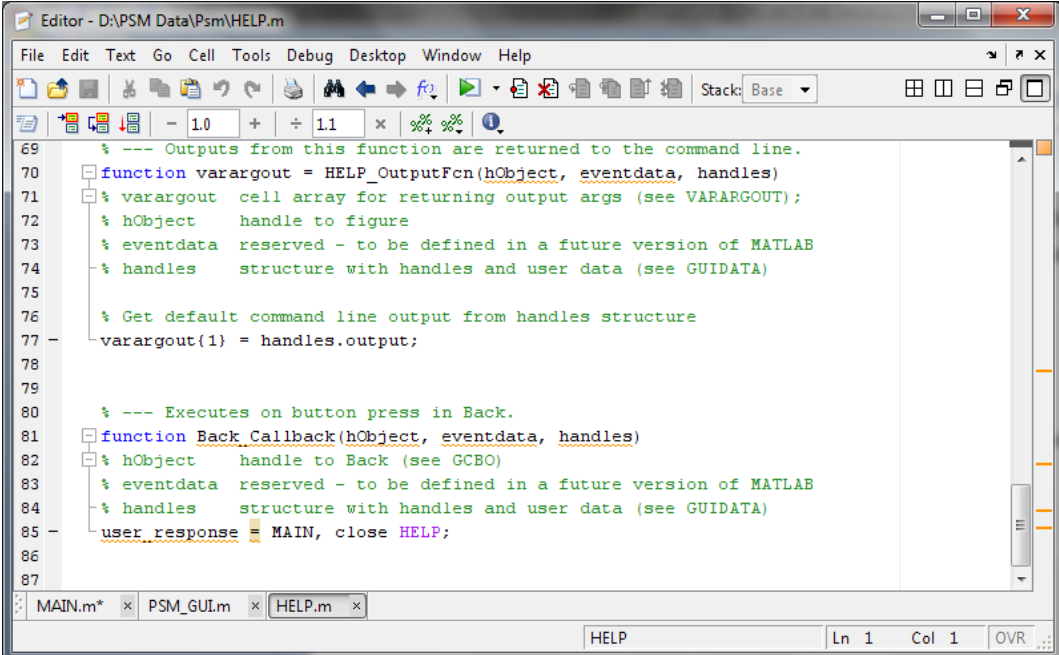


```
Editor - D:\PSM Data\Psm\PSM_GUI.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
439
440 % --- Executes on button press in pushbutton8.
441 function clear_Callback(hObject, eventdata, handles)
442 % hObject    handle to pushbutton8 (see GCBO)
443 % eventdata  reserved - to be defined in a future version of MATLAB
444 % handles    structure with handles and user data (see GUIDATA)
445 set(handles.edit_Load,'string','0');
446 set(handles.edit_Ia,'string','0');
447 set(handles.edit_Im,'string','0');
448 set(handles.edit_Nr,'string','0');
449 set(handles.edit_Te,'string','0');
450 set(handles.edit_Vi,'string','0');
451 set(handles.answer,'string','');
452
453 % --- Executes during object creation, after setting all properties.
454 function ump_logo_CreateFcn(hObject, eventdata, handles)
455 % hObject    handle to axes2 (see GCBO)
456 % eventdata  reserved - to be defined in a future version of MATLAB
MAIN.m* x PSM_GUI.m* x
PSM_GUI/ump_logo_CreateFcn Ln 456 Col 68 OVR
```

Figure.3.21: The code blocks for 'clear_Callback' push button

3.4.5.3 Programming HELP GUI

This GUI is used to guide first time user on how to use the software created. This GUI consists only one push button to go back to the Main Menu.



```
Editor - D:\PSM Data\PSM\HELP.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
69 % --- Outputs from this function are returned to the command line.
70 function varargout = HELP_OutputFcn(hObject, eventdata, handles)
71 % varargout cell array for returning output args (see VARARGOUT);
72 % hObject handle to figure
73 % eventdata reserved - to be defined in a future version of MATLAB
74 % handles structure with handles and user data (see GUIDATA)
75
76 % Get default command line output from handles structure
77 varargout{1} = handles.output;
78
79
80 % --- Executes on button press in Back.
81 function Back_Callback(hObject, eventdata, handles)
82 % hObject handle to Back (see GCBO)
83 % eventdata reserved - to be defined in a future version of MATLAB
84 % handles structure with handles and user data (see GUIDATA)
85 user_response = MAIN, close HELP;
86
87
```

Figure 3.22: The code blocks for 'Back_Callback' push button

For the 'Main Menu' push button uses the same concept like the Main Menu GUI. By selecting 'Back_Callback' it will show the generated code. Add the code to call the MAIN.fig. It will close the HELP.fig and call the MAIN.fig (HELP).

```
user_response = MAIN, close HELP;
```

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Graphical User Interface (GUI)

4.1.1 Main Menu GUI

For this project, On-line incipient fault detection in single-phase squirrel-cage induction motors using Artificial Intelligence, a GUI has been created by using MATLAB. This GUI is consists of three main part START, HELP and EXIT. Each push button will execute its own GUI. For example if the 'HELP' push button is clicked, it will call 'HELP.fig' which is the GUI for the 'HELP' push button and for the rest will be shown later.

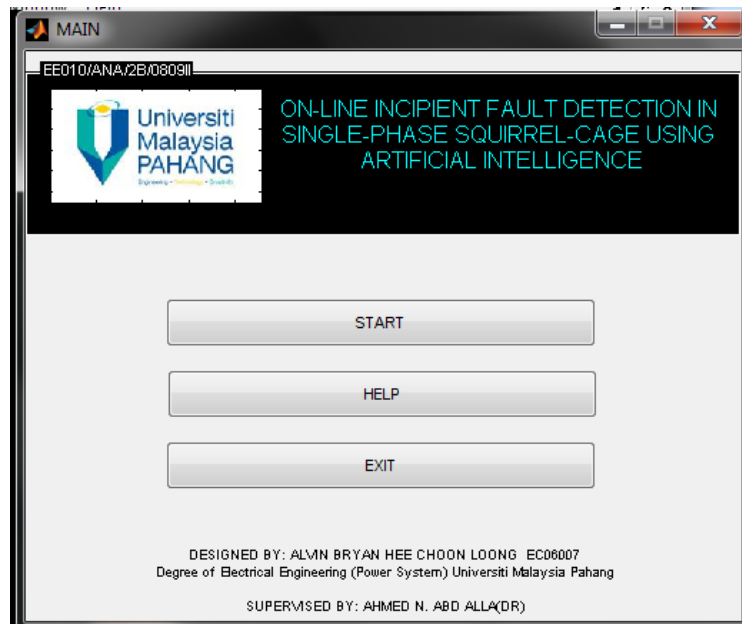


Figure 4.1: Main Menu which consists of three push button

4.1.2 Help GUI

This GUI will be called when the 'HELP' push button from the Main Menu GUI is clicked.

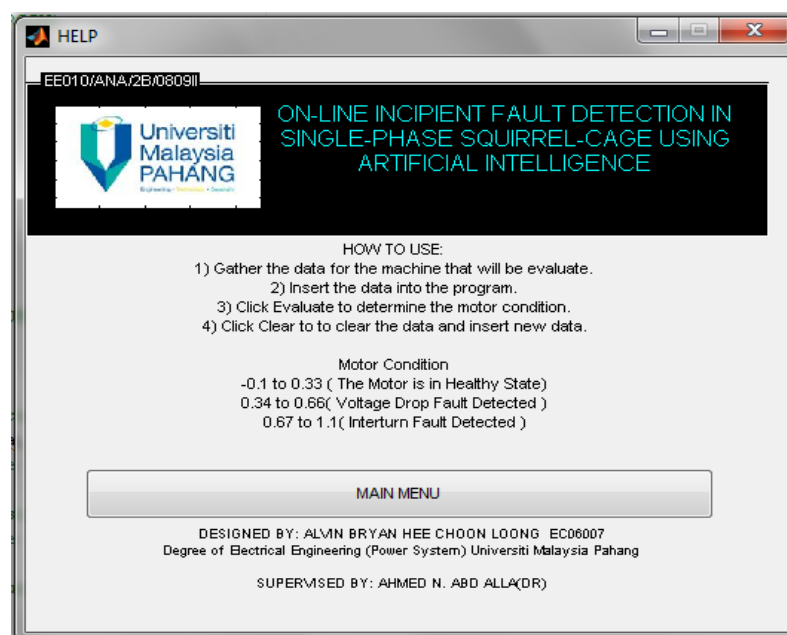


Figure 4.2: Help Menu which consist of one push button

4.1.3 Start GUI

This GUI will be called when the 'START' push button from the Main Menu GUI is clicked. The Start GUI is the most important part of this project. This is where the ANN can be tested by inserting data into the GUI. The data inserted into the GUI will be sent into the ANN as the inputs of the ANN.

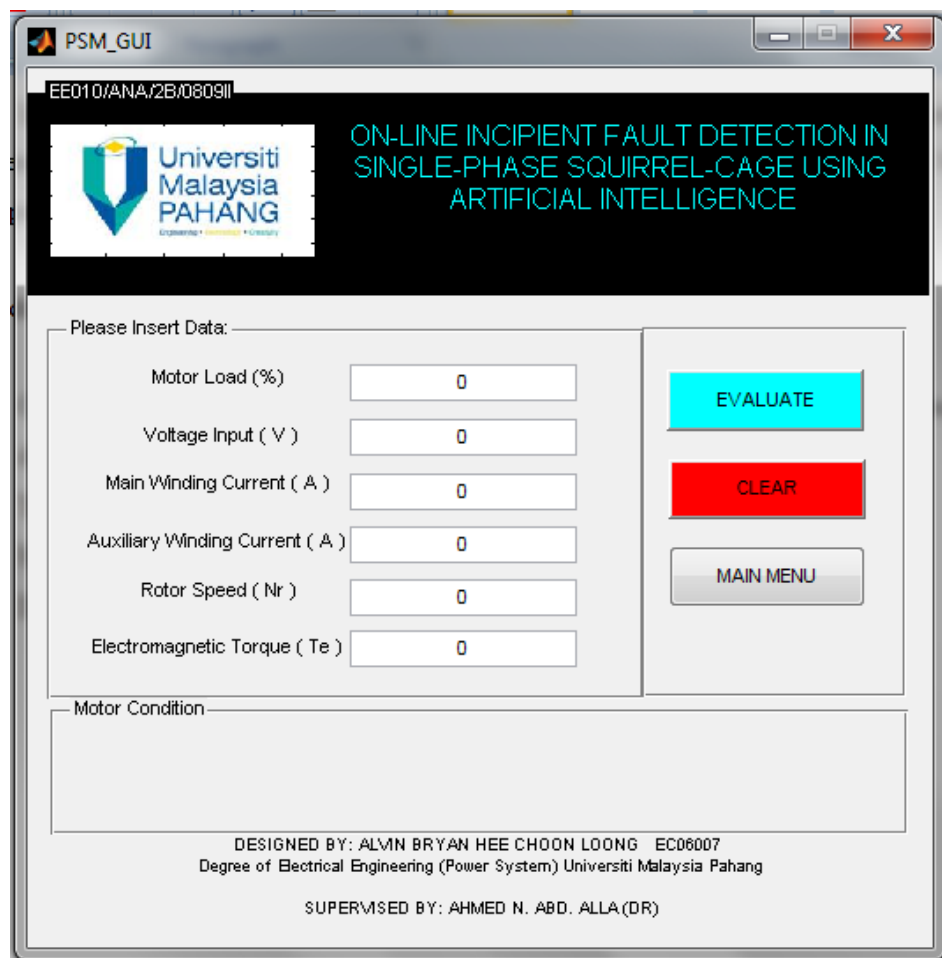


Figure 4.3: Start Menu which consist of edit text component for data to be inserted.

The data inserted in to the GUI have to meet certain specification for each parameter. It would not accept value that are not numbers and value that is not logic for the type of machine that is used. For example, a 220V motor will not run under 100V.

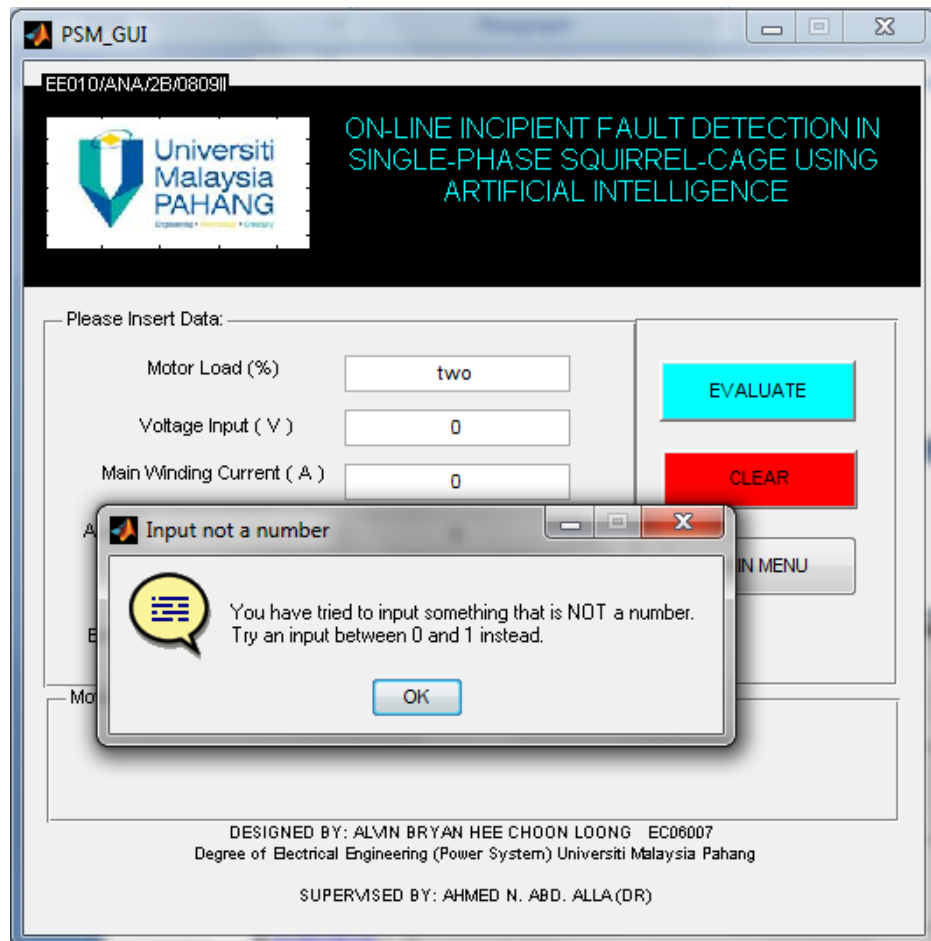


Figure.4.4: The Message box indicate that the input is not a number

In this GUI, a message box will pop out if the value inserted does not meet the specifications. The Figure 4.4 shows when value that is not a number type, it will pop out a message box that indicate that the value is not valid.

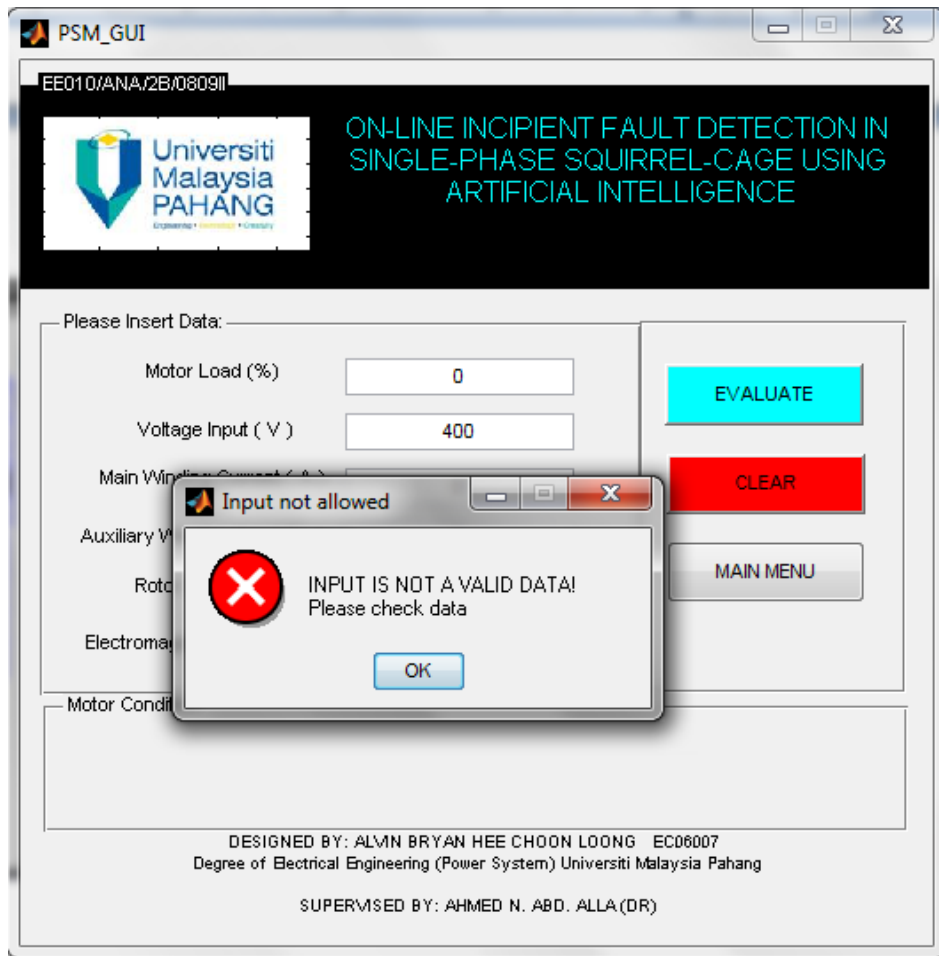


Figure 4.5: The Message box indicate that the input is not valid.

In this GUI, a message box will pop out if the value inserted does not meet the specifications. The Figure 4.5 shows when value does not make sense, it will pop out a message box that indicates that the value is not valid and request so that the data is checked.

If the data is properly inserted within the parameter specification, the ANN will be able to evaluate the data of the inserted machine and determine the condition of the motor whether it is in a healthy state, voltage drop fault or interturn fault.

After inserting the data, click 'EVALUATE' push button. Message box will pop up and show the condition of the machine evaluated.



Figure 4.6: The Message box indicate that the motor is in a healthy condition

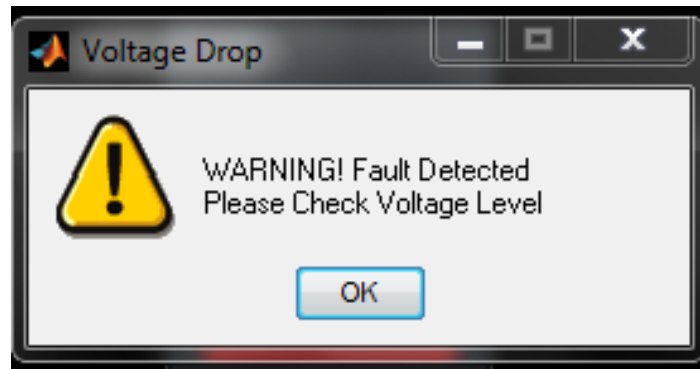


Figure 4.7: The Message box indicate that a voltage drop fault detected

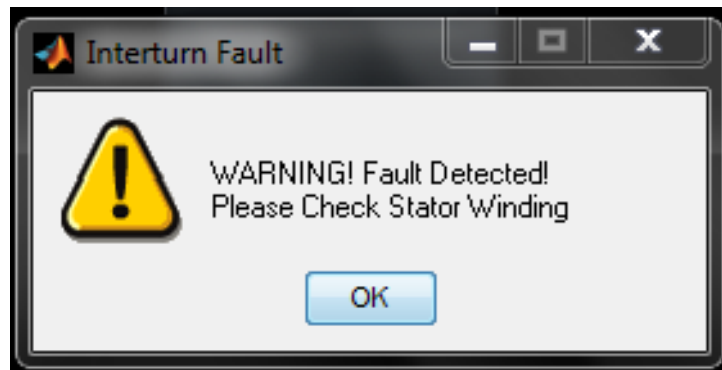


Figure 4.8: The Message box indicate that interturn fault detected

The message box that pop out is the result of the motor condition. When the motor condition is between -0.1 to 0.33 it is in a healthy state, if motor condition is between 0.34 to 0.66 a voltage drop detected and if motor condition is between 0.67 to 1.1 interturn fault detected.

4.2 Testing the ANN

The evaluation of the machine 1 with the parameters below:

Load (per unit)	= 0.75 p.u
Voltage Input (V_i)	= 109.6 V
Main Winding Current (I_a)	= 3.304 A
Auxiliary Winding Current (I_m)	= 0.0002092 A
Rotor Speed (N_r)	= 22.12 rad/s
Electromagnetic Torque (T_e)	= 0.04742 Nm

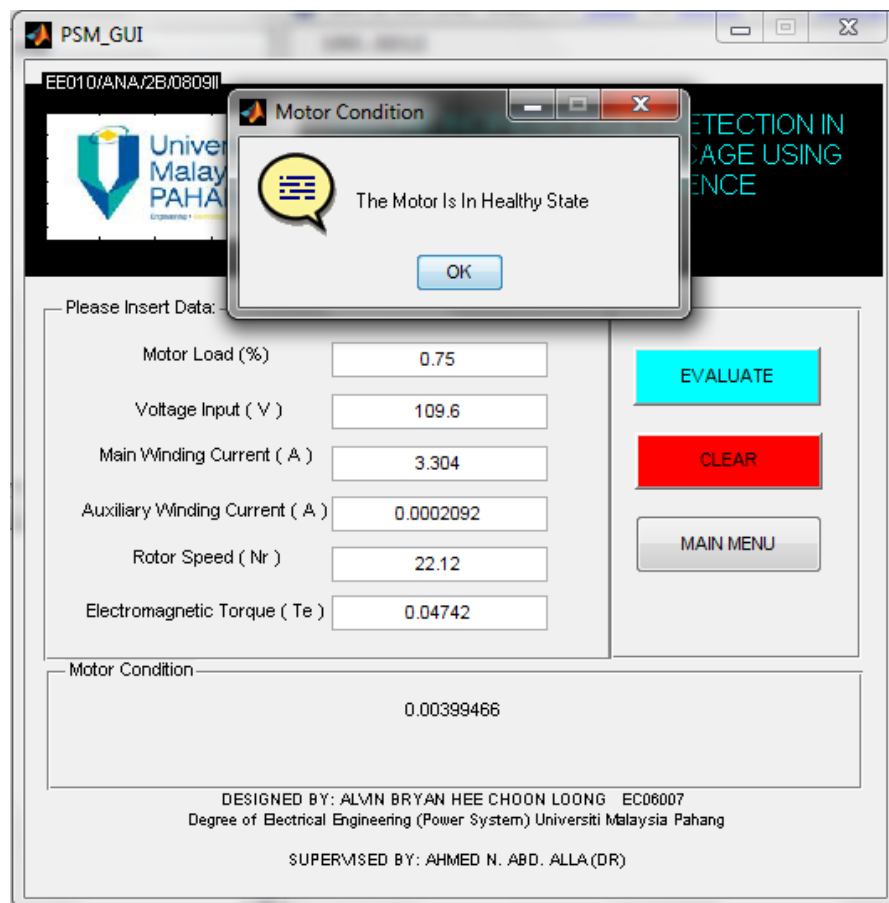


Figure 4.9: The result of the testing for machine 1

The Figure 4.9 shows the result of the evaluation for the machine 1. From the message box, it shows that the motor is in a healthy state. The motor condition also shows the value 0.00399466 which in the range of -0.1 to 0.33 (Healthy state).

The evaluation of the machine 2 with the parameters below:

Load (per unit)	= 0.25 p.u
Voltage Input (Vi)	= 104.3 V
Main Winding Current (Ia)	= 2.827 A
Auxiliary Winding Current (Im)	= 0.0002082 A
Rotor Speed (Nr)	= 2.422 rad/s
Electromagnetic Torque (Te)	= 0.03887 Nm

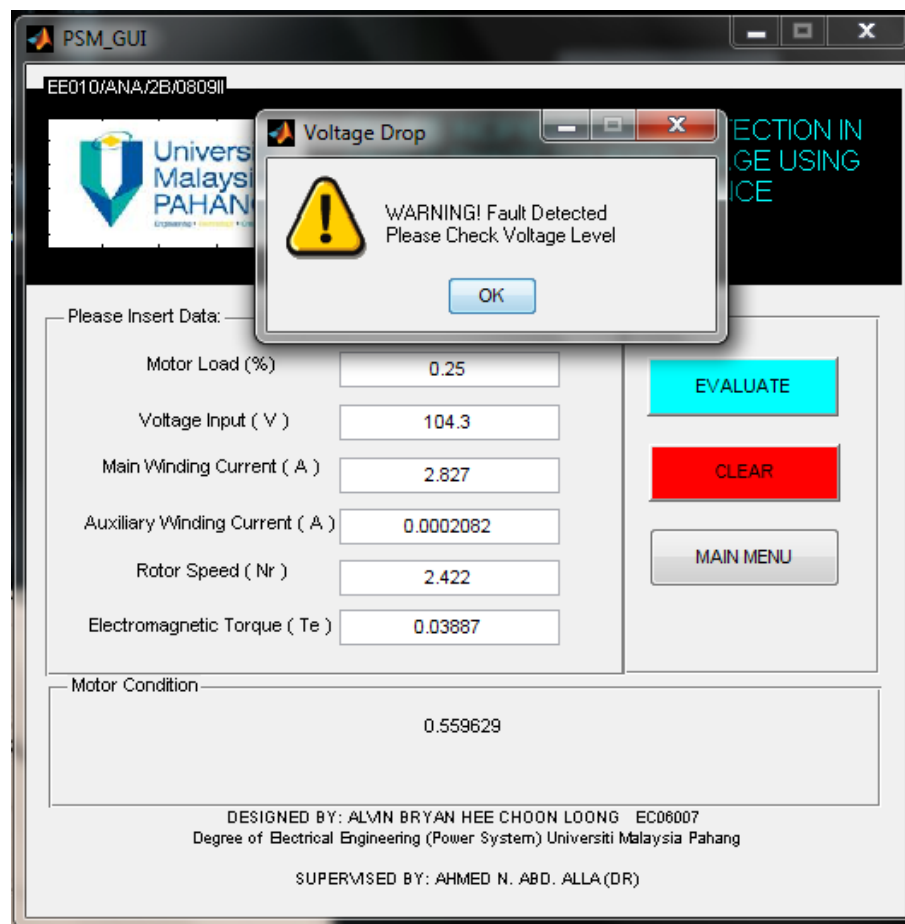


Figure4.10: The result of the testing for machine 2

The Figure 4.10 shows the result of the evaluation for the machine 2. From the message box, it shows that ANN detected a voltage drop fault. The motor condition also shows the value 0.559629 which in the range of 0.34 to 0.66 (Voltage Drop Fault).

The evaluation of the machine 3 with the parameters below:

Load (per unit)	= 0.5 p.u
Voltage Input (Vi)	= 110.4 V
Main Winding Current (Ia)	= 3.087 A
Auxiliary Winding Current (Im)	= 0.0002149 A
Rotor Speed (Nr)	= 29.15 rad/s
Electromagnetic Torque (Te)	= 0.03013 Nm

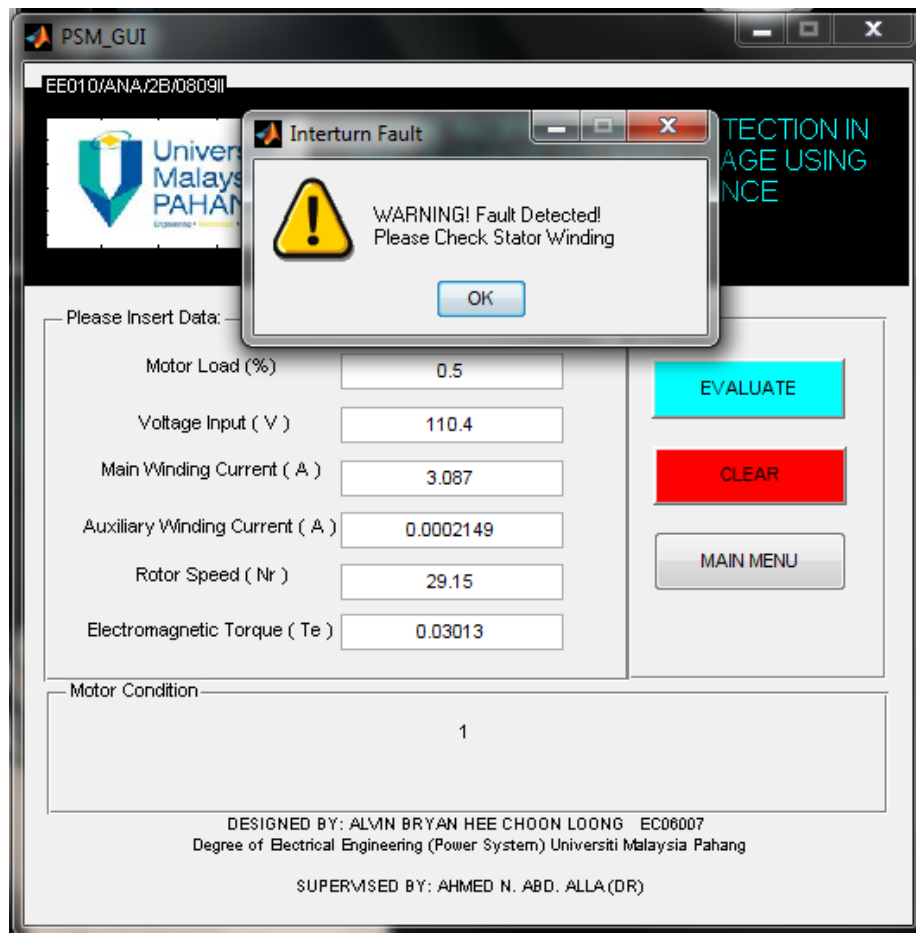


Figure 4.11: : The result of the testing for machine 3

The Figure 4.11 shows the result of the evaluation for the machine 3. From the message box, it shows that ANN detected a interturn fault. The motor condition also shows the value 1 which in the range of 0.67 to 1.1 (Interturn Fault).

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

For this final year project, which is the development of On-line Incipient Fault Detection in Single-Phase Squirrel-Cage Using Artificial Intelligence, the objectives has been achieved which are to improve and create other alternatives for condition monitoring of single phase squirrel cage induction motors by developing a Neural Network that is capable of detecting faults in single phase squirrel-cage induction motors while gaining knowledge on the use of Neural Network in MATLAB. The Graphical User Interface (GUI) been created is also user friendly, low cost and easy to maintain.

5.2 Suggestion for Future Work

The project is functioning well from the user friendly interface. However, in implementing it, need to be improved to a more advanced and better application in the future. For future improvement, several suggestions are proposed:

- Experimental data be taken from a real motor.
By creating a fault in a real motor, the experimental data could be more accurate and reliable.
- Interface the ANN created with real motor.
Instead of inserting the data into the GUI, interfacing the program with real motor could really give more reliability in monitoring the motor condition.

REFERENCES

1. http://www.mathworks.com/academia/student_version/r2009a_products/neuralnet.pdf
2. M.E.H. Benbouzid,(2000) “*A Review of Induction Motors Signature Analysis as a Medium for Faults Detection*”.
3. 20 March 20[1]Barker, S., (2000), *Power Engineering Journal*,
4. C.M. Riley, B.K. Lin, T.G. Habetler and G.B. Kliman, (1999) “*Stator Current Harmonics and their Causal Vibrations: A Preliminary Investigation of Sensorless Vibration Monitoring Applications*,”.
5. Filippetti, F., Franceschini, G., Tassoni, C(2005), ‘*Neural networks aided on-line diagnostics of induction motor rotor faults*’
6. Huang, X., Habetler, T.G.; Harley, R.G.(2004) ‘*Detection of rotor eccentricity faults in closed-loop drive-connected induction motors using an artificial neural network*’ .
7. .Li, B., Chow, M-Y., Tipsuwan, Y., Hung, J.C.(2000), ‘*Neural-network-based motor rolling bearing*’.
8. S.Nandi, H.A Toliyat, Li, (1999) *Condition Monitoring and Fault Diagnosis of Electrical Motors—A Review*.
9. Salles, G., Filippetti, F., Tassoni, C., Crellet, G., Francenschini, G.(2000), ‘*Monitoring of induction motor load by neural network techniques*’.
10. M.L. Sin, W.L. Soong and N. Ertugrul(2003) ‘*Induction machine on-line condition monitoring and fault diagnosis*’.

11. Tallam, R.M., Habetler, T.G., Harley, R.G., Gritter, D.J., Burton, B.H.,(2000), '*Neural network based on-line stator winding turn fault detection for induction motors*'.
12. Tavner, P., Penman, J., (1987), '*Condition Monitoring of Electrical Machines*', Research studies press LTD, 1987. ISSN/ISBN: 0-86380-061-0, 302 p.
13. Thorsen, O. V., Dalva, M., (1999), '*Failure identification and analysis for high voltage induction molar*s in the petrochemical industry', IEEE Transactions on Industry Applications, Vol.35, No. 4, pp. 810-818.
14. P. Vas,(1993) *Condition Monitoring, Diagnosis and Parameter Estimation of Electrical Machines*. Oxford, U.K.: Oxford Univ. Press, 1993.
15. Marian Dumitru Negrea., (2006). '*Electromagnetic flux monitoring for detecting faults in electrical machines*', Master Thesis, Helsinki University of Technology
16. http://en.wikipedia.org/wiki/Radial_basis_function_network

APPENDIX A

Main Menu GUI coding

```
function varargout = MAIN(varargin)
% MAIN M-file for MAIN.fig
%   MAIN, by itself, creates a new MAIN or raises the existing
%   singleton*.
%
%   H = MAIN returns the handle to a new MAIN or the handle to
%   the existing singleton*.
%
%   MAIN('CALLBACK',hObject,eventData,handles,...) calls the
%   local
%   function named CALLBACK in MAIN.M with the given input
%   arguments.
%
%   MAIN('Property','Value',...) creates a new MAIN or raises the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before MAIN_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application
%   stop. All inputs are passed to MAIN_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help MAIN

% Last Modified by GUIDE v2.5 21-Oct-2009 23:47:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @MAIN_OpeningFcn, ...
                  'gui_OutputFcn',  @MAIN_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes during object creation, after setting all properties.
```



```

% --- Executes just before MAIN is made visible.
function MAIN_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to MAIN (see VARARGIN)

[x,map]=imread('UMP.JPG');
image(x)
colormap(map)

% Choose default command line output for MAIN
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes MAIN wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = MAIN_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = PSM_GUI, close MAIN;

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
% hObject    handle to Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = HELP, close MAIN;

% --- Executes on button press in Exit.
function Exit_Callback(hObject, eventdata, handles)
% hObject    handle to Exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pos_size = get(handles.figure1,'Position');
% Call modaldlg with the argument 'Position'.
user_response = modaldlg('Title','Confirm Close');
switch user_response
case {'No'}
    % take no action

```

```
case 'Yes'  
    % Prepare to close GUI application window  
    %  
    %  
    %  
    delete(handles.figure1)  
end
```

APPENDIX B

Help GUI coding

```
function varargout = HELP(varargin)
% HELP M-file for HELP.fig
%   HELP, by itself, creates a new HELP or raises the existing
%   singleton*.
%
%   H = HELP returns the handle to a new HELP or the handle to
%   the existing singleton*.
%
%   HELP('CALLBACK',hObject,eventData,handles,...) calls the
%   local
%   function named CALLBACK in HELP.M with the given input
%   arguments.
%
%   HELP('Property','Value',...) creates a new HELP or raises the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before HELP_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application
%   stop. All inputs are passed to HELP_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help HELP

% Last Modified by GUIDE v2.5 26-Feb-2006 14:11:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @HELP_OpeningFcn, ...
                  'gui_OutputFcn',  @HELP_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before HELP is made visible.
```

```

function HELP_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin   command line arguments to HELP (see VARARGIN)

% Choose default command line output for HELP
handles.output = hObject;

[x,map]=imread('UMP.JPG');
image(x)
colormap(map)

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes HELP wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = HELP_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in Back.
function Back_Callback(hObject, eventdata, handles)
% hObject    handle to Back (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
user_response = MAIN, close HELP;

```

APPENDIX C

Start GUI coding

```
function varargout = PSM_GUI(varargin)

% PSM_GUI M-file for PSM_GUI.fig
%   PSM_GUI, by itself, creates a new PSM_GUI or raises the
%   existing
%   singleton*.
%
%   H = PSM_GUI returns the handle to a new PSM_GUI or the handle
%   to
%   the existing singleton*.
%
%   PSM_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in PSM_GUI.M with the given input
%   arguments.
%
%   PSM_GUI('Property','Value',...) creates a new PSM_GUI or
%   raises the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before PSM_GUI_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application
%   stop. All inputs are passed to PSM_GUI_OpeningFcn via
%   varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help PSM_GUI

% Last Modified by GUIDE v2.5 21-Oct-2009 23:29:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @PSM_GUI_OpeningFcn, ...
                  'gui_OutputFcn',  @PSM_GUI_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
end
```

```

else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before PSM_GUI is made visible.
function PSM_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to PSM_GUI (see VARARGIN)

% Choose default command line output for PSM_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PSM_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = PSM_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in Evaluate_pushbutton.
function Evaluate_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Evaluate_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = get(handles.edit_Load, 'String');
b = get(handles.edit_Vi, 'String');
c = get(handles.edit_Ia, 'String');
d = get(handles.edit_Im, 'String');
e = get(handles.edit_Nr, 'String');
f = get(handles.edit_Te, 'String');

% need to convert the answer back into String type to display it

P = [0.00  107.9  2.965  0.0002305  100.3  0.05529;
     0.25  17.26  5.579  1.184e-009  2.335  3.67;
     0.50  47.09  4.211  0.0004339  11.64  2.066;
     0.75  109.6  3.304  0.0002092  22.12  0.04742;
     1.00  109.08  3.555  0.0002095  0.05377  0.05058;
     0.00  104.2  2.789  0.0002106  0.09707  0.03887;
     0.25  104.3  2.827  0.0002082  2.422  0.03571;
     0.50  49.53  4.232  3.727e-005  37.25  1.898;

```

```

0.75 104.1 3.178 0.0002033 2.348 0.04365;
1.00 104.1 3.555 0.0003083 23.31 0.04951;
0.00 10.4 5.186 0.0008248 7.115 3.546;
0.25 24.08 2.644 0 61.94 1.478;
0.50 110.4 3.087 0.0002149 29.15 0.03013;
0.75 109.5 3.264 0.0002153 4.123 0.03934;
1.00 113.3 3.471 0.0002109 69.85 0.0993];
T=[0;0;0;0;0;.5;.5;.5;.5;.5;1;1;1;1;1];
P=P';
T=T';

%Training
eg = 0.02; % sum-squared error goal
sc = 1; % spread constant
net = newrb(P,T,eg,sc);

%3, Test

X = [str2num(a) str2num(b) str2num(c) str2num(d) str2num(e)
      str2num(f)];
X=X';

Y = sim(net,X)

set(handles.answer,'String',Y);
guidata(hObject, handles);
input = Y;

if input <= 0.3333 & input >= -0.1

    %this is the first line of the msgbox
    msgboxText{1} = 'The Motor Is In Healthy State';

    %notice that msgboxText is a Cell array!

    %this command creates the actual message box
    msgbox(msgboxText,'Motor Condition', 'help');

elseif input <= 0.6666 & input >= 0.3334
    msgboxText{1} = 'WARNING! Fault Detected';
    msgboxText{2} = 'Please Check Voltage Level';
    msgbox(msgboxText,'Voltage Drop', 'warn');

elseif input <= 1.1 & input >= 0.6667
    msgboxText{1} = 'WARNING! Fault Detected!';
    msgboxText{2} = 'Please Check Stator Winding';
    msgbox(msgboxText,'Interturn Fault', 'warn');
end

% --- Executes on button press in exit_pushbutton.
function exit_pushbutton_Callback(hObject, eventdata, handles)
% hObject handle to exit_pushbutton (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

```

```

user_response = MAIN, close PSM_GUI;

function edit_Load_Callback(hObject, eventdata, handles)
% hObject      handle to edit_Load (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Load as text
%         str2double(get(hObject,'String')) returns contents of
%         edit_Load as a double
input = get(handles.edit_Load,'String'); %get the input from the
edit text field
input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox
    msgboxText{1} = 'You have tried to input something that is NOT
        a number.';
    %this is the second line
    msgboxText{2} = 'Try an input between 0 and 1 instead.';

    %notice that msgboxText is a Cell array!

    %this command creates the actual message box
    msgbox(msgboxText,'Input not a number', 'help');

elseif input < 0 || input > 1 %if the input is less than 0 or
greater than 1
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText,'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Load_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edit_Load (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_Vi_Callback(hObject, eventdata, handles)
% hObject      handle to edit_Vi (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

```



```

% Hints: get(hObject,'String') returns contents of edit_Vi as text
%       str2double(get(hObject,'String')) returns contents of
%       edit_Vi as a double
input = get(handles.edit_Vi,'String'); %get the input from the edit
text field
input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox
    msgboxText{1} = 'You have tried to input something that is NOT
        a number.';
    %this is the second line
    msgboxText{2} = 'Try an input between 0 and 113.3 instead.';

    %notice that msgboxText is a Cell array!

    %this command creates the actual message box
    msgbox(msgboxText,'Input not a number', 'help');

elseif input < 0 || input > 113.3 %if the input is less than 0 or
greater than 113.3
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText,'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Vi_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Vi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_Ia_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Ia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Ia as text
%       str2double(get(hObject,'String')) returns contents of
%       edit_Ia as a
%       double
input = get(handles.edit_Ia,'String'); %get the input from the edit
text field

```

```

input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox
    msgboxText{1} = 'You have tried to input something that is NOT
        a number.';
    %this is the second line
    msgboxText{2} = 'Try an input between 0 and 5.58 instead.';

    %notice that msgboxText is a Cell array!

    %this command creates the actual message box
    msgbox(msgboxText,'Input not a number', 'help');

elseif input < 0 || input > 5.58 %if the input is less than 0 or
    greater than 5.58
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText,'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Ia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Ia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_Im_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Im (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Im as text
%       str2double(get(hObject,'String')) returns contents of
        edit_Im as a double
input = get(handles.edit_Im,'String'); %get the input from the edit
    text field
input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox

```

```

msgboxText{1} = 'You have tried to input something that is NOT
    a number.';
%this is the second line
msgboxText{2} = 'Try an input between 0 and 0.000825 instead.';

%notice that msgboxText is a Cell array!

%this command creates the actual message box
msgbox(msgboxText,'Input not a number', 'help');

elseif input < 0 || input > 0.000825 %if the input is less than 0 or
    greater than 100
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText,'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Im_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Im (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_Nr_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Nr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_Nr as text
%       str2double(get(hObject,'String')) returns contents of
    edit_Nr as a double
input = get(handles.edit_Nr,'String'); %get the input from the edit
    text field
input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox
    msgboxText{1} = 'You have tried to input something that is NOT
        a number.';
    %this is the second line
    msgboxText{2} = 'Try an input between 0 and 100.3 instead.';

```

```

%notice that msgboxText is a Cell array!

%this command creates the actual message box
msgbox(msgboxText, 'Input not a number', 'help');

elseif input < 0 || input > 100.3 %if the input is less than 0 or
    greater than 100.3
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText, 'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Nr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Nr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
    get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function edit_Te_Callback(hObject, eventdata, handles)
% hObject    handle to edit_Te (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of edit_Te as text
%       str2double(get(hObject, 'String')) returns contents of
    edit_Te as a double
input = get(handles.edit_Te, 'String'); %get the input from the edit
    text field
input = str2num(input); %change from string to number

if isempty(input) %if the input is not a number

    %this is the first line of the msgbox
    msgboxText{1} = 'You have tried to input something that is NOT
        a number.';
    %this is the second line
    msgboxText{2} = 'Try an input between 0 and 3.627 instead.';

    %notice that msgboxText is a Cell array!

    %this command creates the actual message box
    msgbox(msgboxText, 'Input not a number', 'help');

```

```

elseif input < 0 || input > 3.627 %if the input is less than 0 or
    greater than 3.627
    %this is the first line of the msgbox
    msgboxText{1} = 'INPUT IS NOT A VALID DATA!';
    %this is the second line
    msgboxText{2} = 'Please check data';

    msgbox(msgboxText,'Input not allowed', 'error');
end

% --- Executes during object creation, after setting all properties.
function edit_Te_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_Te (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.edit_Load,'string','0');
set(handles.edit_Ia,'string','0');
set(handles.edit_Im,'string','0');
set(handles.edit_Nr,'string','0');
set(handles.edit_Te,'string','0');
set(handles.edit_Vi,'string','0');
set(handles.answer,'string','');

% --- Executes during object creation, after setting all properties.
function ump_logo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
    called

% Hint: place code in OpeningFcn to populate axes
[c,map]=imread('UMP','JPG');
image(c)
set(gca,'visible','off')

```

APPENDIX D

Close Dialog Box coding

```
function varargout = modaldlg(varargin)
% MODALDLG M-file for modaldlg.fig
%   MODALDLG by itself, creates a new MODALDLG or raises the
%   existing singleton*.
%
%   H = MODALDLG returns the handle to a new MODALDLG or the
%   handle to
%   the existing singleton*.
%
%   MODALDLG('CALLBACK',hObject,eventData,handles,...) calls the
%   local
%   function named CALLBACK in MODALDLG.M with the given input
%   arguments.
%
%   MODALDLG('Property','Value',...) creates a new MODALDLG or
%   raises the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before modaldlg_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application
%   stop. All inputs are passed to modaldlg_OpeningFcn via
%   varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help modaldlg

% Last Modified by GUIDE v2.5 10-Oct-2009 23:50:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @modaldlg_OpeningFcn, ...
                  'gui_OutputFcn',  @modaldlg_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```

% --- Executes just before modaldlg is made visible.
function modaldlg_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to modaldlg (see VARARGIN)

% Choose default command line output for modaldlg
handles.output = 'Yes';

% Update handles structure
guidata(hObject, handles);

% Insert custom Title and Text if specified by the user
% Hint: when choosing keywords, be sure they are not easily confused
% with existing figure properties.  See the output of set(figure)
for
% a list of figure properties.
if(nargin > 3)
    for index = 1:2:(nargin-3),
        if nargin-3==index, break, end
        switch lower(varargin{index})
            case 'title'
                set(hObject, 'Name', varargin{index+1});
            case 'string'
                set(handles.text1, 'String', varargin{index+1});
            end
        end
    end
end

% Determine the position of the dialog - centered on the callback
figure
% if available, else, centered on the screen
FigPos=get(0, 'DefaultFigurePosition');
OldUnits = get(hObject, 'Units');
set(hObject, 'Units', 'pixels');
OldPos = get(hObject, 'Position');
FigWidth = OldPos(3);
FigHeight = OldPos(4);
if isempty(gcf)
    ScreenUnits=get(0, 'Units');
    set(0, 'Units', 'pixels');
    ScreenSize=get(0, 'ScreenSize');
    set(0, 'Units', ScreenUnits);

    FigPos(1)=1/2*(ScreenSize(3)-FigWidth);
    FigPos(2)=2/3*(ScreenSize(4)-FigHeight);
else
    GCBFOldUnits = get(gcf, 'Units');
    set(gcf, 'Units', 'pixels');
    GCBFPos = get(gcf, 'Position');
    set(gcf, 'Units', GCBFOldUnits);
    FigPos(1:2) = [(GCBFPos(1) + GCBFPos(3) / 2) - FigWidth / 2, ...
                  (GCBFPos(2) + GCBFPos(4) / 2) - FigHeight / 2];
end
FigPos(3:4)=[FigWidth FigHeight];
set(hObject, 'Position', FigPos);
set(hObject, 'Units', OldUnits);

```

```

% Show a question icon from dialogicons.mat - variables
    questIconData
% and questIconMap
load dialogicons.mat

IconData=questIconData;
questIconMap(256,:) = get(handles.figure1, 'Color');
IconCMap=questIconMap;

Img=image(IconData, 'Parent', handles.axes1);
set(handles.figure1, 'Colormap', IconCMap);

set(handles.axes1, ...
    'Visible', 'off', ...
    'YDir'    , 'reverse'    , ...
    'XLim'    , get(Img,'XData'), ...
    'YLim'    , get(Img,'YData') ...
);

% Make the GUI modal
set(handles.figure1,'WindowStyle','modal')

% UIWAIT makes modaldlg wait for user response (see UIRESUME)
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = modaldlg_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% The figure can be deleted now
delete(handles.figure1);

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.output = get(hObject,'String');

% Update handles structure
guidata(hObject, handles);

% Use UIRESUME instead of delete because the OutputFcn needs
% to get the updated handles structure.
uiresume(handles.figure1);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```



```

% handles      structure with handles and user data (see GUIDATA)

handles.output = get(hObject, 'String');

% Update handles structure
guidata(hObject, handles);

% Use UIRESUME instead of delete because the OutputFcn needs
% to get the updated handles structure.
uiresume(handles.figure1);

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject      handle to figure1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

if isequal(get(handles.figure1, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
    uiresume(handles.figure1);
else
    % The GUI is no longer waiting, just close it
    delete(handles.figure1);
end

% --- Executes on key press over figure1 with no controls selected.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject      handle to figure1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Check for "enter" or "escape"
if isequal(get(hObject, 'CurrentKey'), 'escape')
    % User said no by hitting escape
    handles.output = 'No';

    % Update handles structure
    guidata(hObject, handles);

    uiresume(handles.figure1);
end

if isequal(get(hObject, 'CurrentKey'), 'return')
    uiresume(handles.figure1);
end

```