# SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM

MUHAMMAD ALIF BIN MOHD NOR

UNIVERSITI MALAYSIA PAHANG

# SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM

MUHAMMAD ALIF BIN MOHD NOR

A thesis submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Electrical Engineering (Power system)

Faculty of Electrical & Electronics Engineering
University Malaysia Pahang

NOVEMBER 2009

## DECLARATION

I declare that this thesis entitled "*SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM*" is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

| | |
|---|---|
| Signature | : .................................................... |
| Name | : Muhammad Alif Bin Mohd Nor |
| Date | : 23 November 2009 |

## DEDICATIONS

*Specially dedicated to*

*My beloved father and mother,*

*To my siblings and friends,*

*My supervisor and lecturers,*

*Thanks for all of the encouragement and support.*

# ACKNOWLEDGEMENTS

Praise and glory to Allah S.W.T, God of all creation and greetings and salutations we bring forth to our Prophet Muhammad S.A.W for overseeing this final year project one and constantly guiding this project towards completion.

I’m as the author of this thesis wishes and express the greatest appreciation to Dr. Ahmed N Abd Alla as my supervisor of this final year project. Once, nobody believes the project will manage successfully, but with his dedication and guidance, the project is able to complete on time. Special thanks for him for the opportunity given and for the efforts towards the completion of the project

I want to expresses my gratitude to my beloved family for the support given by them for the commitment and support although indirectly involves in this project, for fully efforts to help in the completion of this final year project.

Last but not least, to my entire friends, your help and support are really appreciated and will remember forever, InsyaAllah. Thank you.

## ABSTRACT

The title of this project is Simulation of electrical faults of three phase induction motor drive system. Induction motor or asynchronous motor is a type of alternating current motor where power is supplied to the rotor by means of electromagnetic induction. Induction motor is now the preferred choice for industrial motor due to their rugged construction, absence of brushes (which are required in most DC motors) and the ability to control the speed of motor. The faults that can occur in the three-phase induction motor and its driver can be divided into two parts; internal and external faults. The internal fault of induction motors account for the proportion almost more than 70% of induction motor failures. This project will cover and study a few type of internal and external faults, which is the stator inter-turn short circuit, unbalanced voltage supply and the single phase open circuit fault. The study of induction motor is crucial and important so that the lifespan of the motor can be prolonged. In this project MATLAB SIMULINK is used to simulate the induction motor faults and analyze the condition. The simulation file is then compiled along with a GUI to simplify the overall process and improves the user friendliness to users.

# ABSTRAK

Tajuk projek ini ialah simulasi kerosakan ke atas motor aruhan tiga fasa dan sistem pacuannya. Motor aruhan, juga dikenali sebagai enjin tidak segerak, adalah sejenis motor arus ulang alik dimana kuasanya dibekalkan kepada rotor melalui proses induksi elektromagnetik. Motor aruhan kini menjadi pilihan utama dalam industri permotoran kerana ciri-ciri dan kelebihan yang dimilikinya, terutamanya pembinaan yang kasar, ketiadaan berus (yang mana paling diperlukan dalam DC motor) dan kelajuan enjin yang boleh dikawal. Kerosakan yang boleh berlaku kepada motor aruhan tiga fasa boleh dibahagikan kepada dua bahagian, iaitu kerosakan dalaman dan kerosakan luaran. Kerosakan dalaman motor aruhan adalah penyumbang utama kepada kegagalan motor aruhan untuk berfungsi, iaitu sebanyak 70%. Projek ini akan mempelajari dan merangkumi beberapa jenis kerosakan dalaman dan luaran motor aruhan, iaitu litar pintas pemegun antara giliran, bekalan voltan tidak seimbang dan satu fasa litar terbuka. Kajian terhadap motor aruhan adalah penting supaya jangka hayat motor dapat dipanjangkan. Untuk projek ini, MATLAB SIMULINK akan diguna pakai bagi membuat simulasi kerosakan motor aruhan dan menganalisis kondisi motor tersebut. Fail simulasi kemudiannya akan dihimpunkan di dalam satu GUI untuk memudahkan keseluruhan proses dan meningkatkan tahap mesra pengguna utiliti ini.

## TABLE OF CONTENTS

| CHAPTER | TITLE | PAGE |
|---|---|---|
| | **DECLARATION** | ii |
| | **DEDICATION** | iii |
| | **ACKNOWLEDGMENTS** | iv |
| | **ABSTRACT** | v |
| | **ABSTRAK** | vi |
| | **TABLE OF CONTENTS** | vii |
| | **LIST OF FIGURES** | x |
| | **LIST OF SYMBOLS** | xii |
| | **LIST OF APPENDIXES** | xiv |
| **I** | **INTRODUCTION** | |
| | 1.1 Chapter Overview | 1 |
| | 1.2 Background | 2 |
| | 1.3 Problem Statement | 2 |
| | 1.4 Objectives | 3 |
| | 1.5 Scopes of Study | 3 |
| | 1.6 Thesis Outline | 4 |
| **II** | **LITERATURE REVIEW** | |
| | 2.1 Chapter Overview | 6 |
| | 2.2 Definition of Three Phase Induction Motor | 7 |
| | 2.2.1 Stator | 8 |

| | | |
|---|---|---|
| | 2.2.2 Rotor | 9 |
| | 2.3 Simulation of Electrical Faults for Induction Motor | 11 |
| | 2.3.1 Stator Inter-turn Short Circuit | 11 |
| | 2.3.2 Unbalanced Voltage Supply | 12 |
| | 2.4 MATLAB | 16 |
| | 2.5 Simulink | 17 |
| | 2.6 Block Set Power System | 20 |
| | 2.6.1 Area of the Power System Block Set | 20 |
| III | **METHODOLOGY** | |
| | 3.1 Chapter Overview | 23 |
| | 3.2 Developing the MATLAB Program | 25 |
| | 3.3 Developing the Simulation Model | 26 |
| | 3.3.1 Stator Inter-turn Short Circuit Model | 33 |
| | 3.3.2 Unbalanced Voltage Supply Model | 34 |
| | 3.3.3 Single-Phase Open Circuit Fault Model | 36 |
| | 3.4 Developing the GUI | 37 |
| | 3.4.1 Main Window M-FILE Description | 41 |
| | 3.4.2 Main Window M-FILE | 41 |
| | 3.4.1.2 Description | 41 |
| IV | **RESULT & DISCUSSION** | |
| | 4.1 Chapter Overview | 45 |
| | 4.2 Simulation of Healthy State Motor | 45 |
| | 4.3 Simulation of Faulty State Motor | 51 |
| | 4.3.1 Simulation for Inter-turn Short Circuit | 52 |
| | 4.3.2 Simulation for Unbalanced Voltage Supply | 56 |
| | 4.3.3 Simulation for Single Phase Open Circuit | 60 |
| | 4.4 Result Analysis | 64 |

4.4.1 Inter-turn Short Circuit 64
4.4.2 Unbalanced voltage supply 65
4.4.3 Single phase open circuit 67

**V CONCLUSION & RECOMMENDATION**
5.1 Conclusion 71
5.2 Recommendations 72

**REFERENCES** 73

**APPENDIX A** 75

## LIST OF TABLES

| FIGURE NO. | TITLE | PAGE |
|---|---|---|
| 3.1 | Description of the Motor Block Parameters | 3 |

# LIST OF FIGURES

| FIGURE NO. | TITLE | PAGE |
|---|---|---|
| 2.1 | Squirrel Cage Rotor | 9 |
| 2.2 | Wound Rotor | 10 |
| 2.3 | Balanced Voltage Supply | 13 |
| 2.4 | Unbalanced Voltage Supply | 13 |
| 2.5 | Library Browser for Simulink | 19 |
| 2.6 | Window for Model Using Functional Block | 19 |
| 2.7 | Library Browser for SimPower System | 22 |
| 3.1 | Work Flow of the Project | 24 |
| 3.2 | Opening M-file Window | 25 |
| 3.3 | New M-file Window | 26 |
| 3.4 | Three Phase Asynchronous Machine Model | 27 |
| 3.5 | Three Phase Induction Motor Model | 28 |
| 3.6 | Block Parameters of AC Voltage Source | 28 |
| 3.7 | Block Parameters of 1.5HP – 450V 50Hz – 1475rpm | 29 |
| 3.8 | Step Block | 31 |
| 3.9 | Three Phase Induction Motor model | 32 |
| 3.10 | Three Resistors before the Motor (healthy state) | 33 |
| 3.11 | One of the Resistor were Short Circuited | 34 |
| 3.12 | Voltage Setting for Healthy State Motor | 35 |
| 3.13 | Voltage Setting for Faulty State Motor | 35 |
| 3.14 | Voltage Supply for Healthy Model | 36 |
| 3.15 | Voltage Supply for Faulty State | 37 |
| 3.16 | GUI Window | 38 |

3.17 Blank GUI Window 38
3.18 Example of Created GUI 40
3.19 Example of the Generated Program 40
3.20 Display the Program for Opening Function 41
3.21 Display the Opening Function Position 42
3.22 Display the Callback Function Line Position 43
4.1 Main GUI Window 46
4.2 Display the Confirm Action Window 46
4.3 Display the Healthy State Simulation Window 47
4.4 The Three Phase Voltages 48
4.5 The Three Phase Currents 48
4.6 Rotor Current 49
4.7 Stator Current 49
4.8 Electromagnetic Torque 50
4.9 Motor Speed 50
4.10 GUI for Faulty State 51
4.11 Simulation for Inter-turn Short Circuit 52
4.12 Three Phase Voltages 53
4.13 Three Phase Currents 53
4.14 Rotor Current 54
4.15 Stator Current 54
4.16 Electromagnetic Torque 55
4.17 Motor Speed 55
4.18 Simulation for Unbalanced Voltage Supply 56
4.19 The Three Phase Voltages 57
4.20 Three Phase Currents 57
4.21 Rotor Current 58
4.22 Stator Current 58
4.23 Electromagnetic Torque 59
4.24 Motor Speed 59
4.25 Simulation for Single Phase Open Circuit 60
4.26 The Three Phase Voltages 61
4.27 Three Phase Currents 61
4.28 Rotor Current 62

4.29 Stator Current 62
4.30 Electromagnetic Torque 63
4.31 Motor Speed 63

# LIST OF APPENDIX

| **APPENDIX** | **TITLE** | **PAGE** |
|---|---|---|
| A | Program for GUI | 77 |

# CHAPTER 1

# INTRODUCTION

## 1.1 Chapter Overview

The title of this project is Simulation of electrical faults of three phase induction motor drive system. The faults that can occur in the three-phase induction motor and its driver can be divided into two parts; internal and external faults. The internal fault of induction motors account for the proportion almost more than 70% of induction motor failures. As example, stator inter-turn short circuit. For external faults, it happens at voltage supply, such as unbalance voltage supply and one phase open circuit.

From the faults that might occur, this project will analyze and simulate the electrical faults of three-phase induction motor and its drive. The modeling of the induction motor and the simulation of electrical faults in three phase induction motor drive will be done by using MATLAB tools.

This project can be divided into 3 different stages:

- Data extraction
- Develop Simulation
- Develop GUI

## 1.2 Background

Simulation technique has been proved to have many advantages rather than just doing a practical attempt. Especially for this project, the faults are intentionally being created to motor, to study the behavior of the motor when faulted. If we were doing this project with an actual motor, it will be a waste the motor gets damaged.

## 1.3 Problem Statement

The increased in demand has greatly improved the approach of fault detection in polyphase induction motor. Monitoring the motor condition in an early stage is crucial to detect any fault to eliminate the hazards of severe motor faults and preventing damage.

Nowadays simulation technique is implemented to improve traditional techniques, where the results can be obtained instantaneously after it analyzes the input data of the motor. In fact, some company use simulation technique while designing their new product.

In this project MATLAB SIMULINK is used to simulate the induction motor faults and analyze the condition.

## 1.4 Objectives

Simulation of electrical faults of three phase induction motor drive system is developed with the listed objectives below:

- To study the features for a various kind of faults of the induction motor and its drive system.
- To build an induction motor model and to simulate the internal and external faults using MATLAB tools.

## 1.5 Scopes of study

There are several scopes for the project:

- This project is mainly about a simulation of faults that may occur in three-phase induction motor and its drive.
- This project is use to detect faults in three phase induction motors only. It is the most popular poly phase induction motor in industry.
- The modeling and simulation will be done by using MATLAB tools.
- The type of faults which will be studied is limited to a few types of external and internal faults.

## 1.6 Thesis Outline

This thesis consists of five chapters. In the first chapter, this chapter discussed the overall idea of this project including objectives of project, problem statement, the scope of this project and summary of this thesis.

Chapter 2 discussed more on theory and literature review that have been done. It is well discusses about the MATLAB, basic concept of the fault in induction motor, SIMULINK and parameters related to this project.

Chapter 3 described briefly the methodology of the data extraction, simulation development and GUI development for this project. The figures, tables and extra information are aided into this chapter to be the benchmark thesis in development of Simulation of electrical faults of three phase induction motor drive system.

Chapter 4 presents a discussion of the implementation, result and analysis of the whole project. This chapter also explains the reasons of some failure.

Chapter 5 provides the conclusions of the project. There are also several suggestions that can be used for future implementation or upgrading for this project.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Chapter overview

This chapter includes all the paper works and related research as well as the studies regards to this project. The chapter includes all important studies which have been done previously by other research work. The related works have been referred carefully since some of the knowledge and suggestions from the previous work can be implemented for this project.

Literature review was an ongoing process throughout the whole process of the project. It is very essential to refer to the variety of sources in order to gain more knowledge and skills to complete this project. These sources include reference books, thesis, journals and also the materials obtained from internet.

At the beginning of the project, the basic concept of fault in induction motor has been well acquired. In addition, the function of all the components used in this project such as basic operation of MATLAB Simulink, and so on was explored first before starting the project.

## 2.2 Definition of three phase induction motor

The AC induction motor is a rotating electric machine designed to operate from a three-phase source of alternating voltage. The stator is a classic three phase stator with the winding displaced by 120°. The most common type of induction motor has a squirrel cage rotor in which aluminum conductors or bars are shorted together at both ends of the rotor by cast aluminum end rings. When three currents flow through the three symmetrically placed windings, a sinusoidally distributed air gap flux generating the rotor current is produced. The interaction of the sinusoidally distributed air gap flux and induced rotor currents produces a torque on the rotor. The mechanical angular velocity of the rotor is lower than the angular velocity of the flux wave by so called slip velocity. [1]

AC induction motors are the most common motors used in industrial motion control systems, as well as in main powered home appliances. Simple and rugged design, low-cost, low maintenance and direct connection to an AC power source are the main advantages of AC induction motors. [6]

The induction motor essentially consists of two parts:

1. Stator
2. Rotor

The supply is connected to the stator and the rotor received power by induction caused by the stator rotating flux, hence the motor obtains its name –induction motor. [2]

### 2.2.1 Stator

The stator consists of a cylindrical laminated & slotted core placed in a frame of rolled or cast steel. The frame provides mechanical protection and carries the terminal box and the end covers with bearings. In the slots of a 3-phase winding of insulated copper wire is distributed which can be wound for 2, 4, 6 etc. poles. The rotor consists of a laminated and slotted core tightly pressed on the shaft [3]

The stator is made up of several thin laminations of aluminum or cast iron. They are punched and clamped together to form a hollow cylinder (stator core) with slots as shown in Figure 1. Coils of insulated wires are inserted into these slots. Each grouping of coils, together with the core it surrounds, forms an electromagnet (a pair of poles) on the application of AC supply. The number of poles of an AC induction motor depends on the internal connection of the stator windings. The stator windings are connected directly to the power source. Internally they are connected in such a way, that on applying AC supply, a rotating magnetic field is created. [6]

### 2.2.2 Rotor

The rotor consists of a laminated and slotted core tightly pressed on the shaft. There are two general types of rotors:

1. The squirrel-cage rotor
2. The wound (or slip ring) rotor

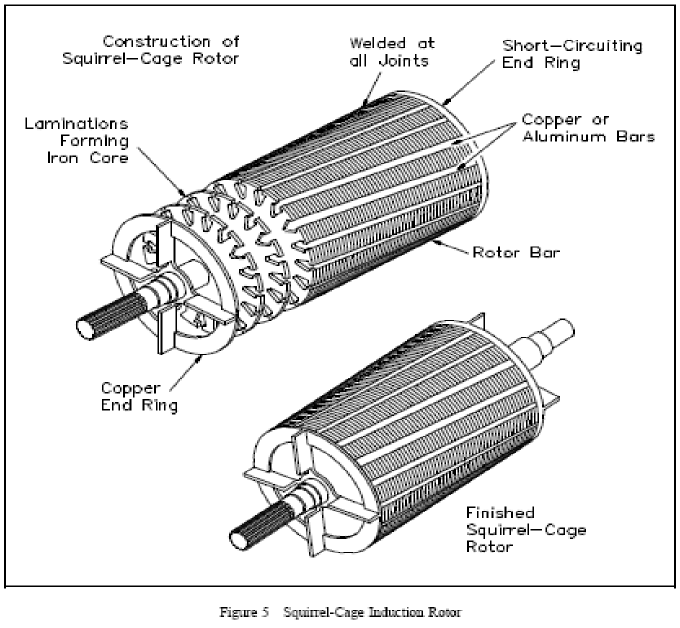

Construction of Squirrel—Cage Rotor
Welded at all Joints
Short—Circuiting End Ring
Laminations Forming Iron Core
Copper or Aluminum Bars
Rotor Bar
Copper End Ring
Finished Squirrel—Cage Rotor
Figure 5 Squirrel-Cage Induction Rotor

**Figure 2.1**: Squirrel Cage Rotor

In the squirrel-cage rotor, the rotor winding consists of single copper or aluminum bars placed in the slots and short-circuited by end-rings on both sides of the rotor. [3]

The field windings in the stator of an induction motor set up a rotating magnetic field around the rotor. The relative motion between this field and the rotation of the rotor induces electric current in the conductive bars. In turn these currents lengthwise in the conductors react with the magnetic field of the motor to produce force acting at a tangent to the rotor, resulting in torque to turn the shaft. In effect the rotor is carried around with the magnetic field but at a slightly slower rate of rotation. The difference in speed is called "slip" and increases with load. [4]

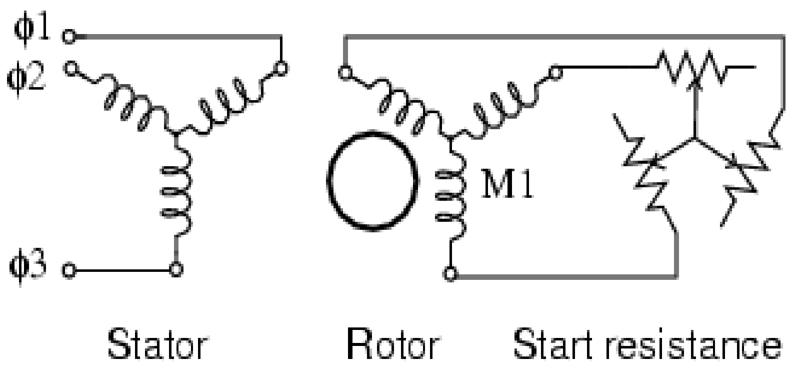


**Figure 2.2**: Wound rotor

A wound rotor induction motor has a stator like the squirrel cage induction motor, but a rotor with insulated windings brought out via slip rings and brushes. However, no power is applied to the slip rings. Their sole purpose is to allow resistance to be placed in series with the rotor windings while starting. This resistance is shorted out once the motor is started to make the rotor look electrically like the squirrel cage counterpart. [5]

## 2.3 Simulation of electrical faults for induction motor

Computer simulation of electric motor operation is particularly useful for gaining an insight into their dynamic behavior and electro-mechanical interaction. A suitable model enables motor faults to be simulated and the change in corresponding parameters to be predicted without physical experimentation. [8]

Modeling of induction motors with shorted turns is the first step in the design of turn fault detection systems. Simulation of transient and steady state behavior of motors with these models enable correct evaluation of the measured data by diagnostics techniques. [7]

This paper will cover few types of electrical faults. The electrical faults can be divided into two parts, the external and internal faults. An external fault is a fault that occurs outside the motor, and the internal fault is a fault that occurs inside the motor. For the external fault, we will cover the unbalance voltage supply and one-phase open circuit fault. For the internal fault we will cover the stator inter-turn short circuit.

### 2.3.1 Stator inter-turn short circuit

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.

Stator winding faults of synchronous generator are considered serious problems because of the damage associated with high fault currents and high cost of maintenance. A high speed bias differential relay is normally used to detect three phase, phase-phase and double phase to ground faults. In case of inter-turn winding fault the current on both side of the winding is same. Due to this factor we cannot adapt the differential scheme of protection for inter-turn winding fault.[10]

When there is an insulation failure in between the winding inter-turns they get short circuited and the amount of winding involved in generation gets reduced. As the amount of winding under generating action is reduced the amount of current produced by induction principle also gets reduced. This reduces the power generated and affects the life time of the winding. When this problem is left undealt the inter winding insulation gets affected there by further reducing the amount of winding involved in generation. This fault will completely damage the winding at the extreme stage. The cost of winding is very high when compared to the protection methods which can adapt.[10]

### 2.3.2 Unbalanced voltage supply

Voltage unbalance is regarded as a power quality problem of significant concern at the electricity distribution level. Although the voltages are quite well balanced at the generator and transmission levels the voltages at the utilization level can become unbalanced due to the unequal system impedances and the unequal distribution of single-phase loads. [11]

An excessive level of voltage unbalance can have serious impacts on mains connected induction motors. The level of current unbalance that is present is several times the level of voltage unbalance. Such an unbalance in the line currents can lead to excessive losses in the stator and rotor that may cause protection systems to operate causing loss of production. Although induction motors are designed to tolerate a small level of unbalance they have to be derated if the unbalance is excessive. If operated at the nameplate rated capacity without derating the useful life of such induction motors can become quite short. If an induction motor is oversized to a given application then some level of protection is built into its operation although the motor does not operate at the best efficiency and power factor. [11]

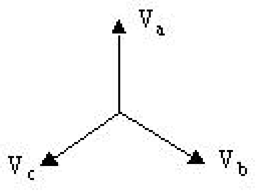


**Figure 2.3** Balanced Voltage Supply

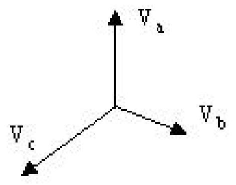


**Figure 2.4** Unbalanced Voltage Supply

Three phase induction motors are designed and manufactured such that all three phases of the winding are carefully balanced with respect to the number of turns, placement of the winding, and winding resistance. When line voltages applied to a polyphase induction motor are not exactly the same, unbalanced currents will flow in the stator winding, the magnitude depending upon the amount of unbalance. A small amount of voltage unbalance may increase the current an excessive amount. The effect on the motor can be severe and the motor may overheat to the point of burnout.

The voltages should be evenly balanced as closely as can be read on the usually available commercial voltmeter.

- Effect on performance – General

  The effect of unbalanced voltages on polyphase induction motors is equivalent to the introduction of a "negative sequence voltage" having a rotation opposite to that occurring with balanced voltages. This negative sequence voltage produces in the air gap a flux rotating against the rotation of the rotor, tending to produce high currents. A small negative sequence voltage may produce in the windings currents considerably in excess of those present under balanced voltage conditions.
- Unbalance Defined

  The voltage unbalance (or negative sequence voltage) in percent may be defined as follows:

$$\text{Percent Voltage Unbalance} = 100 * \frac{\text{Maximum Voltage Deviation From Average Voltage}}{\text{Average Voltage}}$$

*Example:*

With voltages of 220, 215 and 210, the average is 215, the maximum deviation from the average is 5, and the percent unbalance = 100 X 5/215 = 2.3 percent.

- Temperature rise and load carrying capacity

  A relatively small unbalance in voltage will cause a considerable increase in temperature rise. In the phase with the highest current, the percentage increase in temperature rise will be approximately two times the square of the percentage voltage unbalance. The increase in losses and consequently, the increase in average heating of the whole winding will be slightly lower than the winding with the highest current.

To illustrate the severity of this condition, an approximate 3.5 percent voltage unbalance will cause an approximate 25 percent increase in temperature rise.

- Torques

  The locked-rotor torque and breakdown torque are decreased when the voltage is unbalanced. If the voltage unbalance should be extremely severe, the torque might not be adequate for the application.

- Full-load speed

  The full-load speed is reduced slightly when the motor operates at unbalanced voltages.

- Currents

  The locked-rotor current will be unbalanced to the same degree that the voltages are unbalanced but the locked-rotor KVA will increase only slightly.

  The currents at normal operating speed with unbalanced voltages will be greatly unbalanced in the order of approximately 6 to 10 times the voltage unbalance. This introduces a complex problem in selecting the proper overload protective devices, particularly since devices selected for one set of unbalanced conditions may be inadequate for a different set of unbalanced voltages. Increasing the size of the overload protective device is not the solution in as much as protection against heating from overload and from single phase operation is lost.

## 2.4 MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar no interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation. MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

## 2.5 Simulink

Simulink® is software for modeling, simulating, and analyzing dynamic systems. Simulink enables you to pose a question about a system, model it, and see what happens. With Simulink, you can easily build models from scratch, or modify existing models to meet your needs. Simulink supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multi rate — having different parts that are sampled or updated at different rates.

Simulink provides a graphical user interface (GUI) for building models as block diagrams, allowing you to draw models as you would with pencil and paper. Simulink also includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors. If these blocks do not meet your needs, however, you can also create your own blocks. The interactive graphical environment simplifies the modeling process, eliminating the need to formulate differential and difference equations in a language or program.

Models are hierarchical, so you can build models using both top-down and bottom-up approaches. You can view the system at a high level, then double-click blocks to see increasing levels of model detail. This approach provides insight into how a model is organized and how its parts interact.

After a model is defined, it can simulate, using a choice of mathematical integration methods, either from the Simulink menus or by entering commands in the MATLAB’s Command Window. The menus are convenient for interactive work, while the command line is useful for running a batch of simulations.

Using scopes and other display blocks, the simulation result can be seen while the simulation is running. Many parameters can be changed and immediately see what happens for "what if" exploration. The simulation results can be put in the MATLAB workspace for post processing and visualization.

Model analysis tools include linearization and trimming tools, which can be accessed from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes. Because MATLAB and Simulink are integrated, you can simulate, analyze, and revise your models in either environment at any point.

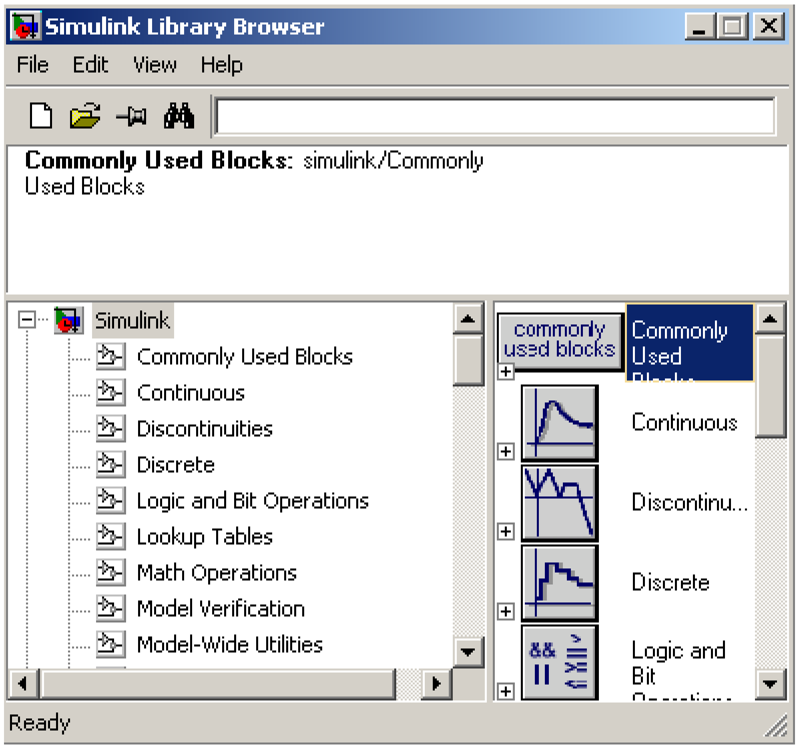


**Figure 2.5**: Library Browser for Simulink

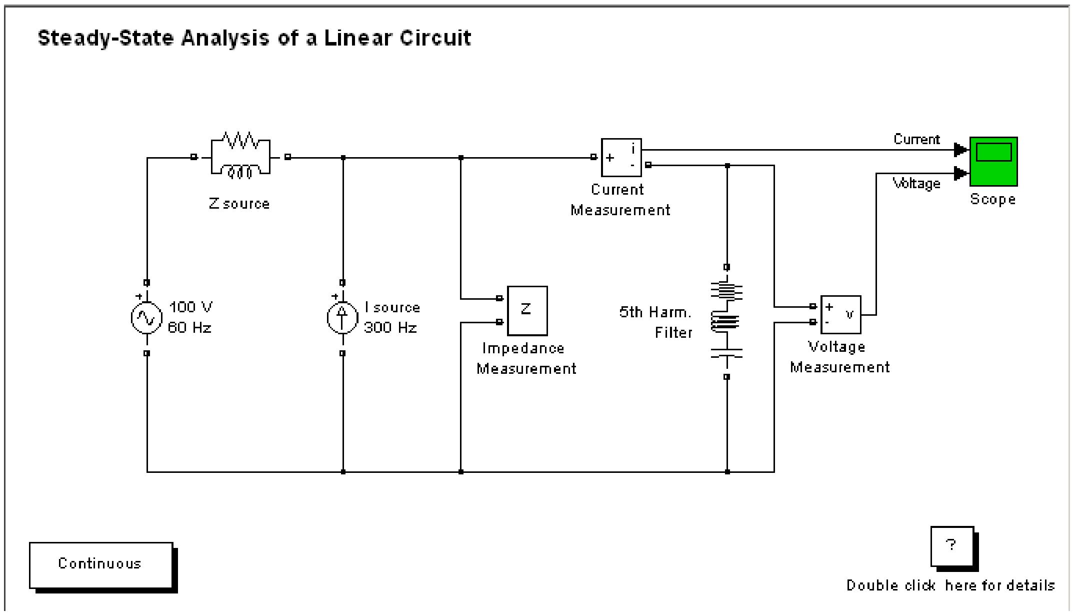


**Figure 2.6**: Window for Model Using Functional Block

## 2.6 Block set power system

SimPowerSystems is a modern design tool that allows scientists and engineers to rapidly and easily build models that simulate power systems. SimPowerSystems uses the Simulink environment, allowing you to build a model using simple click and drag procedures. Not only can you draw the circuit topology rapidly, but your analysis of the circuit can include its interactions with mechanical, thermal, control, and other disciplines. This is possible because all the electrical parts of the simulation interact with the extensive Simulink modeling library. Since Simulink uses MATLAB as its computational engine, designers can also use MATLAB toolboxes and Simulink block sets. SimPowerSystems and SimMechanics share a special Physical Modeling block and connection line interface.

### 2.6.1 Area of the power system block set

**Power system networks**

- RLC branches and loads
- Pi section lines
- Linear and saturable transformers/td
- Surge arrester
- Mutual inductance
- Distributed parameter lines
- AC voltage and current source
- DC voltage sources

**Electric machinery**

- Complete and simplified models of synchronous machines
- Asynchronous machines
- Permanent magnet synchronous machines
- Excitation system
- Hydraulic turbines
- Governors

**Power electronics**

- Diodes
- Simplified and complex thyristors
- GTOs
- Switches
- MOSFETs
- IGBT

**Control and measurement blocks**

- Voltage and current measurement
- RMS measurements
- Active power calculations
- Synchronized 6-pulse generators

**Triphase library**

- Triphase loads and branches
- Pi section lines
- AC voltage sources
- 6-pulse thyristor bridge
- Diode rectifiers
- Triphase transformer in Y-delta, Y-Y, and delta-Y configuration

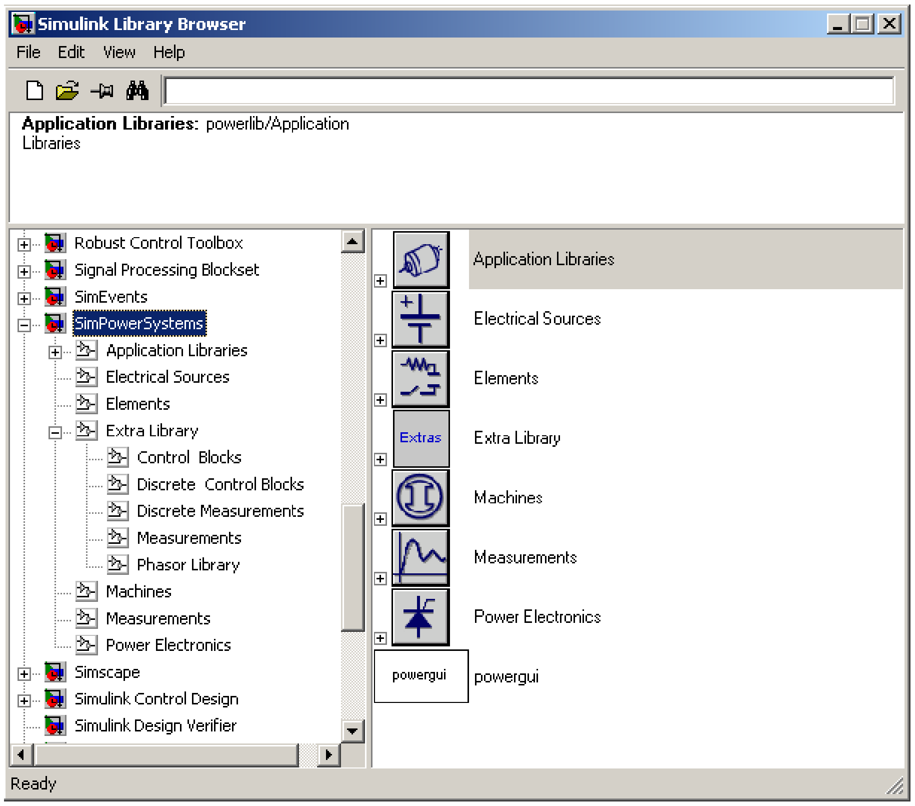


**Figure 2.7**: Library Browser for SimPower System

# CHAPTER 3

# METHODOLOGY

## 3.1 Chapter overview

In this chapter, the procedures of using software MATLAB will be explained. MATLAB R2007b is used for this project. To achieve the objectives of study, all the knowledge are applied on the simulation, program building, and GUI building process into this incorporated software. The workflow for this project was shown with **Figure 3.1**. The flow chart is important to make sure the work is well organized and to keep the project done within the specific time. The flow chart shows that for this project MATLAB program, simulation, and GUI were necessary.

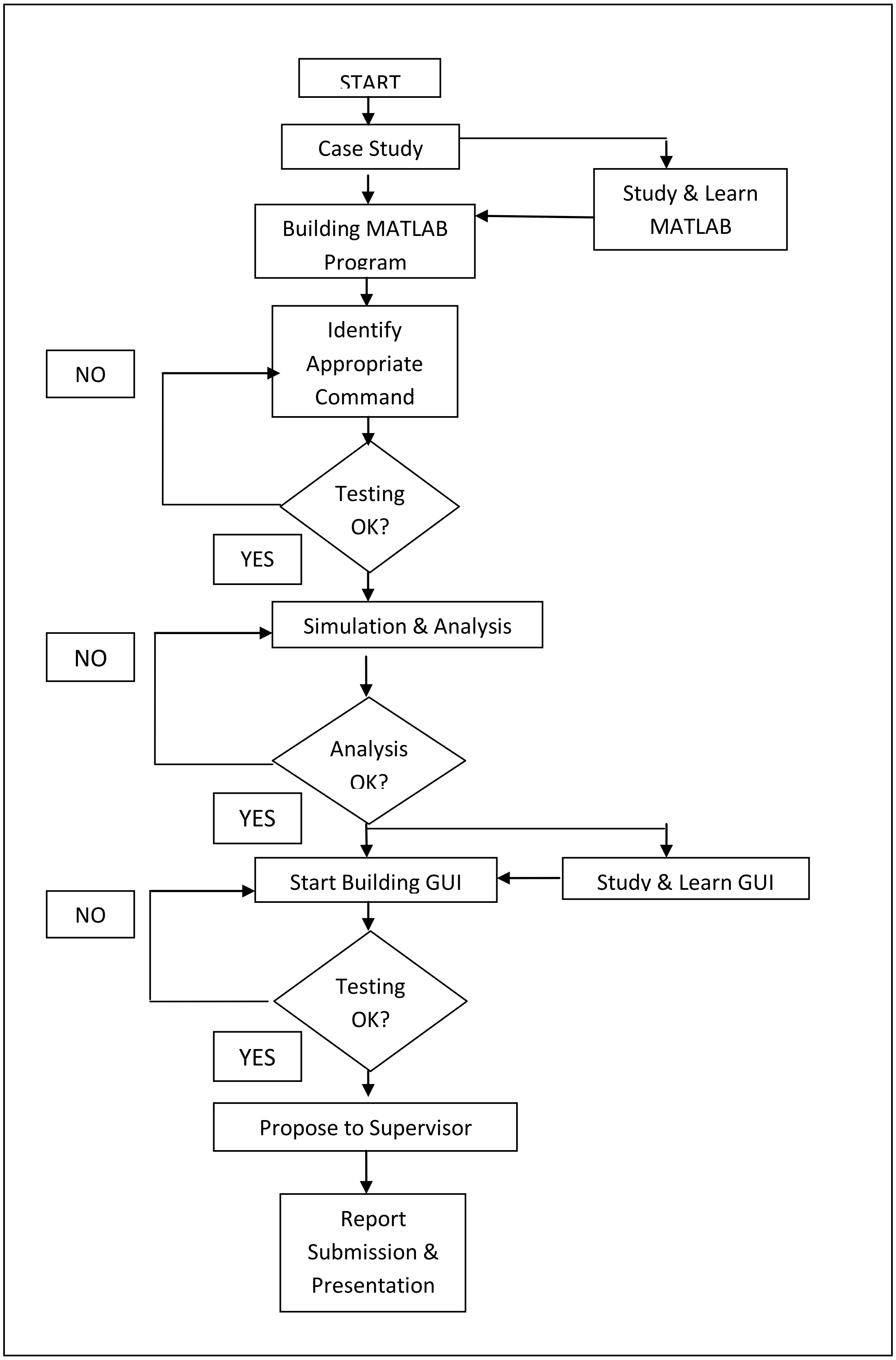


**Figure 3.1**: Work Flow of the Project

## 3.2 Developing the MATLAB program

To write a new program, open a new M-file window. All programs must be written here. Figure below show how to open the M-file window. The program written here must follow the MATLAB rule, especially when using commands and syntaxes, to prevent any error when the program is running. It is important to identify appropriate commands before writing any programs.

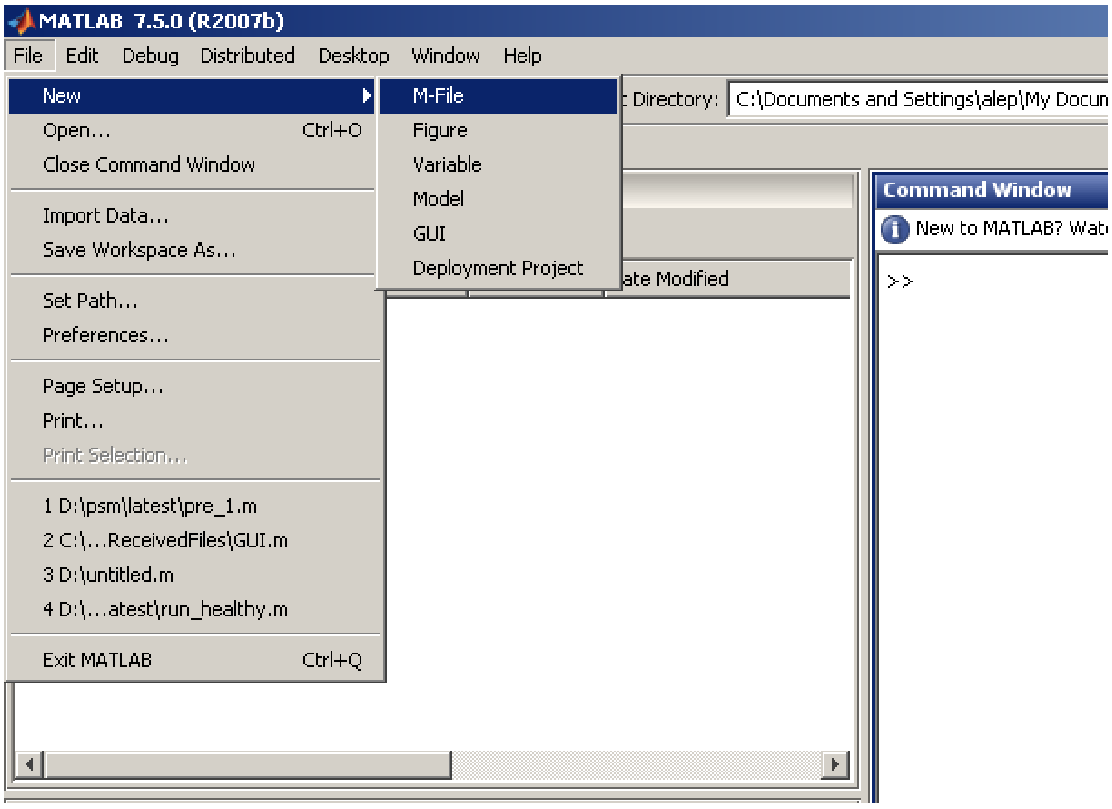


**Figure 3.2**: Opening M-file Window

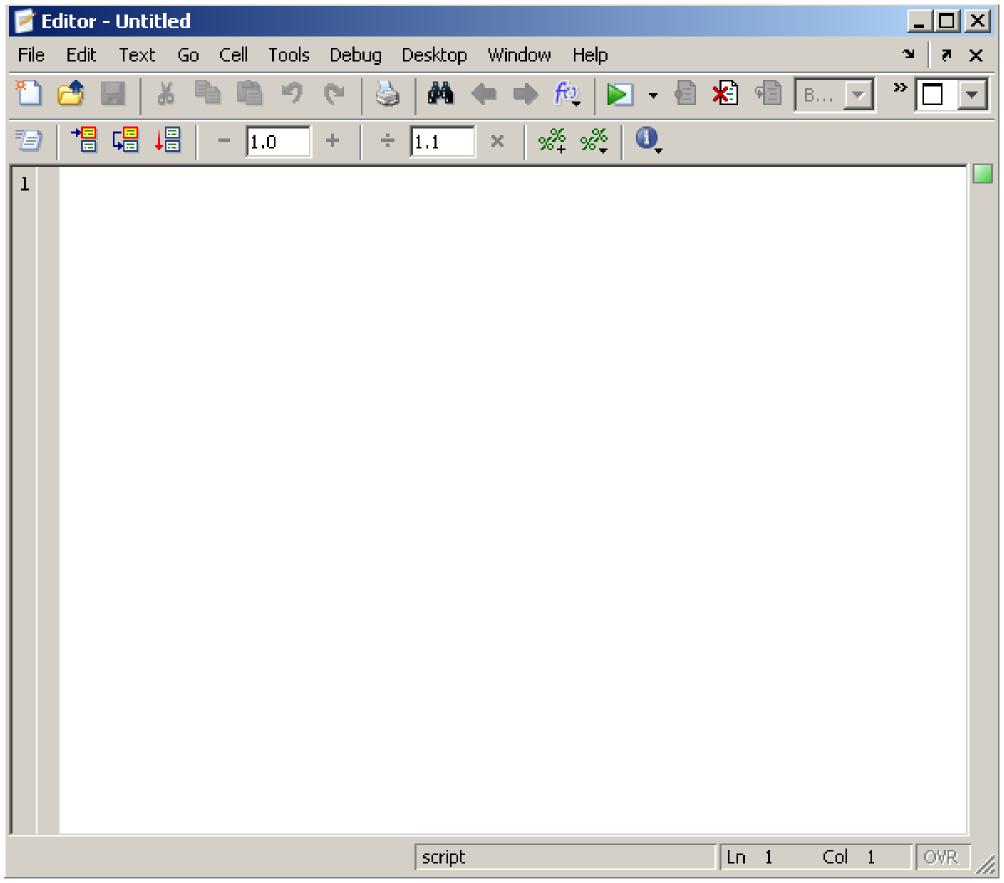
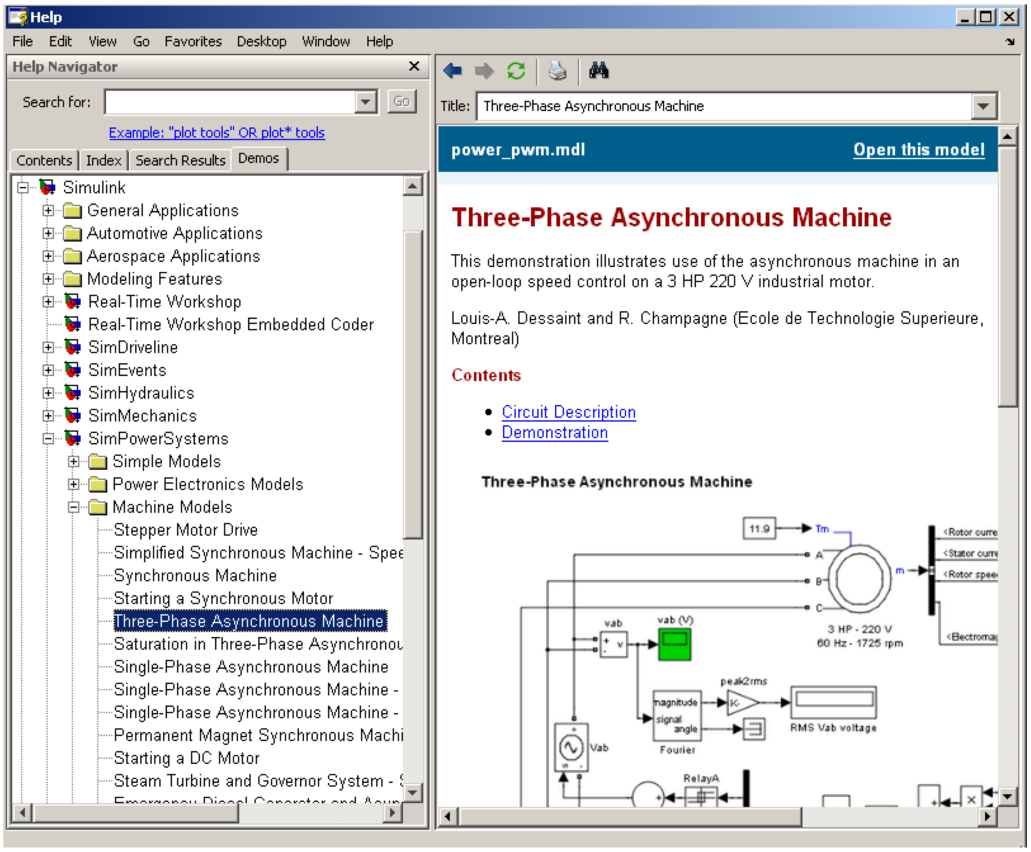


**Figure 3.3**: New M-file window

### 3.3 Developing the simulation model

To construct a simulation program, which is for this project a simulation of induction motor is needed, first we need to open the simulation window. Then the block that we need to use can simply be found at simulink library browser. For the induction motor model, the three phase asynchronous machine is used. For the supply, three phase voltage supply is used, and the parameters for measurement purpose were also added. These are the item that needed to build the initial/healthy state of three phase induction motor for this project.



Help

File Edit View Go Favorites Desktop Window Help

Help Navigator

Search for:

Example: "plot tools" OR plot* tools

Contents | Index | Search Results | Demos

- Simulink
  - General Applications
  - Automotive Applications
  - Aerospace Applications
  - Modeling Features
  - Real-Time Workshop
  - Real-Time Workshop Embedded Coder
  - SimDriveline
  - SimEvents
  - SimHydraulics
  - SimMechanics
  - SimPowerSystems
    - Simple Models
    - Power Electronics Models
    - Machine Models
      - Stepper Motor Drive
      - Simplified Synchronous Machine - Spee
      - Synchronous Machine
      - Starting a Synchronous Motor
      - Three-Phase Asynchronous Machine
      - Saturation in Three-Phase Asynchronou
      - Single-Phase Asynchronous Machine
      - Single-Phase Asynchronous Machine -
      - Single-Phase Asynchronous Machine -
      - Permanent Magnet Synchronous Machi
      - Starting a DC Motor
      - Steam Turbine and Governor System -

Title: Three-Phase Asynchronous Machine

power_pwm.mdl Open this model

**Three-Phase Asynchronous Machine**

This demonstration illustrates use of the asynchronous machine in an open-loop speed control on a 3 HP 220 V industrial motor.

Louis-A. Dessaint and R. Champagne (Ecole de Technologie Superieure, Montreal)

**Contents**

- Circuit Description
- Demonstration

**Three-Phase Asynchronous Machine**

**Figure 3.4:** Three Phase Asynchronous Machine Model

Three phase asynchronous machine model can be obtained from the demos at help window. But the demos were different from what were needed for this project. So a little adjustment need to be done, especially for the voltage supply, measurement parameter, and some assumption were added while doing the model for the faulty state.

The simulation model in the Figure 3.3 above is a three-phase motor rated 3 HP, 220 V, 1725 rpm is fed by a sinusoidal PWM inverter. The base frequency of the sinusoidal reference wave is 60 Hz while the triangular carrier wave's frequency is set to 1980 Hz. The PWM inverter is built entirely with standard Simulink blocks.

As for my project requirement, the voltage supplied must be in three-phase voltage, thus the PWM inverter does not necessary for my simulation.

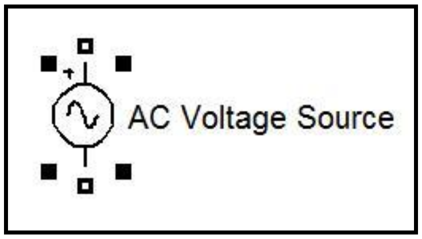
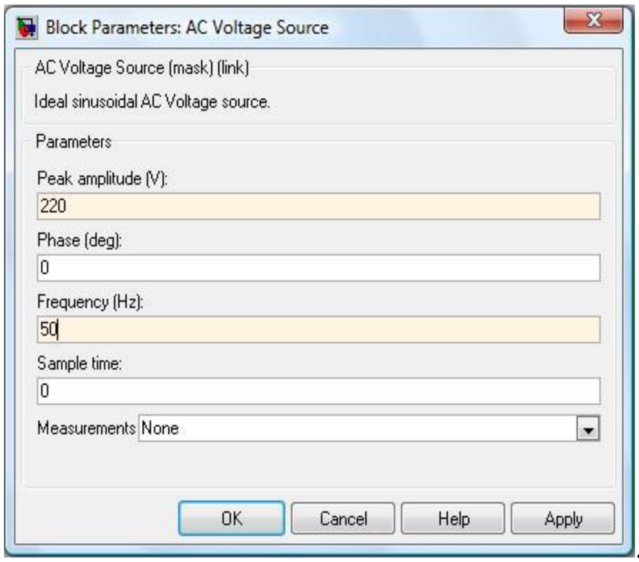


**Figure 3.5**: Three Phase Induction Motor Model

The model was modified to a three-phase supply by eliminate the PWM inverter and replaced it with the AC voltage source from the Library Browser as shown in Figure 3.5


Block Parameters: AC Voltage Source

AC Voltage Source (mask) (link)

Ideal sinusoidal AC Voltage source.

Parameters

Peak amplitude (V): 220

Phase (deg): 0

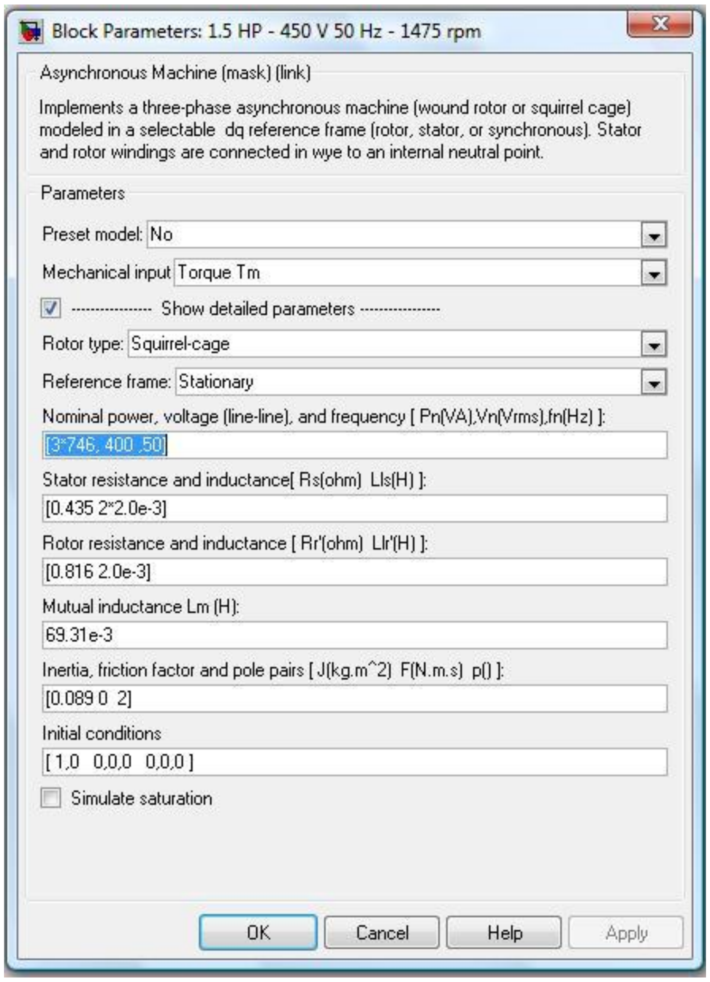
Frequency (Hz): 50

Sample time: 0

Measurements None

.

**Figure 3.6**: Block Parameters of AC Voltage Source



Block Parameters: 1.5 HP - 450 V 50 Hz - 1475 rpm

Asynchronous Machine (mask) (link)

Implements a three-phase asynchronous machine (wound rotor or squirrel cage) modeled in a selectable dq reference frame (rotor, stator, or synchronous). Stator and rotor windings are connected in wye to an internal neutral point.

Parameters

Preset model: No

Mechanical input Torque Tm

☑ --------------- Show detailed parameters ---------------

Rotor type: Squirrel-cage

Reference frame: Stationary

Nominal power, voltage (line-line), and frequency [ Pn(VA),Vn(Vrms),fn(Hz) ]:
[3*746, 400 ,50]

Stator resistance and inductance[ Rs(ohm) Lls(H) ]:
[0.435 2*2.0e-3]

Rotor resistance and inductance [ Rr'(ohm) Llr'(H) ]:
[0.816 2.0e-3]

Mutual inductance Lm (H):
69.31e-3

Inertia, friction factor and pole pairs [ J(kg.m^2) F(N.m.s) p() ]:
[0.089 0 2]

Initial conditions
[ 1,0 0,0,0 0,0,0 ]

☐ Simulate saturation

**Figure 3.7**: Block Parameters of 1.5HP – 450V 50Hz – 1475rpm

Since the motor is set up to PWM inverter supply, the parameters of the motor should be change regarding the supply and the frequency is different. It is done by double click at the motor block to obtain its Block Parameters window as shown in Figure 3.6. The voltage is changed to 400V as for three-phase and the frequency is set up to 50Hz. The motor type now is 1.5HP, 450V 50Hz supply that could generate approximately 1475rpm.

**Table 3.1:** Description of the motor Block Parameters

| | |
|---|---|
| Preset model | Provides a set of predetermined electrical and mechanical parameters for various asynchronous machine ratings of power (HP), phase-to-phase voltage (V), frequency (Hz), and rated speed (rpm). |
| Mechanical input | Allows the selection of either the torque applied to the shaft or the rotor speed as the Simulink signal applied to the block's input. |
| Rotor type | Specifies the branching for the rotor windings. |
| Reference frame | Specifies the reference frame that is used to convert input voltages (abc reference frame) to the dq reference frame, and output currents (dq reference frame) to the abc reference frame |
| Nominal power, L-L volt, and freq. | The nominal apparent power Pn (VA), RMS line-to-line voltage Vn (V), and frequency fn (Hz). |
| Stator | The stator resistance Rs (Ω or pu) and leakage inductance Lls (H or pu). |
| Rotor | The rotor resistance Rr' (Ω or pu) and leakage inductance Llr' (H or pu), both referred to the stator. |
| Mutual inductance | The magnetizing inductance Lm (H or pu). |
| Inertia, friction factor, and pairs of poles | For the **SI units** dialog box: the combined machine and load inertia coefficient J (kg.m$^2$), combined viscous friction coefficient F (N.m.s), and pole pairs p. The friction torque Tf is proportional to the rotor speed ω (Tf = F.w).<br>For the **pu units** dialog box: the inertia constant H (s), combined viscous friction coefficient F (pu), and pole pairs p. |
| Initial conditions | Specifies the initial slip s, electrical angle Θe (degrees), stator current magnitude (A or pu), and phase angles (degrees) |

| | |
|---|---|
| Simulate saturation | Specifies whether magnetic saturation of rotor and stator iron is simulated or not. |
| Saturation parameters | Specifies the no-load saturation curve parameters. |

**Table 3.1** shows the description of every parameter that appears in the Block Parameters of the motor. User can characterized desirably the motor specification depends on their purpose of usage.

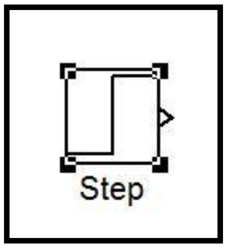


**Figure 3.8**: Step Block

**Figure 3.7** shows the Step block, represents the external mechanical torque which is driven by the motor, considered as load applied. From the step block parameter, the final value represents the load condition. In this project, the load condition is range from 0 to 1, which are in per unit value. 0 shows the motor is in no load. 0.5 shows the motor is in half load. 1.0 shows the motor is in full load. In this project, only full load is being used.

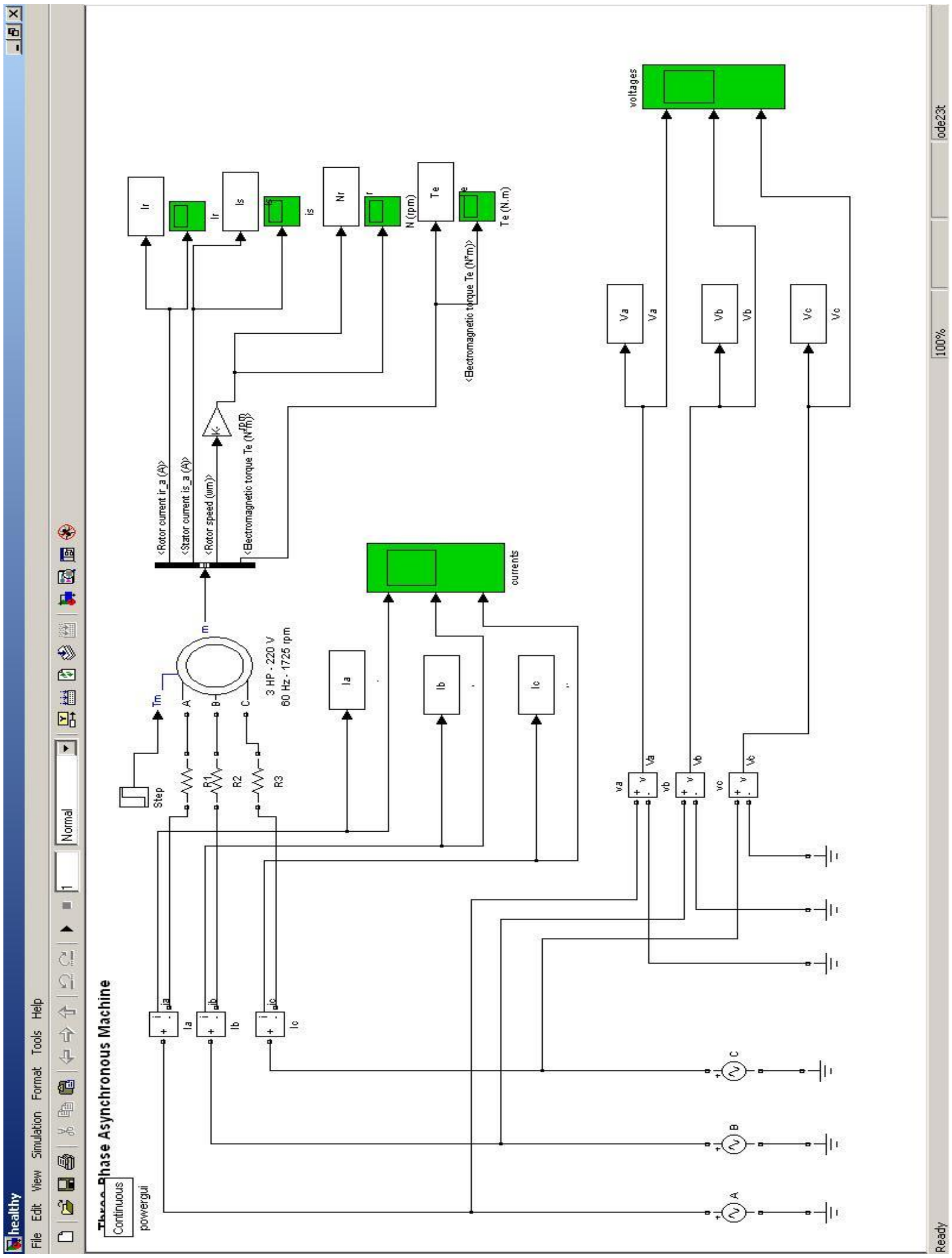


**Figure 3.9**: Three Phase Induction Motor model

### 3.3.1 Stator inter-turn short circuit fault model

For stator inter-turn short circuit model, assumption that was made is for initial/ healthy state, 3 resistors were added right before the motor for each phase. To create the stator inter-turn short circuit state, one of the resistor will be short circuited, and act like the insulation failure in between the winding inter-turns get short circuited for actual motor.

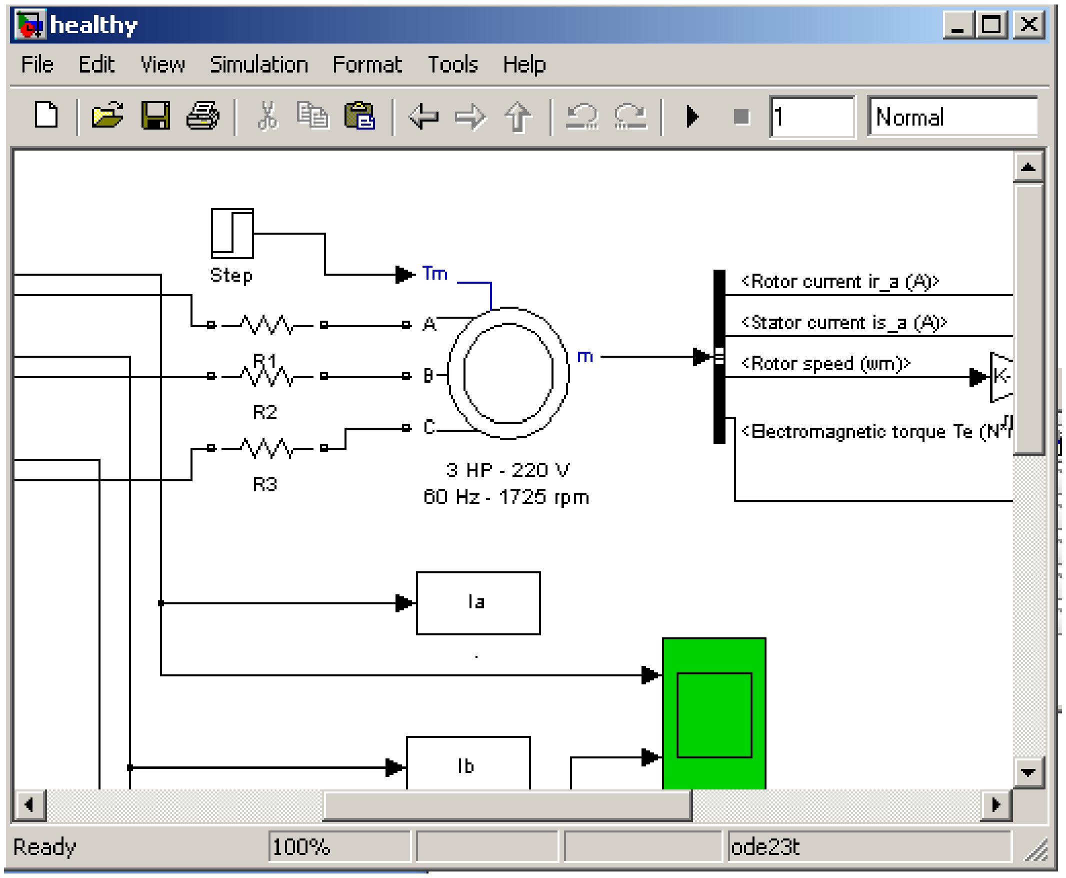


**Figure 3.10**: Three Resistors before the Motor (healthy state)

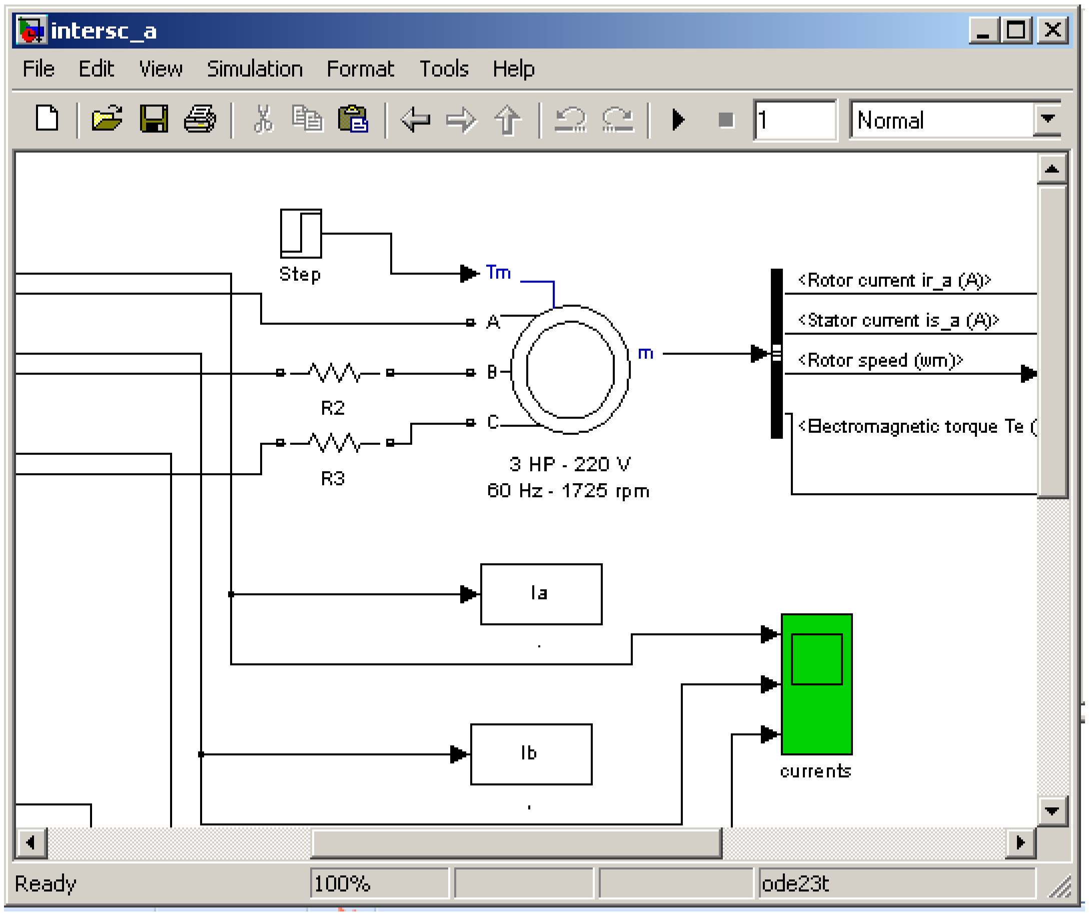
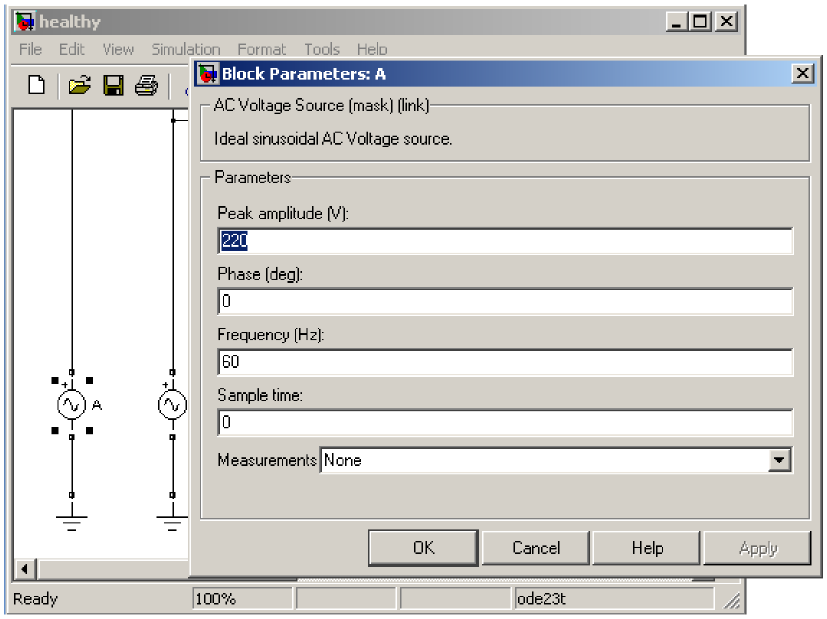


**Figure 3.11**: One of the Resistor were Short Circuited

### 3.3.2 Unbalanced voltage supply fault model

For unbalanced voltage supply model, the model is exactly same for the initial/healthy state model. What make it differ is one of the voltage supply is lowered a bit to create the unbalanced voltage supply faults. The figure below will show how to set the voltages.



Block Parameters: A

AC Voltage Source (mask) (link)

Ideal sinusoidal AC Voltage source.

Parameters

Peak amplitude (V): 220

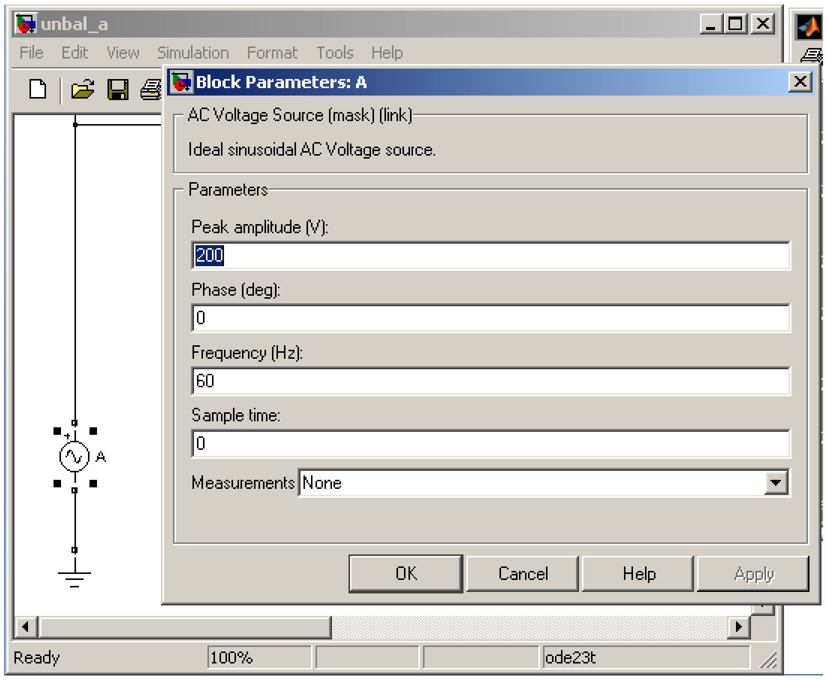
Phase (deg): 0

Frequency (Hz): 60

Sample time: 0

Measurements: None

**Figure 3.12:** Voltage Setting for Healthy State Motor



Block Parameters: A

AC Voltage Source (mask) (link)

Ideal sinusoidal AC Voltage source.

Parameters

Peak amplitude (V): 200

Phase (deg): 0

Frequency (Hz): 60

Sample time: 0

Measurements: None

**Figure 3.13:** Voltage Setting for Faulty State Motor

### 3.3.3 Single-phase open circuit fault model

For single-phase open circuit model, the model is slightly same like the initial/healthy state motor model. The only thing that makes it different is a circuit breaker is added right after the voltage supply. When the model is run, the breaker will operate and make one phase is open circuit. Figure below will explain the process.

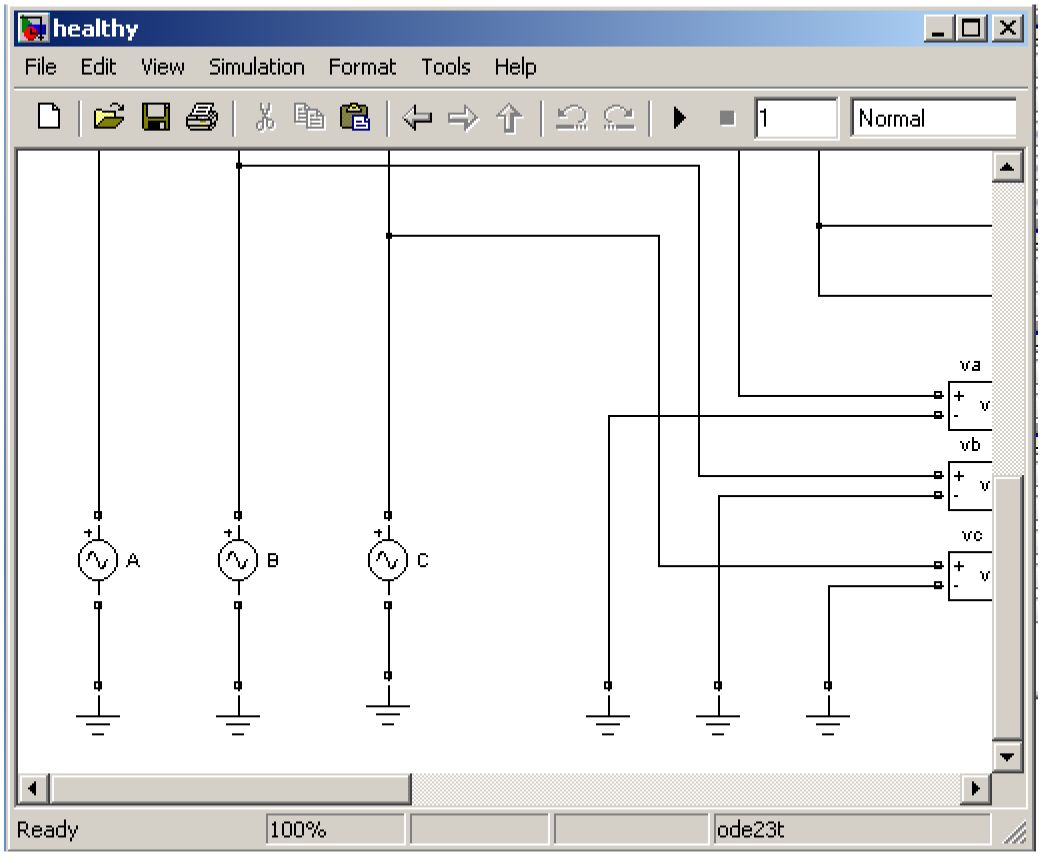


**Figure 3.14**: Voltage Supply for Healthy Model

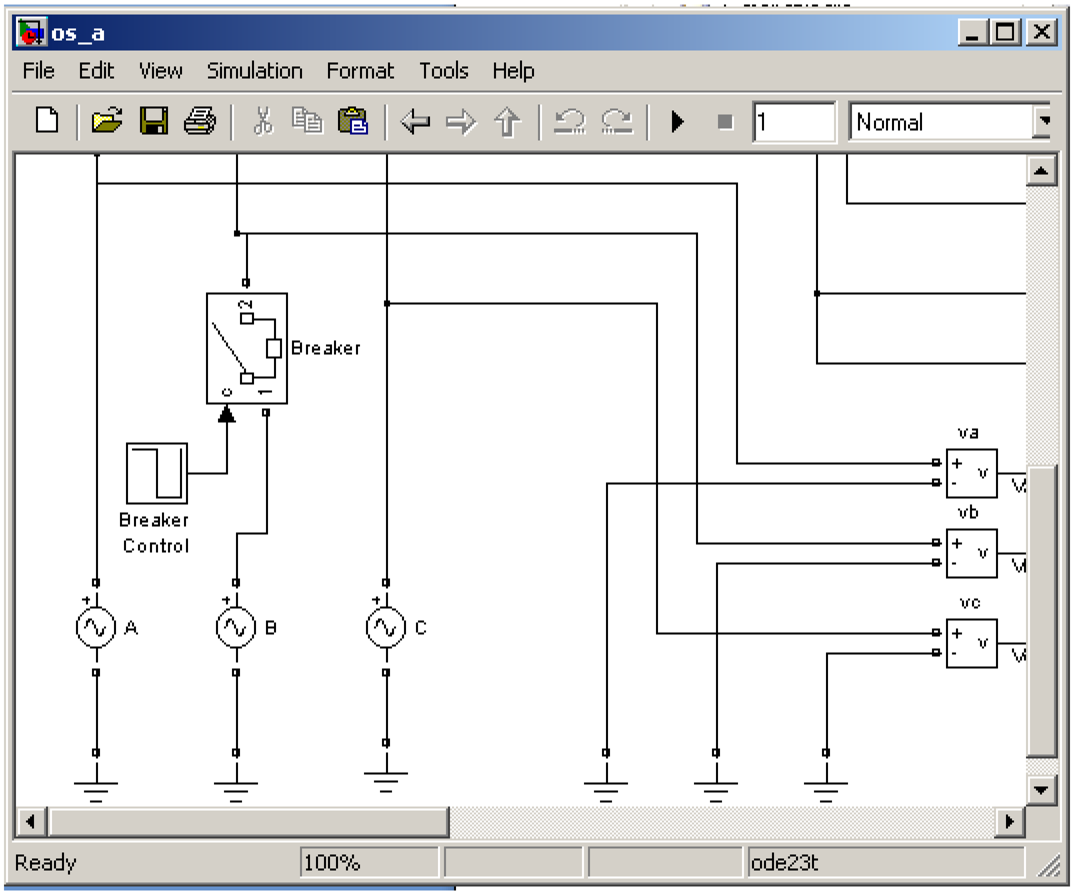
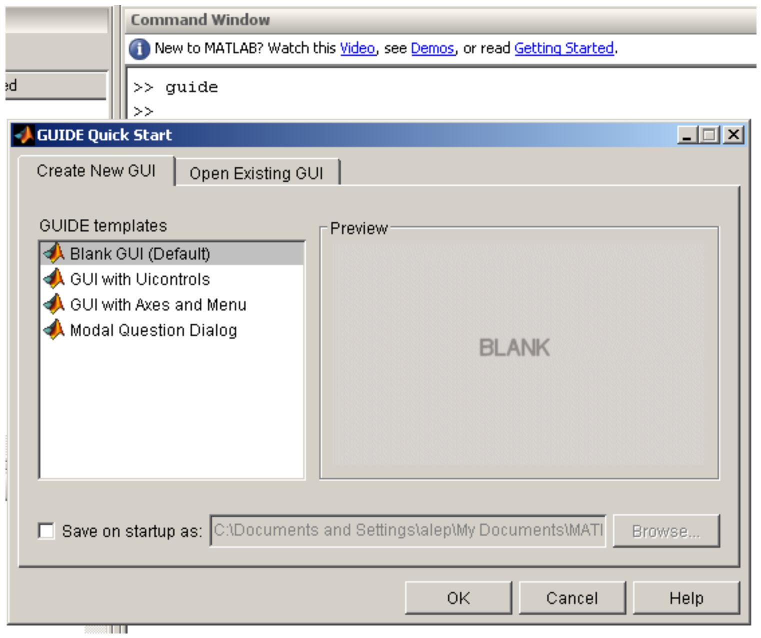


**Figure 3.15**: Voltage Supply for Faulty State

## 3.4 Developing the GUI

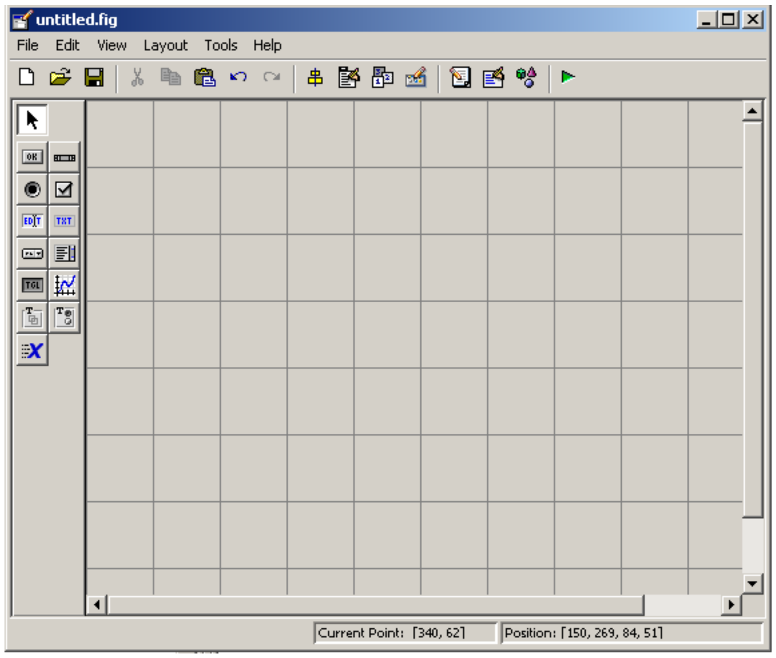
To create a new GUI, simply write down “guide” at the command window, or by choosing to open new GUI window as shown at figure below. Choose blank GUI to create a new one, then the GUI made is depend on our creativity.



```
>> guide
>>
```

C:\Documents and Settings\alep\My Documents\MATI

**Figure 3.16:** GUI Window



Current Point: [340, 62]
Position: [150, 269, 84, 51]

**Figure 3.17**: Blank GUI Window

There several menu button that can help user to customize their GUI. Each one has different usage:

- for push button
- for scroll bar
- for radio button
- for check box
- for edit text
- for textbox
- for popup menu
- for list box
- for toggle button
- for axes
- For panel button
- for group button
- for active-X

After creating the desired GUI, simply press the button at the top of the GUI window. This will generate the program at M-file window. At this point, it is crucial to define the program that you want to use at the GUI you created before.

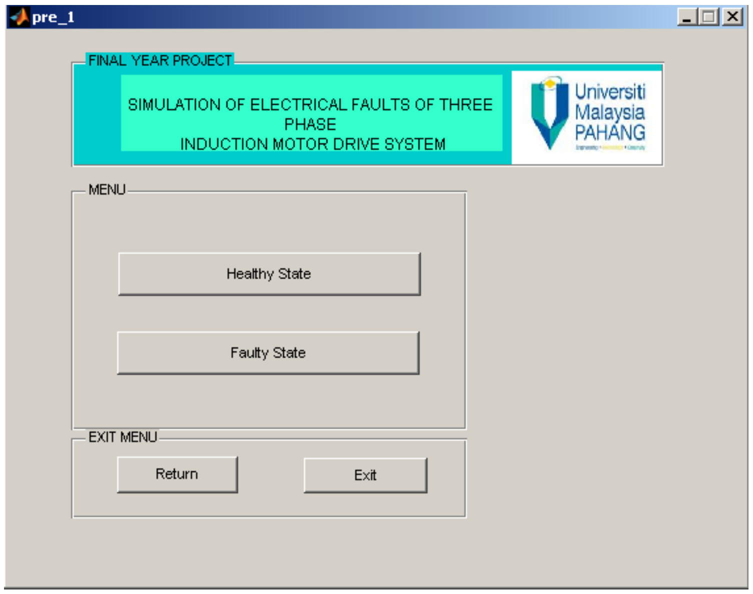
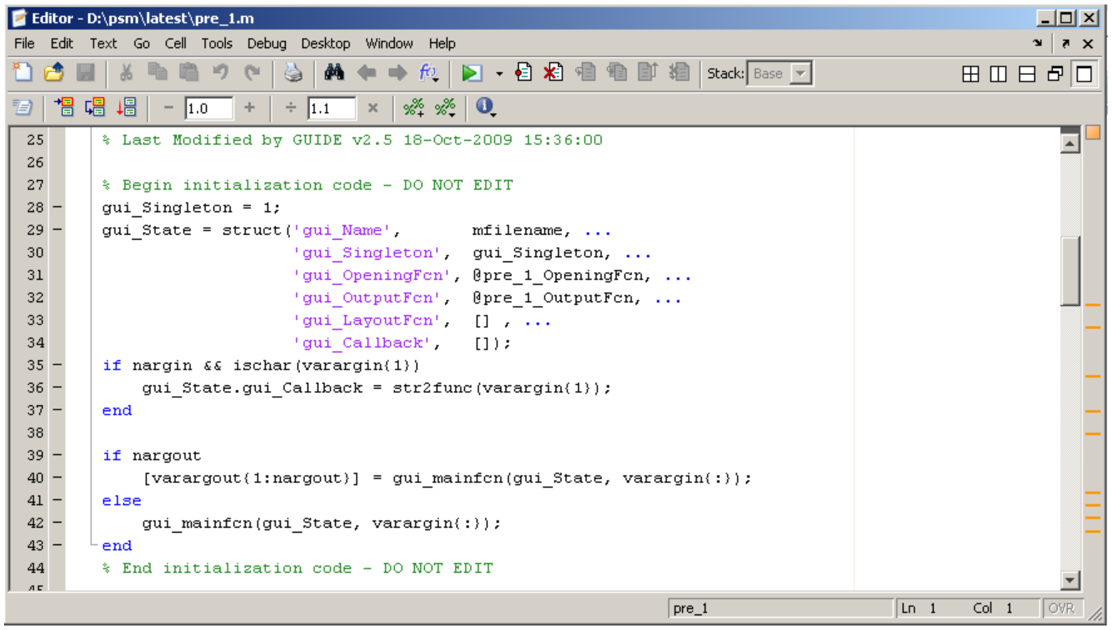

pre_1
FINAL YEAR PROJECT
SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM
Universiti Malaysia PAHANG
MENU
Healthy State
Faulty State
EXIT MENU
Return
Exit

**Figure 3.18**: Example of Created GUI

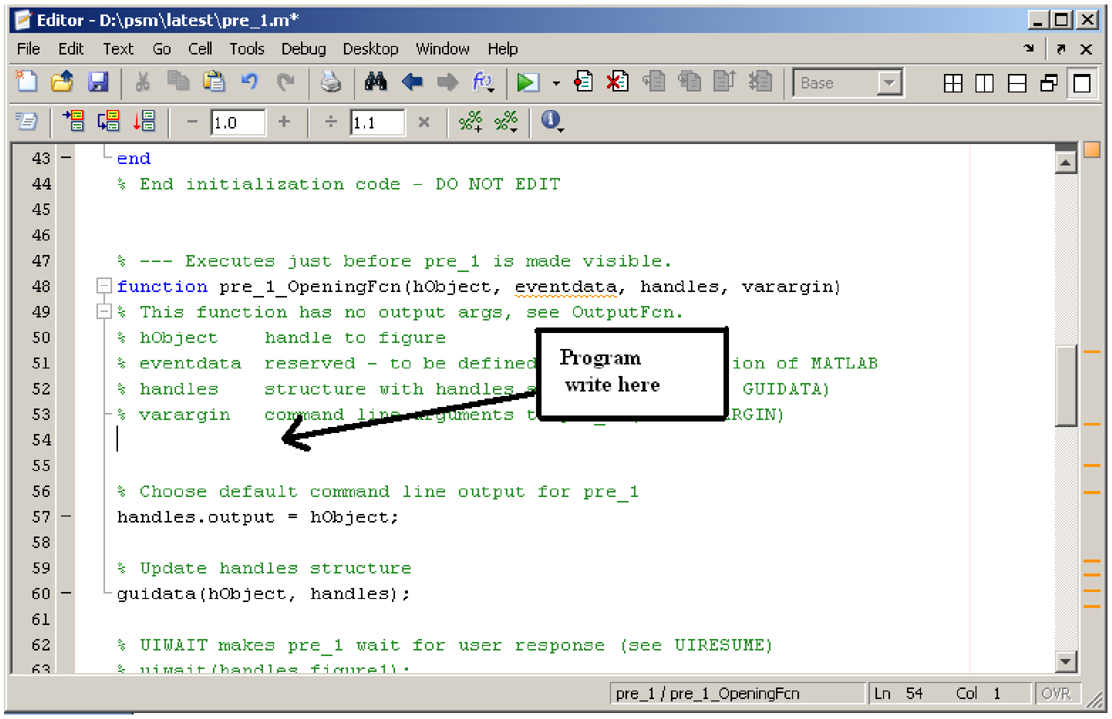

Editor - D:\psm\latest\pre_1.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
pre_1
Ln 1 Col 1 OVR

```
% Last Modified by GUIDE v2.5 18-Oct-2009 15:36:00

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @pre_1_OpeningFcn, ...
                   'gui_OutputFcn',  @pre_1_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
```

**Figure 3.19**: Example of the Generated Program

### 3.4.1 Main window M-FILE description

All programs must be written at M-file after running the FIG-file, after the function line.


Editor - D:\psm\latest\pre_1.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
Base
1.0
1.1
Program write here
pre_1 / pre_1_OpeningFcn
Ln 54
Col 1
OVR

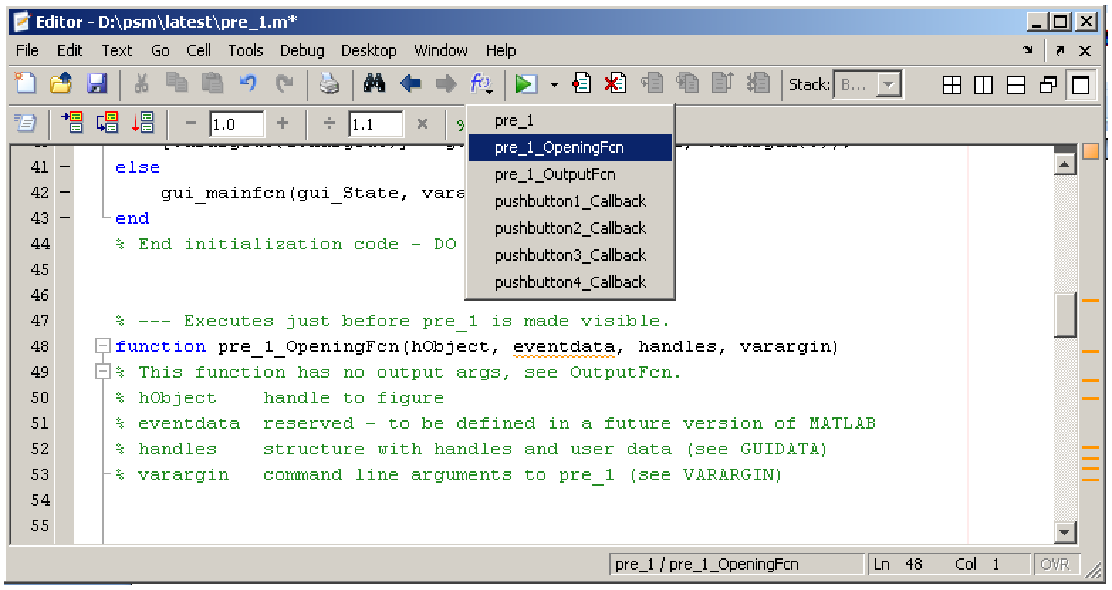
```
end
% End initialization code - DO NOT EDIT


% --- Executes just before pre_1 is made visible.
function pre_1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined               ion of MATLAB
% handles    structure with handles                  GUIDATA)
% varargin   command line arguments t               RGIN)


% Choose default command line output for pre_1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes pre_1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

**Figure 3.20:** Display the Program for Opening Function

#### 3.4.1.1 Main window M-FILE

See appendix A for Main Window Programming.

### 3.4.1.2 Description



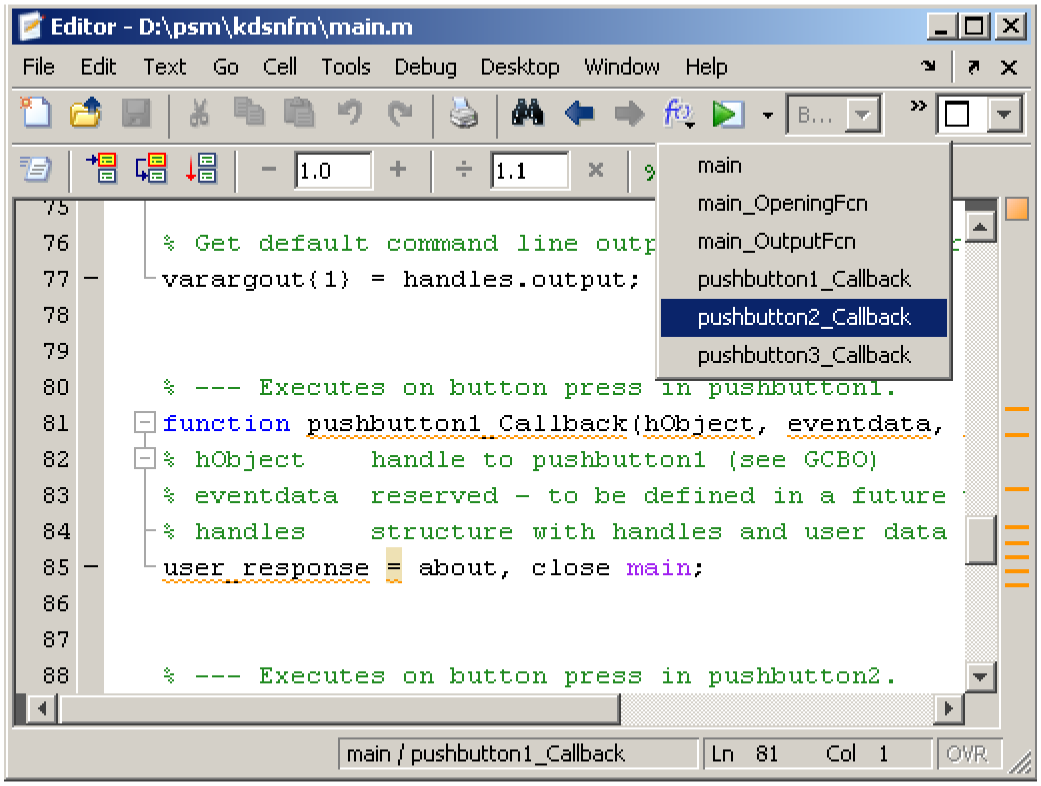
```
else
    gui_mainfcn(gui_State, vara
end
% End initialization code - DO


% --- Executes just before pre_1 is made visible.
function pre_1_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 pre_1 (see VARARGIN)
```

**Figure 3.21:** Display the Opening Function Position

**Opening Function Syntax:**

```
movegui ('center')
```

This syntax demonstrates the usefulness of movegui to ensure that saved GUIs appear on screen when reloaded, regardless of the target computer's screen sizes and resolution.



```
    % Get default command line outp
    varargout{1} = handles.output;


    % --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
    % hObject    handle to pushbutton1 (see GCBO)
    % eventdata  reserved - to be defined in a future
    % handles    structure with handles and user data
    user_response = about, close main;


    % --- Executes on button press in pushbutton2.
```

**Figure 3.22:** Display the Callback Function Line Position

**Callback Function Syntax:**

A callback is a function that executes when user perform a specific action such as clicking a push button or pressing a keyboard key, or when a component is created or deleted. Each component and menu item has properties that specify its callback. When you create a GUI, you must program the callbacks you need to control operation of the GUI.

When a user activates a component of the GUI, the GUI executes the corresponding callback. The name of the callback is determined by the component's

Tag property and the type of the callback. For example, a push button with the Tag print_button executes the callback.

**Syntax:**

```
user_response = run_healthy;
```

**Discription:**

This is the command to call a GUI named as run_healthy, where the GUI will then will call the simulation of the initial/healthy state motor model.

`user_response` command interact with user responds, whether by clicking or ticking, to execute the desired program. It is a very useful command because almost all the pushbutton callback is using this command to execute. We can relate or connect it with other GUI, simulation, and program.

# CHAPTER 4

# RESULT AND ANALYSIS

## 4.1 Chapter overview

In this chapter, the result obtained from the simulations for the healthy and faulty state motor will be discussed. All the results are in graph form, which is from the measurement parameter which is created during methodology phase.

## 4.2 Simulation of Healthy State Motor

With the help from GUI, this simulation file can be executed by only clicking the push button. **Figure 4.1** will explain the process.

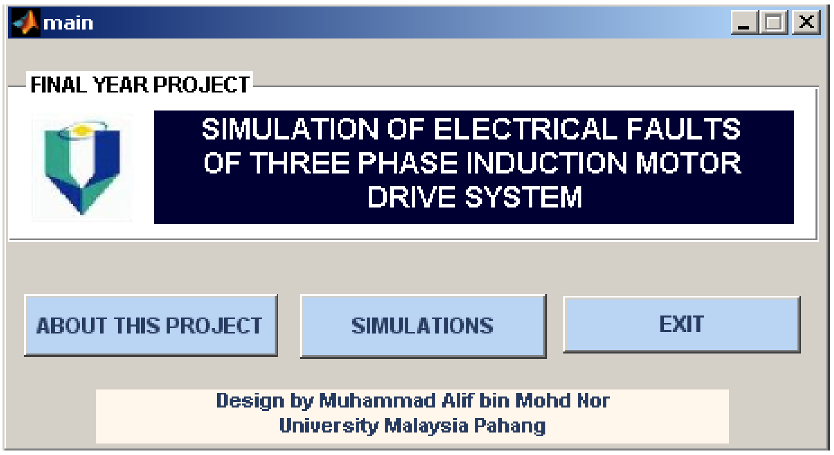
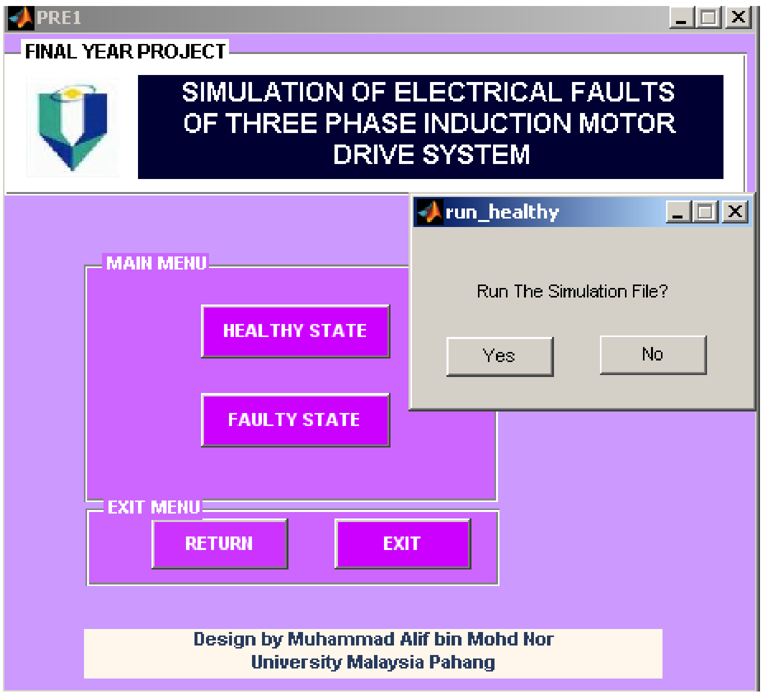

main
FINAL YEAR PROJECT
SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM
ABOUT THIS PROJECT
SIMULATIONS
EXIT
Design by Muhammad Alif bin Mohd Nor
University Malaysia Pahang

**Figure 4.1**: Main GUI Window


PRE1
FINAL YEAR PROJECT
SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM
MAIN MENU
HEALTHY STATE
FAULTY STATE
EXIT MENU
RETURN
EXIT
run_healthy
Yes
No
Design by Muhammad Alif bin Mohd Nor
University Malaysia Pahang

Run The Simulation File?

**Figure 4.2:** Display the Confirm Action Window

**Figure 4.2** shows the confirm action window will show up after pressing the ‘Healthy State’ push button. Pressing ‘Yes’ pushbutton at confirm action window will open the simulation file for healthy state.

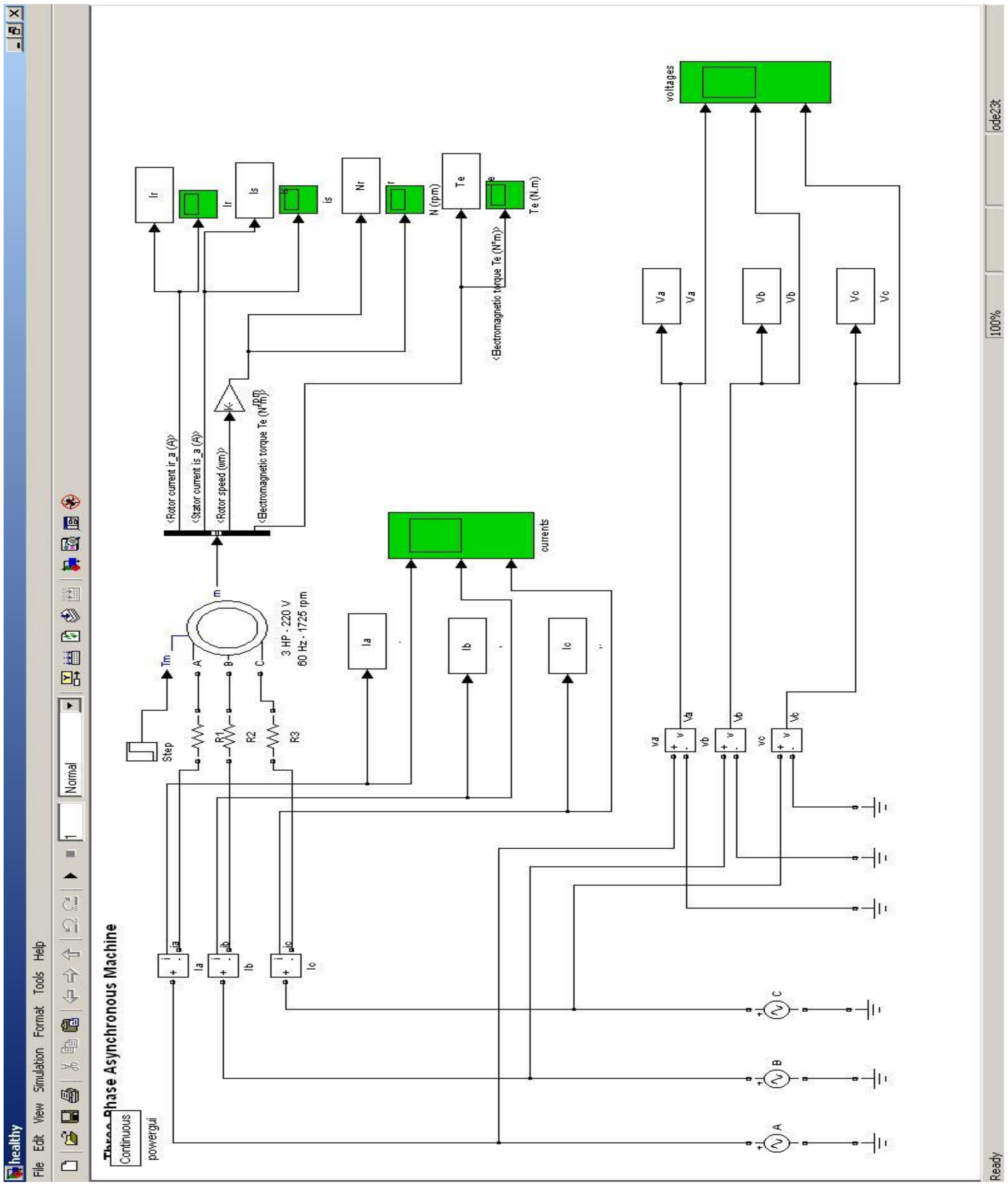


**Figure 4.3:** Display the Healthy State Simulation Window

By pressing ‘RUN’ button, the simulation will run. The result can be obtained from the ‘SCOPE’ block. The results for this simulation are as follow:

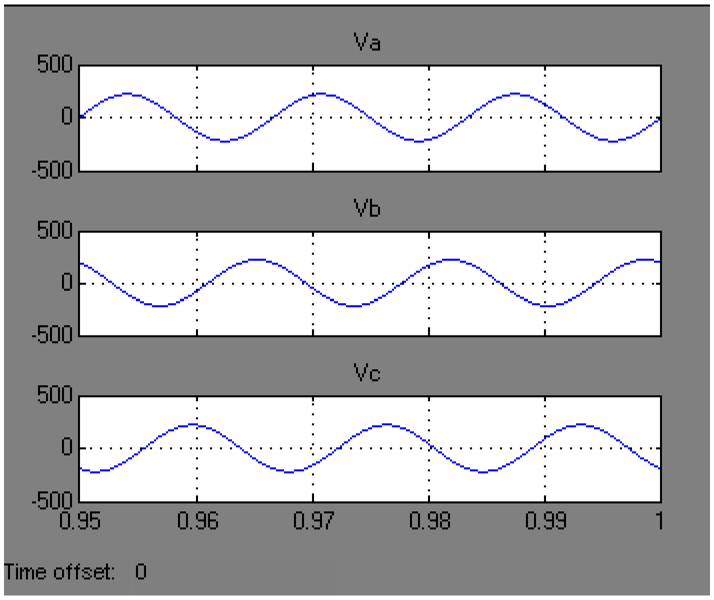


**Figure 4.4:** The three Phase Voltages

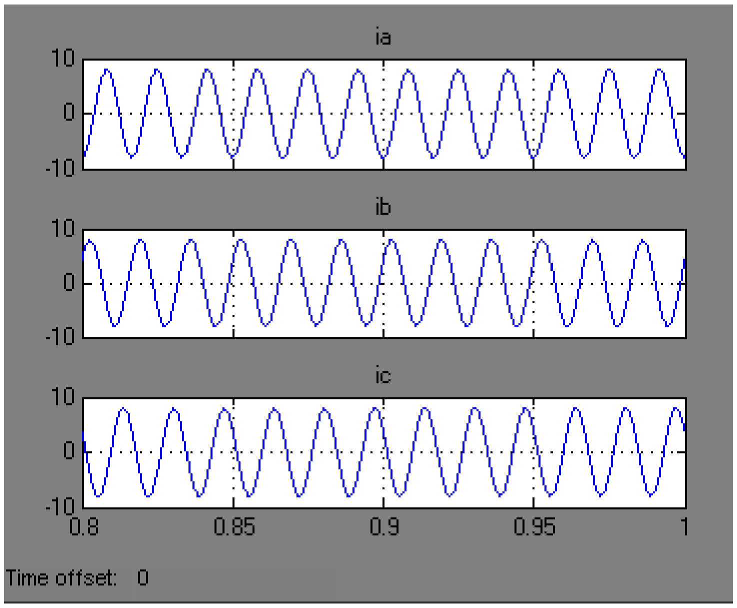


**Figure 4.5:** The Three Phase Currents

**Figure 4.4** and **figure 4.5** show the respected voltages and currents.

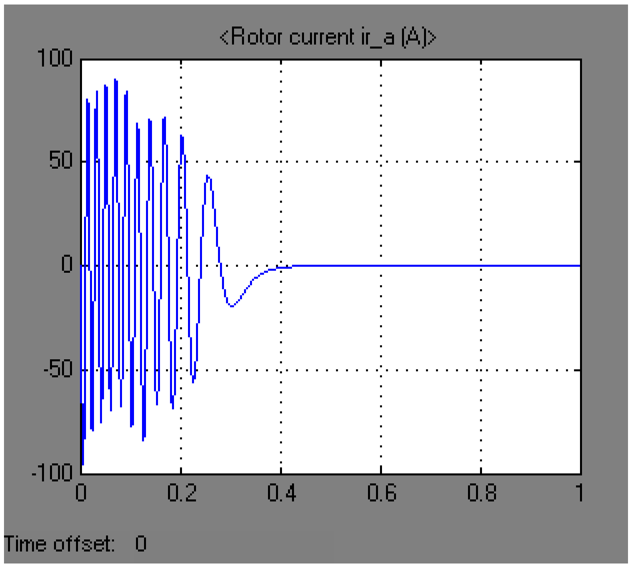


**Figure 4.6:** Rotor Current

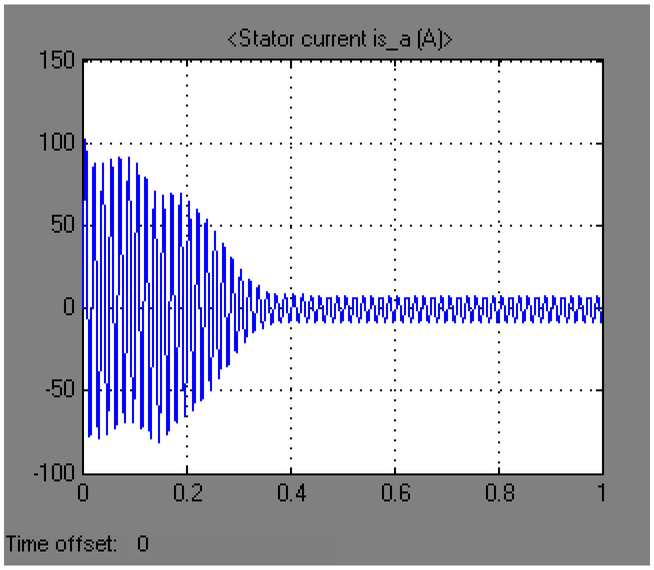


**Figure 4.7:** Stator Current

**Figure 4.6** and **figure 4.7** show the rotor and stator current.

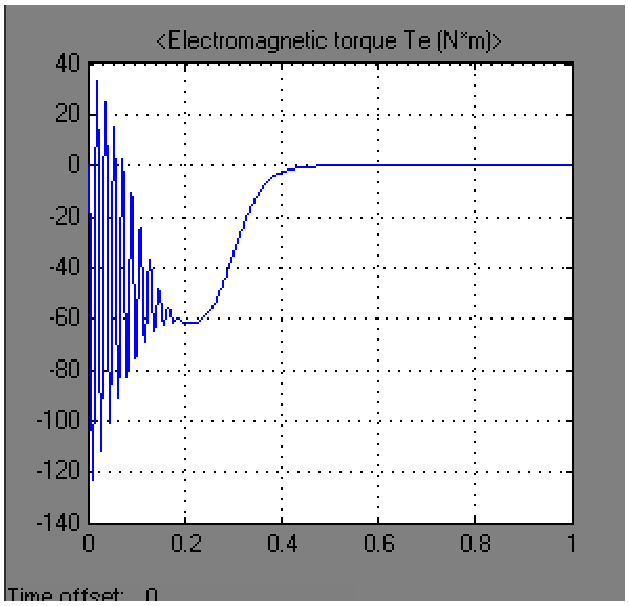


**Figure 4.8:** Electromagnetic Torque

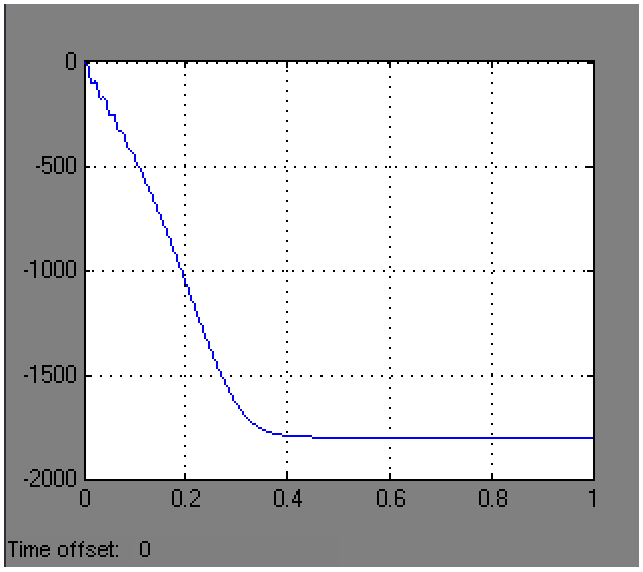


**Figure 4.9:** Motor Speed

**Figure 4.8** and **figure 4.9** shows the Electromagnetic torque and motor speed.

## 4.3 Simulation of Faulty State Motor

By returning to the previous GUI, there is another option can be selected, the Faulty State. By pressing the button, another GUI will showed up, giving another option to choose, Inter-turn Short Circuit, Single Phase Open Circuit, and Unbalanced Voltage supply. Each option will open the different simulation, according to it owns cases.

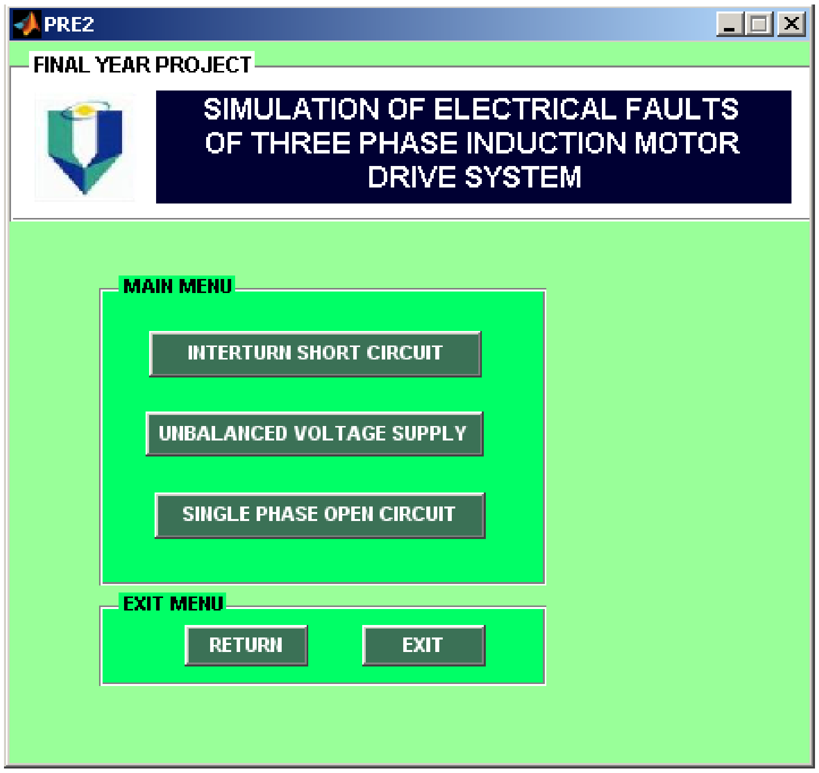

PRE2
FINAL YEAR PROJECT
SIMULATION OF ELECTRICAL FAULTS OF THREE PHASE INDUCTION MOTOR DRIVE SYSTEM
MAIN MENU
INTERTURN SHORT CIRCUIT
UNBALANCED VOLTAGE SUPPLY
SINGLE PHASE OPEN CIRCUIT
EXIT MENU
RETURN
EXIT

**Figure 4.10:** GUI for Faulty State

By pressing ‘Interturn Short Circuit’ button, another confirms action window will open. Pressing ‘Yes’ will open the simulation.

## 4.3.1 Simulation for Inter-turn Short Circuit

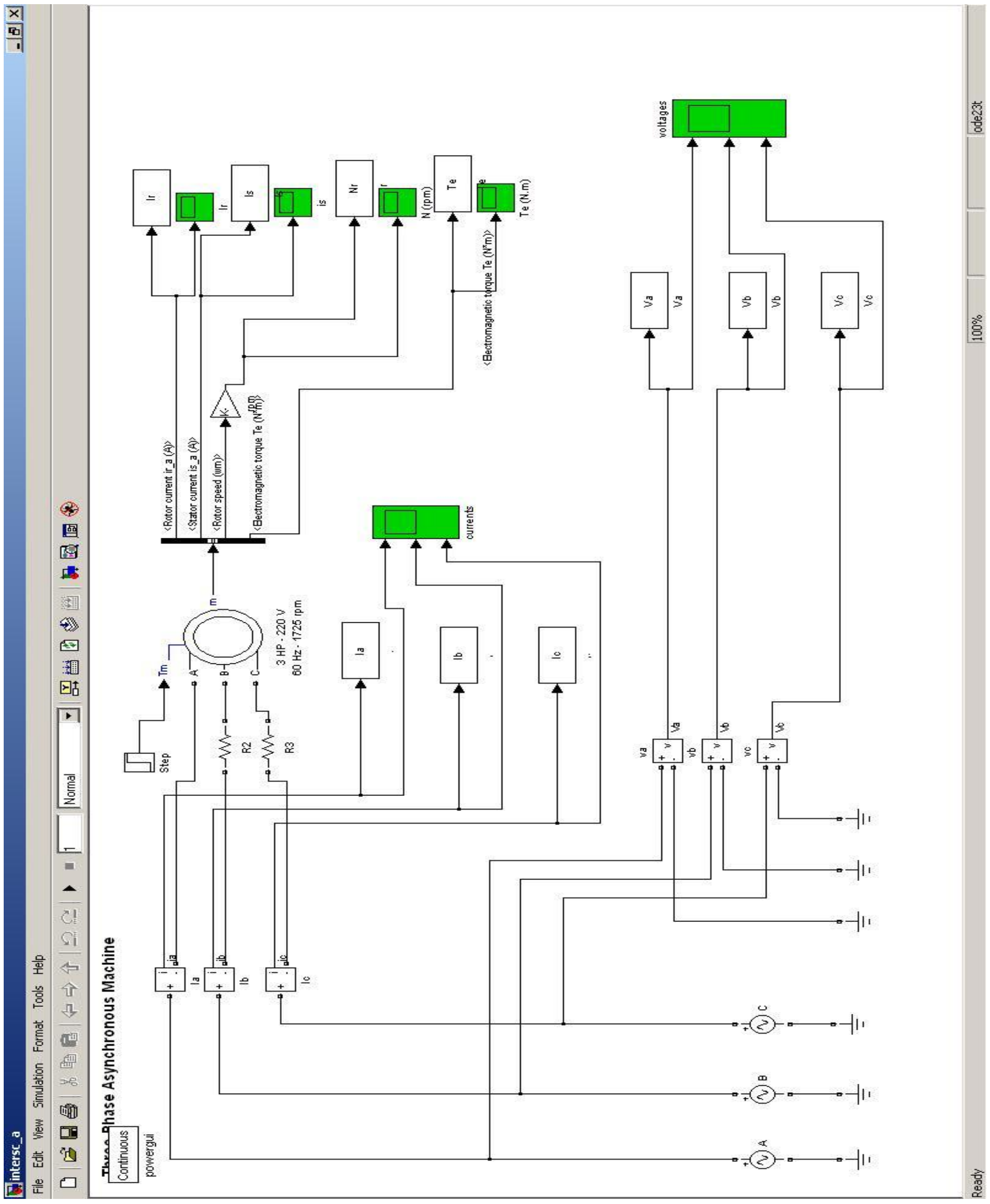


**Figure 4.11:** Simulation for Inter-turn Short Circuit

Run the simulation, the result can be obtained from the scope block.

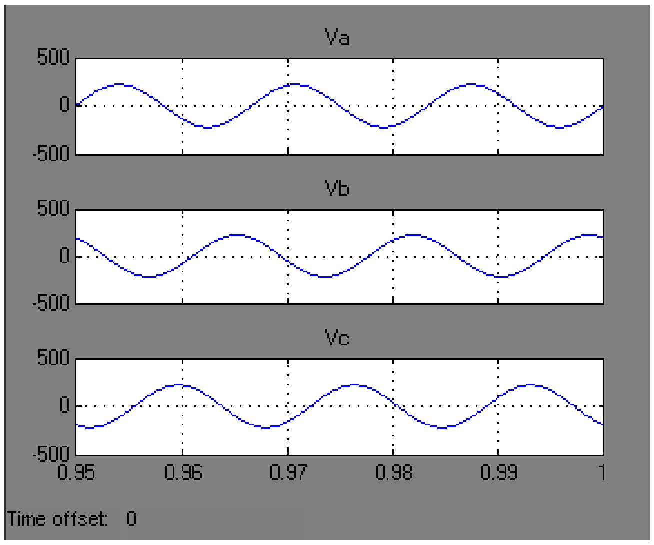


**Figure 4.12:** Three Phase Voltages

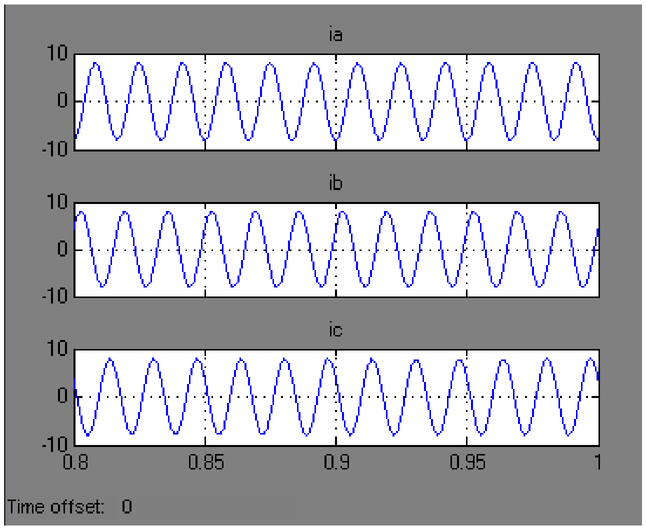


**Figure 4.13:** Three Phase Currents

**Figure 4.12** and **figure 4.13** show the respected voltages and currents.

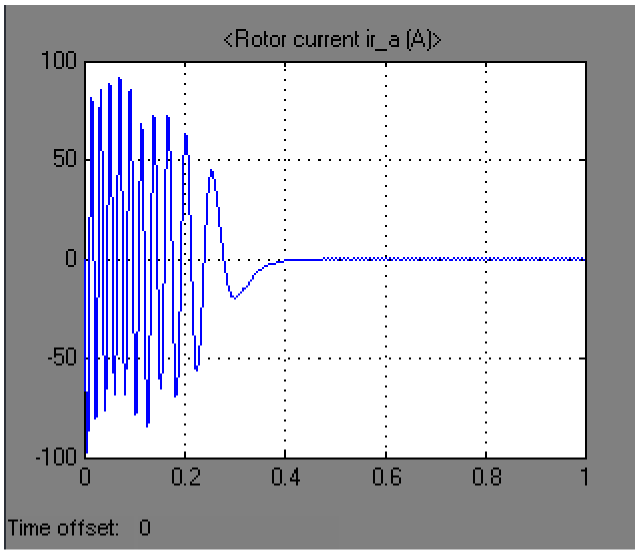


**Figure 4.14:** Rotor Current

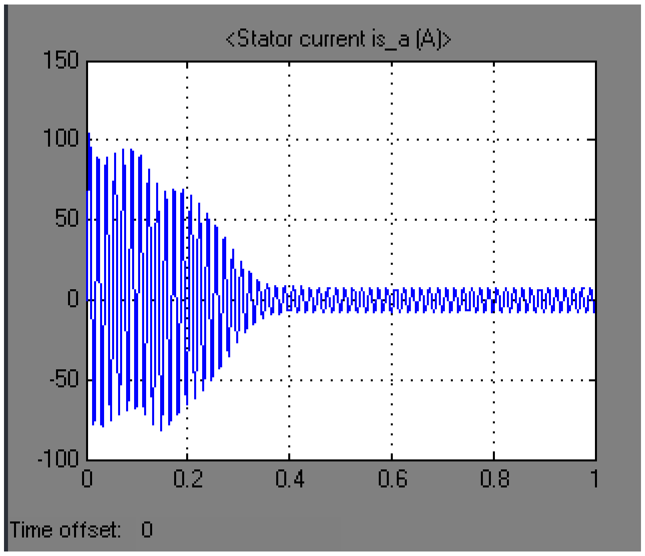


**Figure 4.15:** Stator Current

**Figure 4.14** and **figure 4.15** show the rotor and stator current.

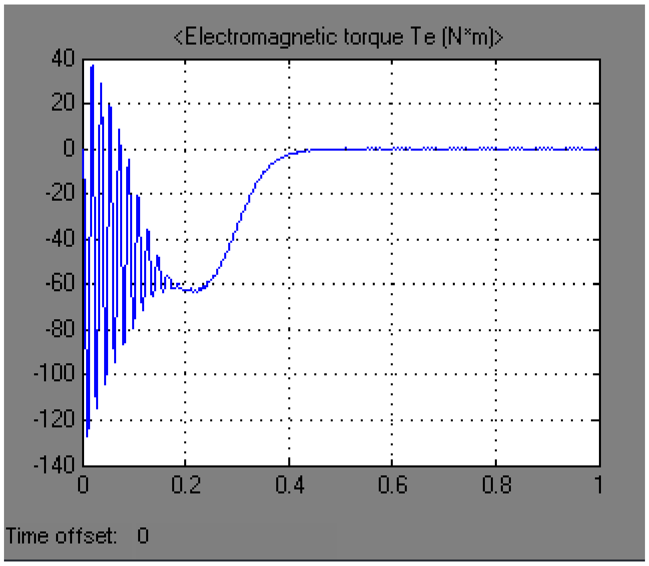


**Figure 4.16:** Electromagnetic Torque

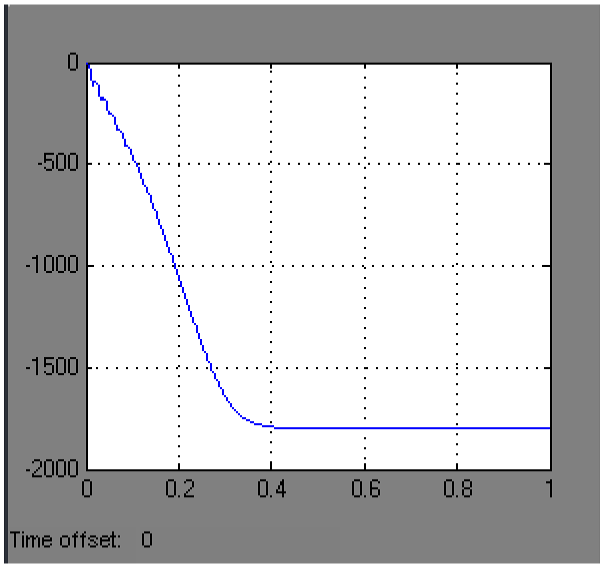


**Figure 4.17:** Motor Speed

**Figure 4.16** and **figure 4.17** shows the Electromagnetic torque and motor speed.

### 4.3.2 Simulation for Unbalanced Voltage Supply

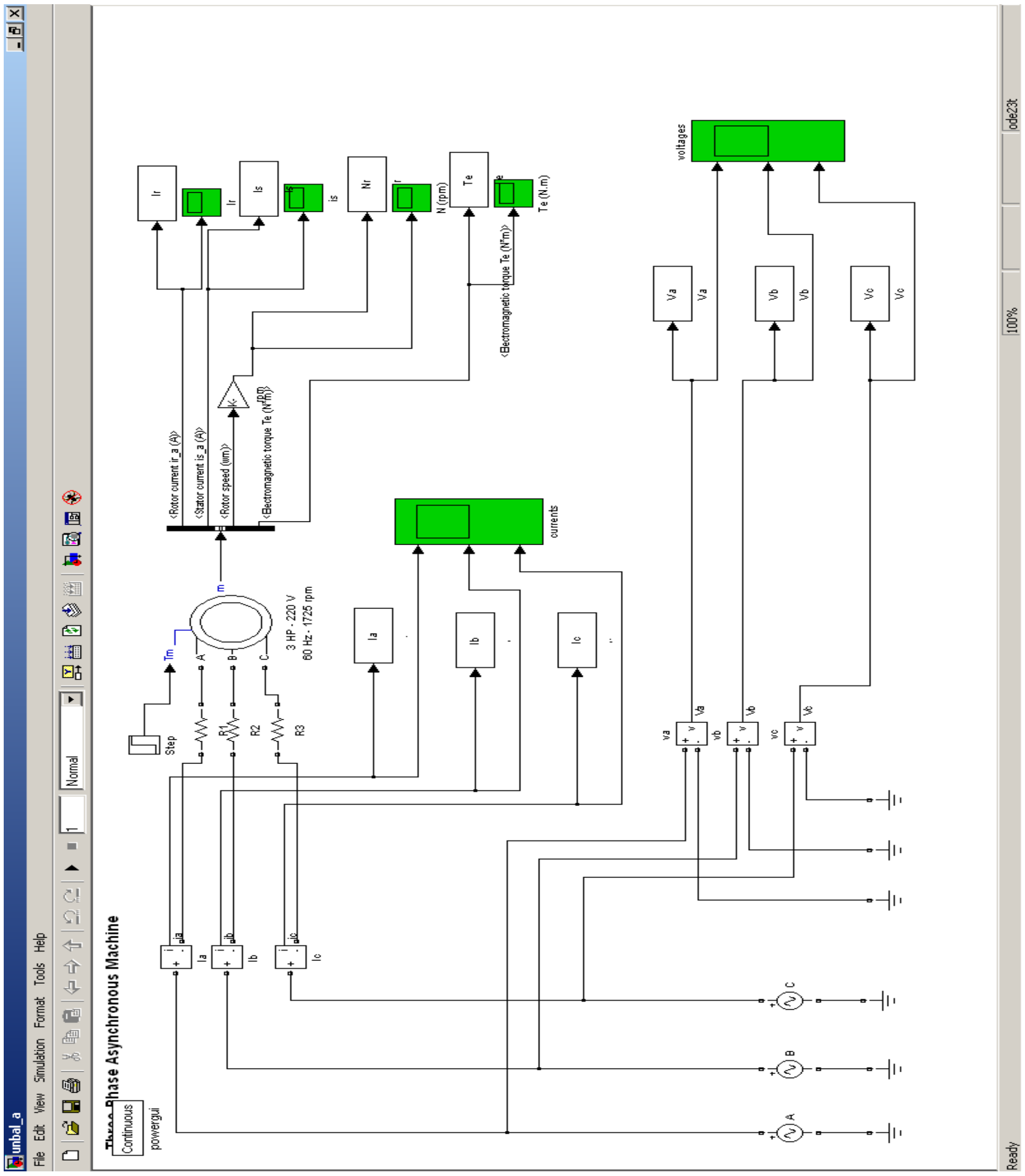


**Figure 4.18:** Simulation for Unbalanced Voltage Supply

Run the simulation, the result can be obtained from the scope block.

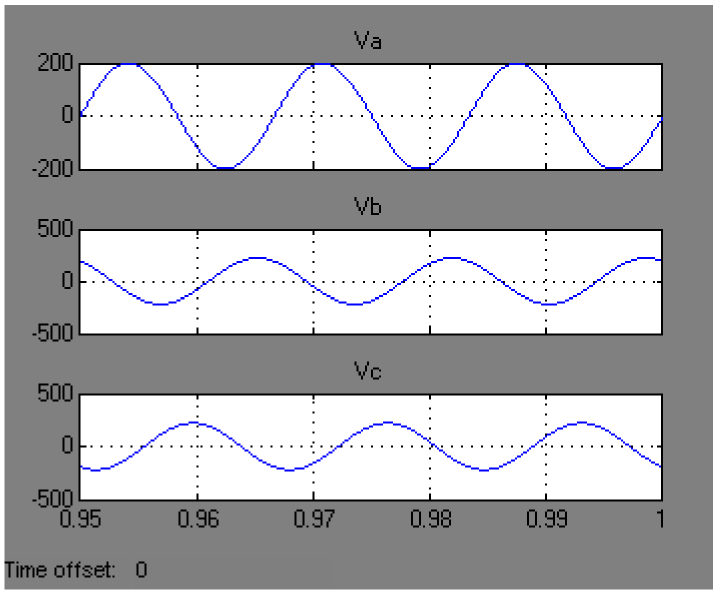


**Figure 4.19:** Three phase voltages

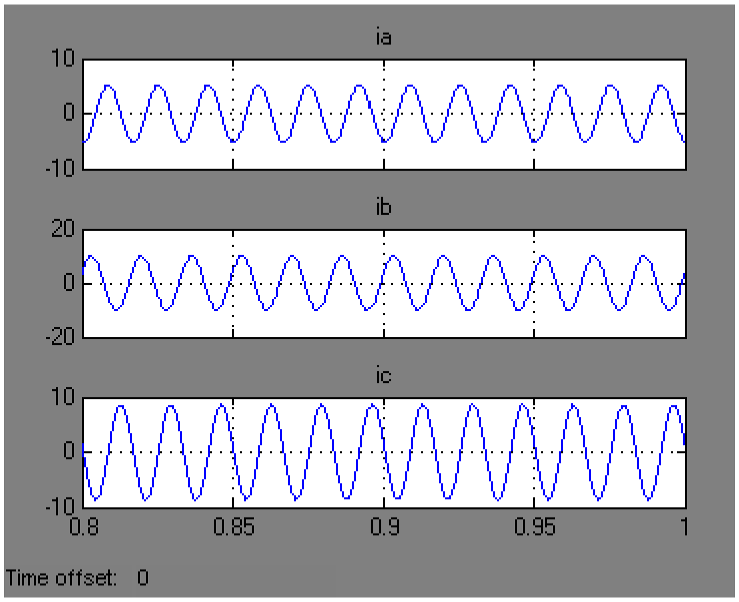


**Figure 4.20:** Three phase currents

**Figure 4.19** and **figure 4.20** show the respected voltages and currents.

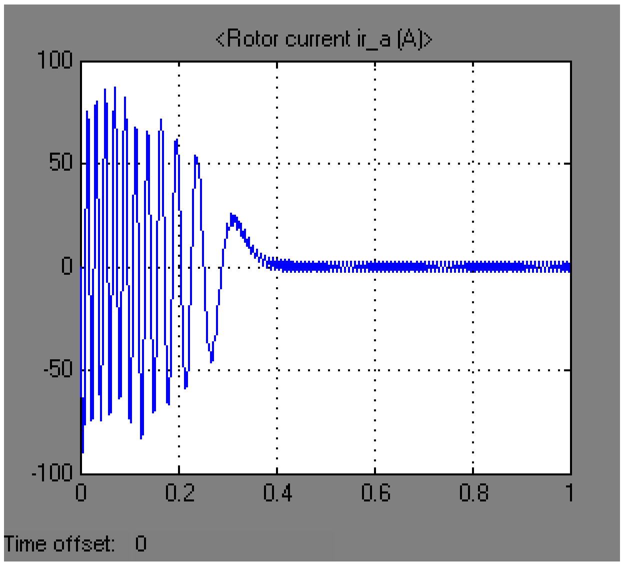


**Figure 4.21:** Rotor current

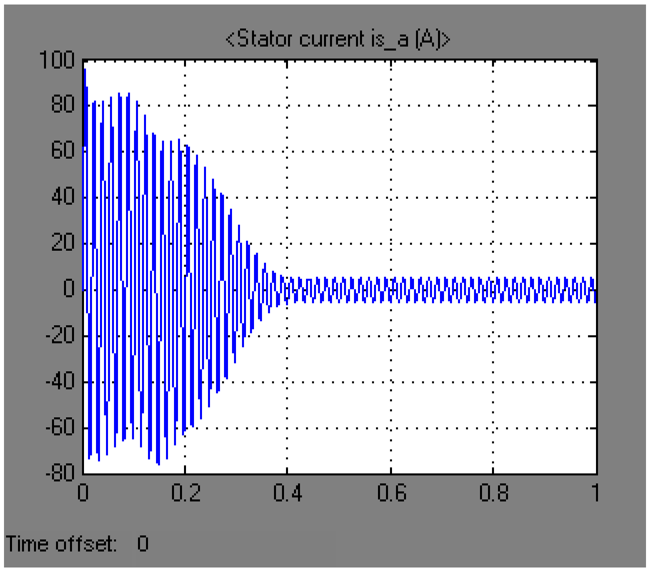


**Figure 4.22:** stator current

**Figure 4.21** and **figure 4.22** show the rotor and stator current.

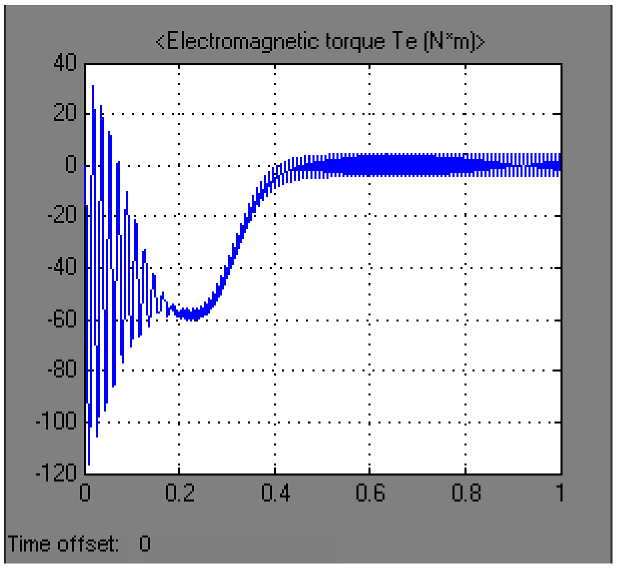


**Figure 4.23**: electromagnetic torque

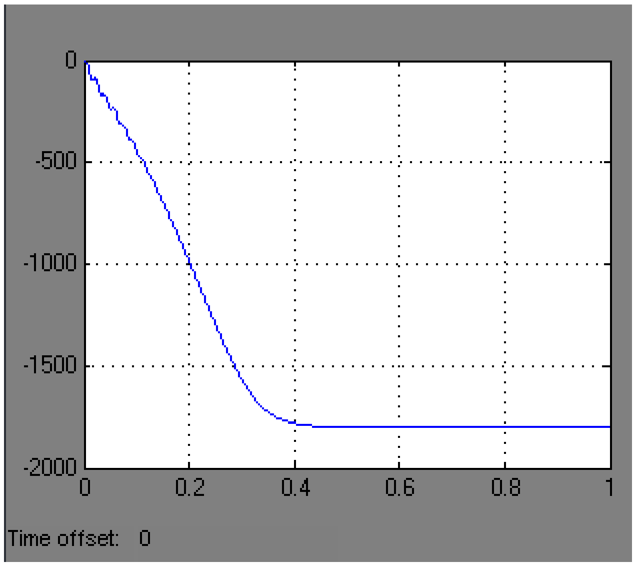


**Figure 4.24:** Motor speed

**Figure 4.23** and **figure 4.24** shows the Electromagnetic torque and motor speed.

### 4.3.3 Simulation for Single Phase Open Circuit

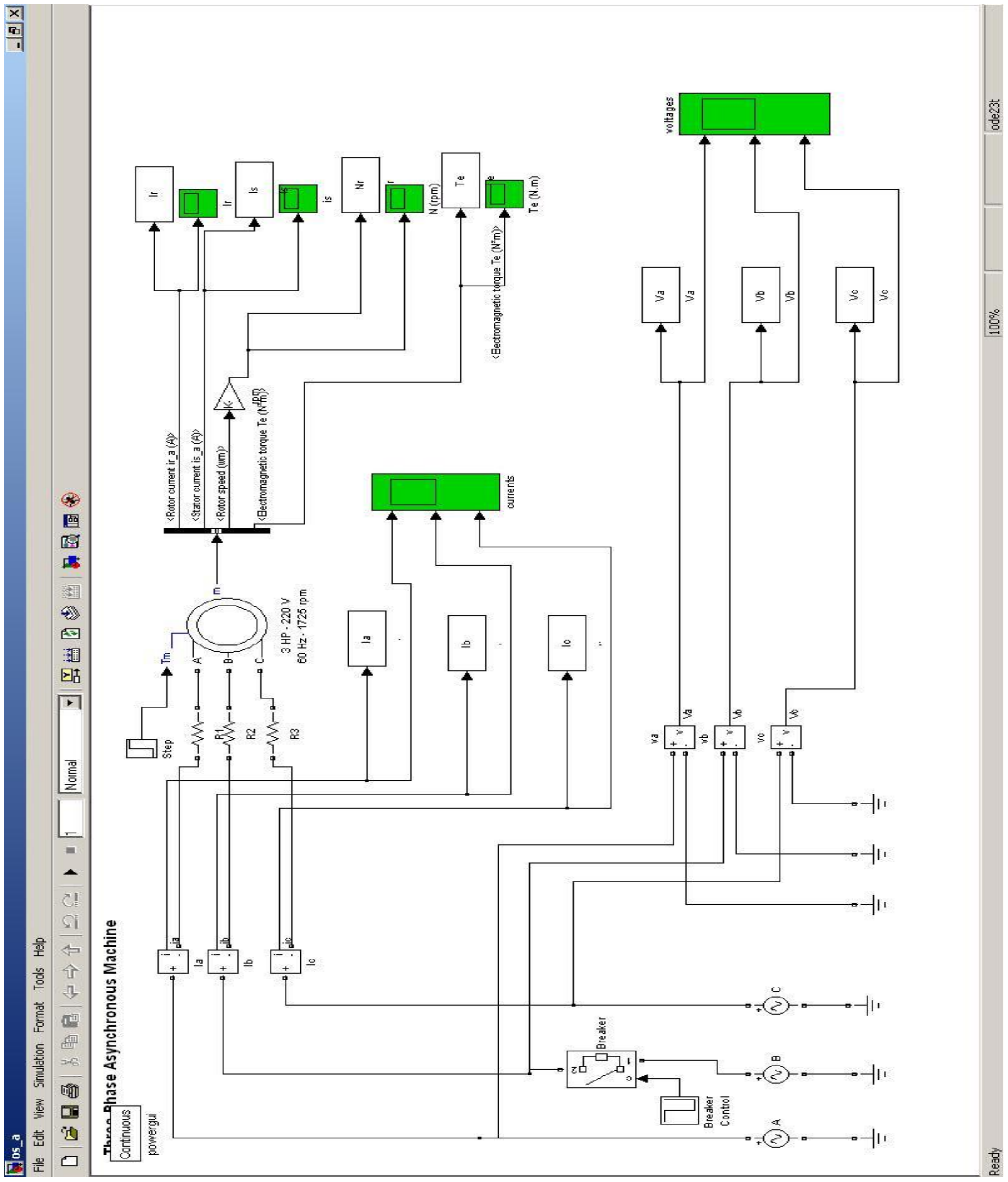


**Figure 4.25:** Simulation for Single Phase Open Circuit

Run the simulation, the result can be obtained from the scope block.

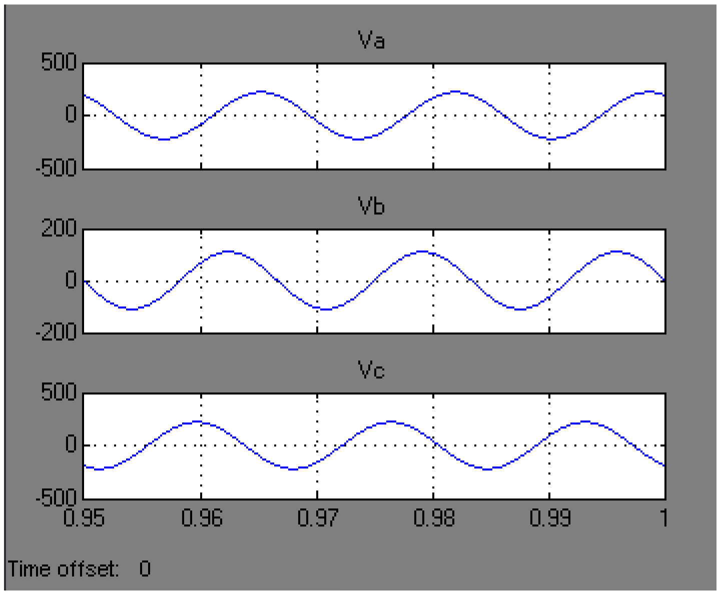


**Figure 4.26:** Three phase voltages

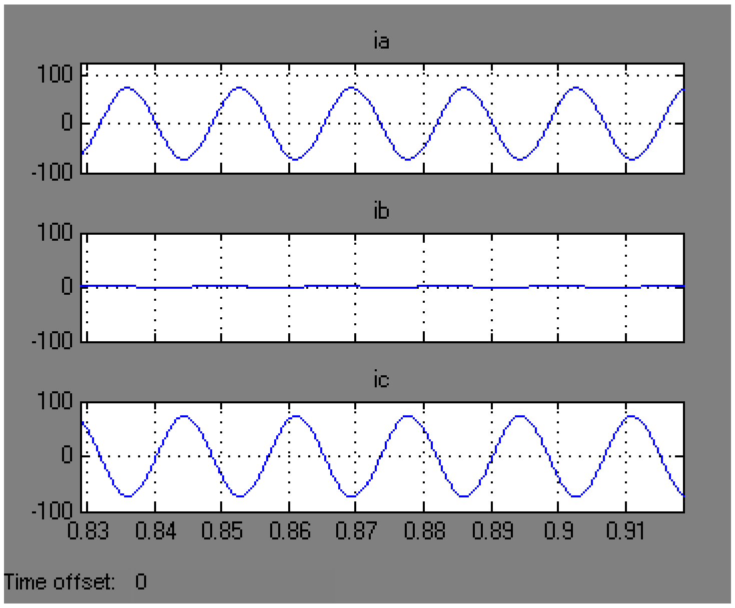


**Figure 4.27:** Three phase currents

**Figure 4.26** and **figure 4.27** show the respected voltages and currents.

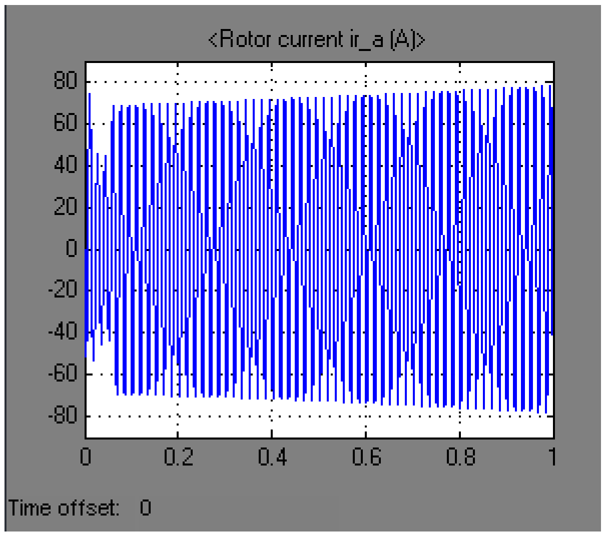


**Figure 4.28:** Rotor current

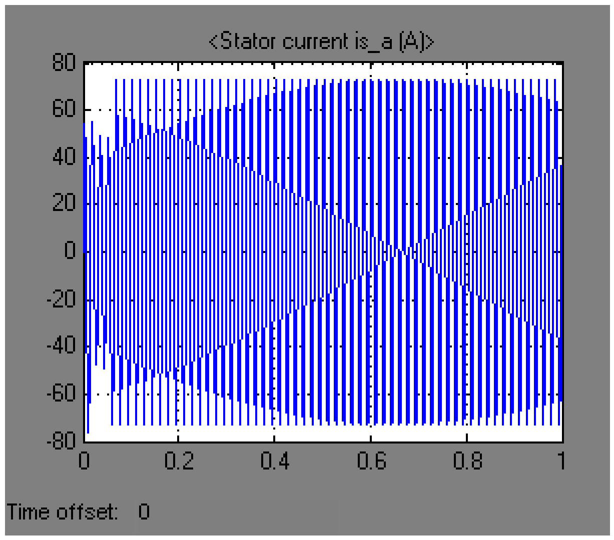


**Figure 4.29:** Stator current

**Figure 4.28** and **figure 4.29** show the rotor and stator current.

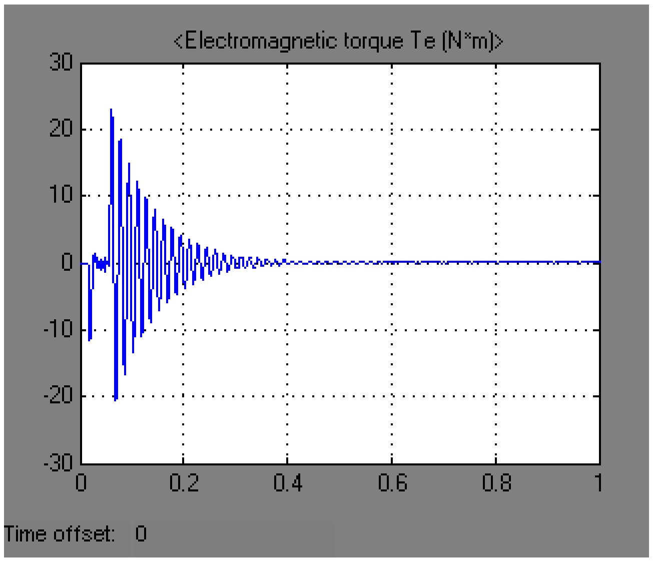


**Figure 4.30:** Electromagnetic torque

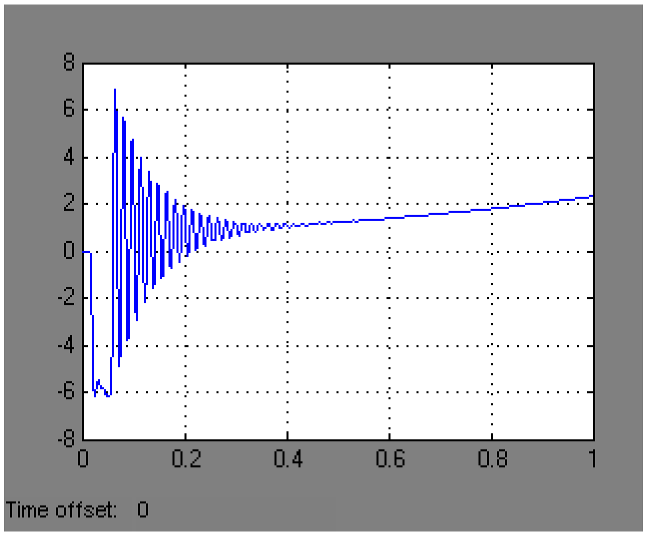


**Figure 4.31:** Motor speed

**Figure 4.30** and **figure 4.31** shows the Electromagnetic torque and motor speed.

## 4.4 Result Analysis

From the results obtained, we can see changes in graph while comparing the faulty state result and the healthy state results. Some of them were only slightly different, but there are graphs that show major differences compared to a healthy one.

### 4.4.1 Inter-turn Short Circuit

There are not much different between healthy state and inter-turn short circuit's results. But there is a slightly different in rotor current. Refer **figure 4.6** and **figure 4.14** for comparison. It looks same during the transient state, but the difference showed up during the steady state. There is a small ripple at the steady state for the inter-turn short circuit model.

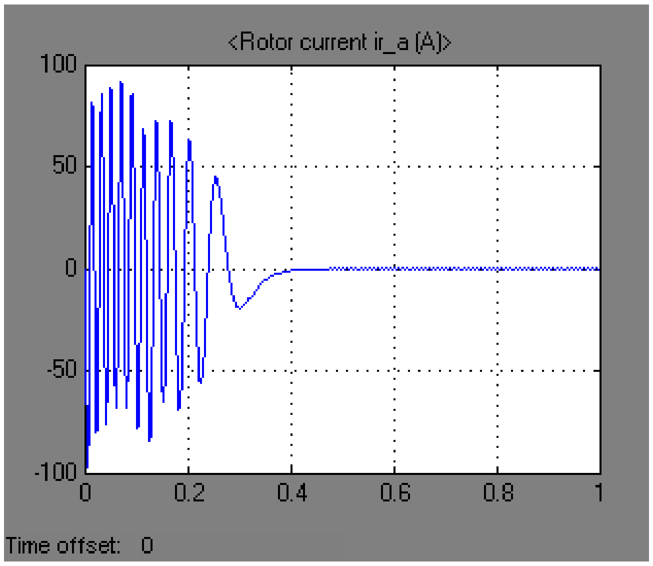


**Figure 4.14:** rotor current for interturn short circuit

### 4.4.2 Unbalanced voltage supply

Since the voltage source for unbalanced voltage supply model has been manipulated, the graphs for the voltages differ from the healthy state model. The same goes with the currents, rotor current and electromagnetic torque. Refer **figure 4.19**, **figure 4.20**, **figure 4.21** and **figure 4.23** for comparison.

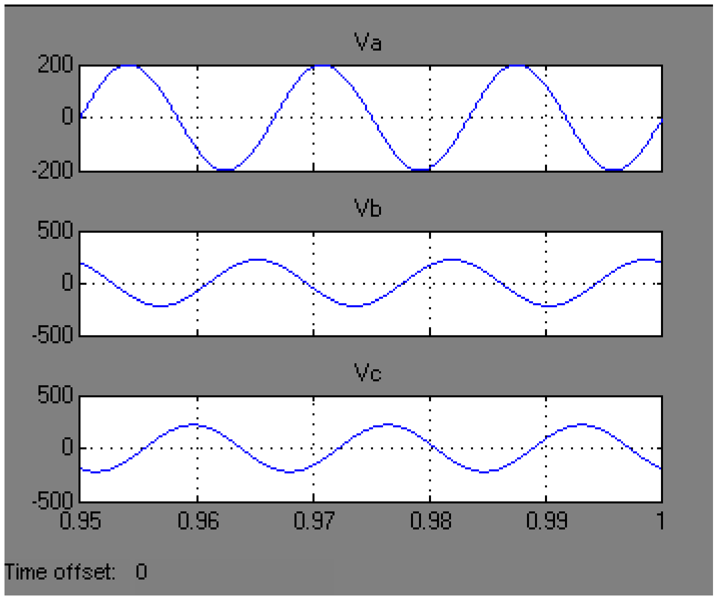


**Figure 4.19:** voltages for unbalanced voltage supply

From the figure above, we can see that the amplitude of the graph is differing from each other. The same thing happens with the currents graph, which will be shown at **figure 4.20**.

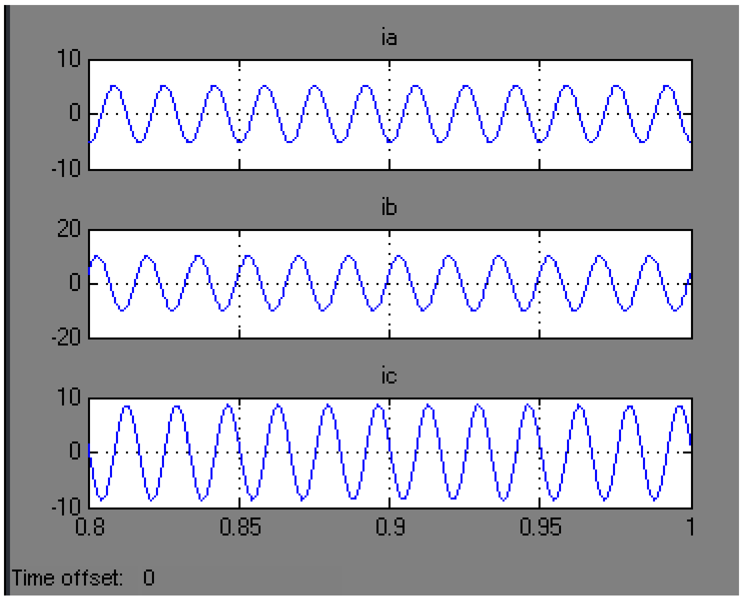


**Figure 4.20**: currents for unbalanced voltage supply

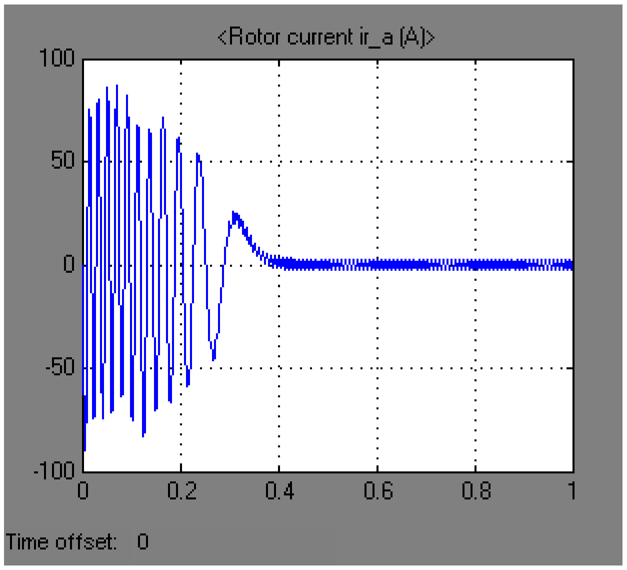


**Figure 4.21**: rotor current for unbalanced voltage supply

For rotor current, it shows a greater scale of ripple during the steady state phase compared to the rotor current for interturn short circuit model (**figure 4.14**).

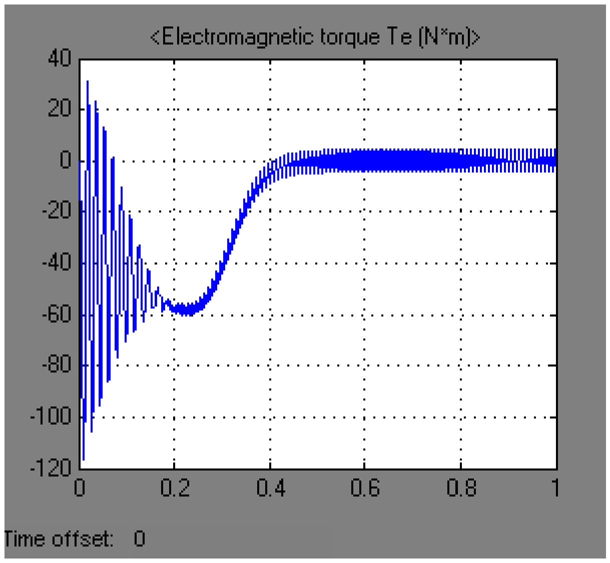


**Figure 4.23:** electromagnetic torque for unbalanced voltage supply

For electromagnetic torque, the ripple also greater at the steady state compared to the healthy stage.

### 4.4.3 Single phase open circuit

The result obtained from the single phase open circuit model show the largest differences compared to the healthy model. Since one phase is intentionally opened, the motor model could not work properly, thus affected the result obtained. When one phase is open circuit, the circuit become incomplete and the current reading become zero. **Figure 4.27** below will show the differences.

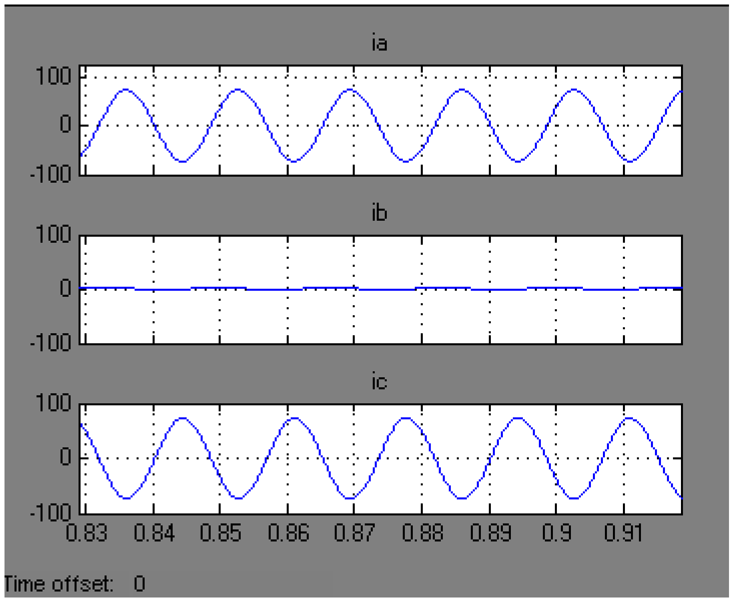


**Figure 4.27:** currents for single phase open circuit

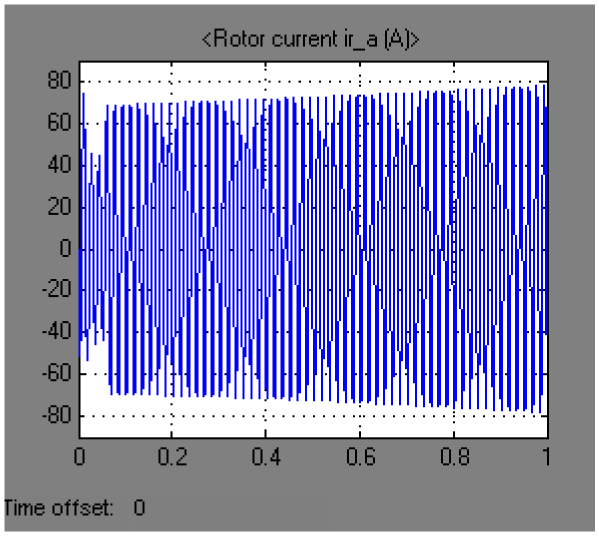


**Figure 4.28:** rotor current for single phase open circuit

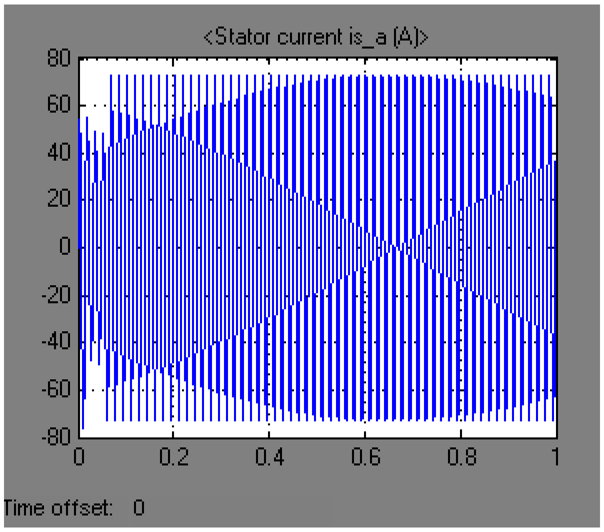


**Figure 4.29:** stator current for single phase open circuit

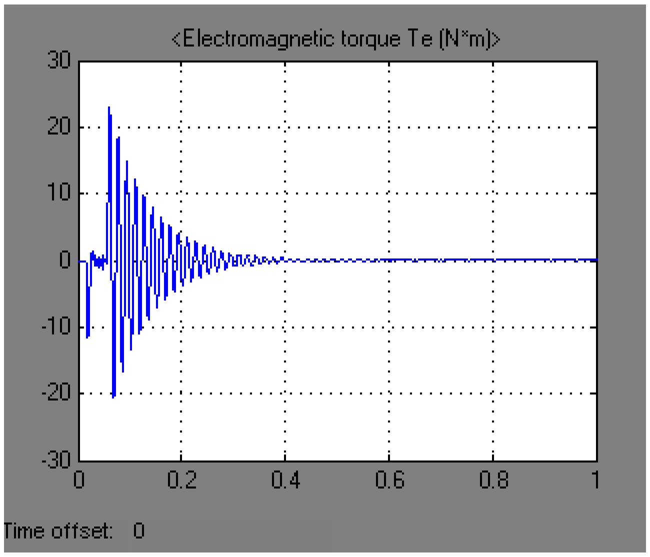


**Figure 4.30:** electromagnetic torque for single phase open circuit

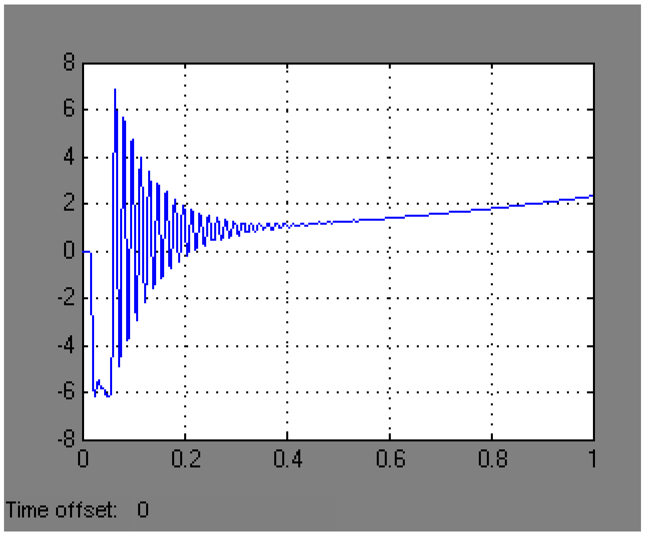


**Figure 4.31:** Motor speed for single phase open circuit

The rotor current, stator current, electromagnetic field and the motor speed also show differences from a healthy one. Since one phase is opened, the motor not operate and the speed becomes zero.

# CHAPTER 5

# CONCLUSION AND RECOMMENDATION

## 5.1 Conclusion

The purpose of this final year project, the Simulation of electrical faults of three phase induction motor drive system is to study the induction motor when fault occur. MATLAB tools are used in this project to prevent the motor from being damaged when the faults are intentionally applied to the motor.

The data from the simulation can be as references when monitoring the motor. The data from actual motor can be compared to the data from simulation, the differences can be assumed as the motor may have some problem. This is important to detect any abnormalities in the motor, so that any fault occur can be detect at early stage.

## 5.2 Recommendation

For future recommendation, several suggestions are proposed:

- Replace the simulation model with actual motor to analyze real time theoretical data where it is time consuming.
- The data obtained from this project can be used as references to develop fault detection system.

# REFERENCES

[1] 24 February 2009, Citing Internet Sources URL http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=02nQXGrrlPglzQMszY

[2] 24 February 2009, Citing Internet Sources URL www.geocities.com/cindulkar/lab41.pdf

[3] 8 March 2009, Citing Internet sources URL http://en.wikipedia.org/wiki/Rotor

[4] 8 March 2009, Citing Internet Sources URL http://en.wikipedia.org/wiki/Squirrel-cage_rotor

[5] 8 March 2009, Citing Internet Sources URL http://www.allaboutcircuits.com/vol_2/chpt_13/8.html

[6] Rakesh Parekh. (2003). *AC Induction Motor Fundamentals:* Microchip Technology Inc.

[7] M. Arkan, D. Kostic Perovic, P.J. Unsworth. (2005). *Modeling and Simulation of Induction Motor with Inter-turn Faults for Diagnostics*

[8] Liang B. Payne B. S. Ball A. D. Iwnicki S. D. (2002). *Simulation and fault detection of three-phase induction motors.*

[9] 19 March 2009 , Citing Internet Sources URL http://www.elec.uow.edu.au/iepqrc/files/technote6.pdf

[10] R. Rajeswari an d N. Kamaraj . *Diagnosis of Inter Turn Fault in the Stator of Synchronous Generator Using Wavelet Based ANFIS*. World Academy of Science, Engineering and Technology 36 2007

[11] Vic Gosbell, Sarath Perera, Vic Smith. *Technical Note No. 6.* October 2002

[12] H. W. Penrose and J. Jette, "Static motor circuit analysis: An introduction to theory and application," *IEEE Electr. Insul. Mag.*, vol. 16, no. 4, pp. 6–10, July /Aug. 2000.

[13] M. A. Cash, H. G. Habetler, and G. B. Kliman, "Insulation failure prediction in AC machines using line-neutral voltages," *IEEE Trans. Ind. Appl.*, vol. 34, no. 6, pp. 1234–1239, Nov./Dec. 1998.

[14] H. A. Toliyat and S. Nandi, "Novel frequency-domain-based technique to detect stator interturn faults in induction machines using stator-induced voltages after swich-off," *IEEE Trans. Ind. Appl.*, vol. 38, no. 1, pp. 101–109, Jan./Feb. 2002.

## APPENDIXE A

(Program for GUI)

**Program for main window GUI**

```
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 04-Nov-2009 06:51:14

% 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 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)
movegui ('center')
[c,map]=imread('UMP','JPG');
image(c)
set(gca,'visible','off')

% 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 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)
user_response = about, close main;


% --- 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)
user_response = PRE1, close main;


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
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
```

**Program for about window**

```
function varargout = about(varargin)
% ABOUT M-file for about.fig
%      ABOUT, by itself, creates a new ABOUT or raises the existing
%      singleton*.
%
%      H = ABOUT returns the handle to a new ABOUT or the handle to
%      the existing singleton*.
%
%      ABOUT('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in ABOUT.M with the given input
arguments.
%
%      ABOUT('Property','Value',...) creates a new ABOUT or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before about_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to about_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 about

% Last Modified by GUIDE v2.5 04-Nov-2009 09:45:41
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @about_OpeningFcn, ...
                   'gui_OutputFcn',  @about_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 about is made visible.
function about_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 about (see VARARGIN)
movegui ('center')
[c,map]=imread('im','JPG');
image(c)
set(gca,'visible','off')



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

```
% Update handles structure
guidata(hObject, handles);

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



% --- Outputs from this function are returned to the command line.
function varargout = about_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 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)
user_response = main, close about;



% --- 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)
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
```

**Program for PRE1 window**

```
function varargout = PRE1(varargin)
% PRE1 M-file for PRE1.fig
%      PRE1, by itself, creates a new PRE1 or raises the existing
%      singleton*.
%
%      H = PRE1 returns the handle to a new PRE1 or the handle to
%      the existing singleton*.
%
%      PRE1('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PRE1.M with the given input
arguments.
%
%      PRE1('Property','Value',...) creates a new PRE1 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before PRE1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to PRE1_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 PRE1

% Last Modified by GUIDE v2.5 04-Nov-2009 10:51:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PRE1_OpeningFcn, ...
                   'gui_OutputFcn',  @PRE1_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 PRE1 is made visible.
function PRE1_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 PRE1 (see VARARGIN)
movegui ('center')
[c,map]=imread('UMP','JPG');
image(c)
set(gca,'visible','off')

% Choose default command line output for PRE1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes PRE1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PRE1_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 pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (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 PRE1;


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
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



% --- 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)
user_response = run_healthy;

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = PRE2, close PRE1;
```

**Program for PRE2 window**

```
function varargout = PRE2(varargin)
% PRE2 M-file for PRE2.fig
%      PRE2, by itself, creates a new PRE2 or raises the existing
%      singleton*.
%
%      H = PRE2 returns the handle to a new PRE2 or the handle to
%      the existing singleton*.
%
%      PRE2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PRE2.M with the given input
arguments.
%
%      PRE2('Property','Value',...) creates a new PRE2 or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before PRE2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to PRE2_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 PRE2

% Last Modified by GUIDE v2.5 04-Nov-2009 13:07:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
```

```
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PRE2_OpeningFcn, ...
                   'gui_OutputFcn',  @PRE2_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 PRE2 is made visible.
function PRE2_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 PRE2 (see VARARGIN)
movegui ('center')
[c,map]=imread('UMP','JPG');
image(c)
set(gca,'visible','off')



% Choose default command line output for PRE2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes PRE2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PRE2_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 pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = PRE1, close PRE2;


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
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


% --- 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)
user_response = run;


% --- 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)
user_response = run_unbal_a;


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = run_os_a;
```

**Program for closing window**

```
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

% 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(gcbf)
    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(gcbf,'Units');
    set(gcbf,'Units','pixels');
    GCBFPos = get(gcbf,'Position');
    set(gcbf,'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
```