Design & Optimisation of the Flux Switching Motor and Drive with Genetic Algorithms

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Kao Siang Chai

Department of Engineering
University of Leicester

June 2004

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

The flux switching (FS) motor is a new class of reluctance machine that has demonstrated potential as a possible replacement for brushed-dc motor in many applications. However the design and optimisation of the motor and its drive system are rather complicated and not much past knowledge and guidelines are available to aid the engineer(s) in the design of the machine.

The development of flexible and versatile design optimisation software to facilitate the design and optimisation of FS motor and drive is presented. The design optimisation software incorporates a genetic algorithm optimisation tool and dynamic simulation model with third party finite element analysis software.

The developed genetic algorithm optimisation program integrated with finite element analysis (FEA) software provides the engineer with the necessary optimisation tools capable of interfacing with the FEA software. This has allowed many FS motor lamination designs to be created without any requirement of user feedback once the program is initialised. In addition the application of the developed design tool can also be extended to other electromagnetic devices.

A dynamic simulation model of the FS motor drive system has been developed. The model can either be used as a standalone program or be integrated into the optimisation software. The dynamic simulation model consisted of a simple time-stepping electrical equivalent circuit coupled with a switch control algorithm, a winding optimisation model and an iron loss model. When interfaced with the FEA software it can support rapid estimation of the motor dynamic performance.

The developed optimisation software has been used to design and optimise FS motors and the results have demonstrated the potential of genetic algorithms in design optimisation of the machine.

Acknowledgements

This thesis is the result of a part of my Ph.D. work completed in the Centre for Advanced Electronically Controlled Drives headed by Prof. Charles Pollock, a division of the Department of Engineering in the University of Leicester. My work in this research centre is financed by both the department and the centre, and I would like to express my gratitude to both parties for making this possible.

I would most of all like to thank my supervisor Prof. Charles Pollock for the support and guidance, without whom the work described in this thesis could not have taken place. I would like to acknowledge the influence of my current and former colleagues with whom I have worked with during this project, in particular Dr John Reeve and Dr. Takashi Kosaka whom I have many interesting discussions and in the process have interactively contributed to my ideas. I would also like to thank the industry members of the research centre for their support of this work and Kenwood Ltd. for support with the construction of the motor.

Outside the workplace, I would like to thank my parents, other family members, and friends for their continuous support and encouragements.

Kao Chai,

June 2004

Table of contents

List of figures

List of tables

List of main symbols used

Symbol	Description	Units
B	Magnetic flux density	Tesla, T
e	Electro motive force, emf	Volt, V
f	Rate of frequency	Hertz, Hz
i	Current	Ampere, A
k	Constant	
L	Inductance	Henry, H
L_{stk}	Stack length	Metre, m
mmf	Magneto motive force	Ampere turns, A.turns
N	Integer number	
P	Power	Watt, W
R	Resistance	Ohm, Ω
T	Time	Second, s
T_{em}	Torque	Newton metre, Nm
V	Voltage	Volt, V
ϕ	Magnetic flux	Weber, Wb
λ	Magnetic flux linkage	Volt seconds, Vs
θ	Rotor angle	Degree, °
ρ	Resistivity	Ohm metre, Ωm
ω	Radian frequency	Radians per second, $rads^{-1}$

Subscript

a	Armature winding	f	Field winding
s	Stator	r	Rotor

1 Introduction

1.1 Background

Living in modern days, electric motor has unwittingly integrated into a feature of daily life, it is no longer the large "stand-lone" device that we have used to know. It has become literally invisible, embedded inside thousands of everyday products. They can be found inside washing machines, vacuum cleaners, refrigerators, and disc drives etc. However, we may often fail to notice such numerous motors by our own eyes. Whatever the size, electric motors have grown to be of common use in our society. While there are so many different classes of electric motor, the type of electric motor that would be discussed within the context of this thesis is the new flux switching motor (FSM) [1]. The flux switching motor is a new class of reluctance machine; experiments of early prototype have offered performance advantages over brushed motor in many applications.

The design optimisation of a reluctance machine and drive system for a given specification is a multi-dimensional problem. Achieving the design objectives often requires tens or even hundreds of mechanical and electrical parameters to be tuned. Within the motor, the size, aspect ratio, lamination shape, number of poles, air-gap radius and thickness, lamination steel choice are all variables which have a dramatic impact on the motor performance. The design of the windings, connection methods and slot areas is very dependent on the power converter topology, operating voltage, value of dc link capacitance and switching angles. The complete system performance is therefore dependent on all of these factors and design optimisation process must take account of all the variables in the system. The interdependent design variables make the design space multi-dimensional and virtually unsolvable by analytical techniques.

The traditional analytical method is used to design electric motor before the Finite Element Method analysis (FEA) was adapted. Using circuit theory, phasor diagrams, and equivalent circuit techniques to represent the operation of an electric motor under specific conditions. Although these techniques have been refined over time, they are still suited to relatively simple electromagnetic circuits. Considerable experience is required because of the need to include correction factors, which must be introduced to minimise the difference between measured and predicted performance. Over the decades, with advances in FEA and computer hardware, FEA has become an integral

part of the electric motor design process. FEA has contributed to cut electric motor design cost and timescales by reducing the number of tests and prototyping. However, it is still just a tool for estimating electromagnetic performance of the electric motor for a given configuration. On the whole, the designer still have to use his existing or real-time learnt knowledge to modify and adjust the design parameters to derive a satisfactory final design. Such design process could be tediously long and, in the end, the design may not be the optimal one.

Based on past experiences [2-7], it is understood that a multi-dimensional type of engineering design problem such as the optimisation of an electric motor and drive system can be solved more effectively by implementing evolutionary computing. Genetic Algorithms (GAs) are one such evolutionary computational technique which tend to search the design space intelligently and not exhaustively. One key feature of GAs is that they search from multiple points in the design space, instead of moving from a single point like gradient-based methods, nor like the enumerative methods that looked at every point of the design space one at a time. Genetic algorithm is proven to be more efficient (with higher probability of converging to an optimal solution in lesser time) in solving discontinuous, multimodal, and noisy problem compared with other deterministic search techniques [8].

The much reduced search time makes design automation possible within a realistic timescale and the multiple solutions make the designs reliable and suitable for multiple objectives. Furthermore, ever-advancing computer processor speed will further reduce the search time. Powerful and versatile enabling technology based on evolutionary computing has an unmatched potential in modelling and design automation of electrical machines, drives and control systems. The underlying aim of this research is to develop such methodology to extend the present bounds of performance and to materialise a rapid transfer of the new flux switching motor and drives technology to commercial products by reducing the design cycle and development gearing. Since the success of any commercial products depends on the cost and timeliness as well as quality. As a consequence, the design process is being reengineered to save cost and timescales.

The FS motor drive technology has been used in several applications traditionally served by other drive types. As the FS technology is drawing attention of a growing list of manufacturers, some comparisons of the characteristics of FS and other motors seem appropriated.

In domestic appliance, FS motor vs. Universal motor

- The life of a universal motor is usually limited by the wear of the commutator and carbon brushes, which can be further reduced by working environment. While the brushless FS motor has no such worries, and due to the simple and rugged motor construction, the FS motor can operate in environment of high temperature and vibration.
- The speed of the universal motor decreases as load increases, this inherent characteristic of the universal motor become critical, when application (such as sawing) requires the motor to deliver flatter output power over a wider speed range. In comparison a FS motor of similar frame size can deliver a flatter, more constant speed load profile [19].

In automotive sub-system, FS motor vs. other drive types

- PM brushed motors are commonly employed in automotive sub-system although it provides the low cost option but there is concern over the lifetime of the brushes.
- Brushless dc motors and drives offer very high performance but they are still relatively expensive to manufacture.
- Switched reluctance (SR) motors have rugged construction but the cost of its drive is still considered as too expensive for the very cost competitive applications.

1.2 Thesis objectives

The objective of this research is to develop a versatile and efficient computer-automated design method in design and optimisation of the flux switching motor and drive such that the technology can be rapidly transferred to commercial products. To achieve this goal, the following would be investigated.

Stage 1: Integrate the FEA with genetic algorithms

Central to this task is the development of a general-purpose genetic algorithm, which offers versatility and flexibility to its user, as well as its capability to interface with finite element analysis software. The integration of the genetic algorithms with a FEA software package liked OPERA-2D, allows multiple design variables of an electrical machine to be tuned simultaneously, and produces a wide range of machine laminations. The integrated software would ideally displace or reduce any requirements of expert feedback during the design loop, such that the automated design cycle would free up designer time for conceptual design.

Stage 2: Develop an accurate and versatile FSM and drive simulation system

Before a genetic algorithm can be used to optimise a motor and drive system an accurate and very versatile simulation system must be developed. The simulation model should be capable of modelling the interaction between the motor, the drive circuit and the power switch control algorithm, in order to predict the electromechanical performances under a range of operating speeds and voltages. Nonetheless, the simulation should be rapid enough to be of practicable use within an iterative design process. With the completion of both stages a computer-automated design process in design optimisation of flux switching motor and drive is realisable.

Stage 3: Design and optimisation of the flux switching motor and drive based on GAs

To confirm the ability of the genetic algorithms in electrical machine design optimisation, the design optimisation of flux switching motor and drive system for two different applications will be demonstrated in chapter 6 and 7. What an optimal design is will be very dependent on the application, but inevitably reduced cost, both initial manufacturing cost and running cost, will be design targets.

1.3 Genetic Algorithms

1.3.1 What are genetic algorithms?

Genetic Algorithms are a class of evolutionary algorithms first proposed by John Holland in the 1970s. They were search algorithms invented to mimic the processes observed in the biological evolution [9]. Holland was inspired by the Darwinian notion of evolution in which only the fittest survive. He believed that appropriately incorporated in a computer algorithm, the GAs might yield a technique for solving difficult problems in the way that nature has done through evolution. There are three features which distinguish GAs as first proposed by Holland from other evolutionary algorithms; i) binary numbers are used as *representation*, ii) the *proportional selection*, and iii) *crossover* as the primary method of producing variations.

Many subsequent GA implementations [8,10] have adopted alternative methods of selection, and many have abandoned binary representations for other representations depending on the problems. Also many alternative methods of crossover have been proposed, but a simple GA that produces good results in problems normally comprised three important operators: *Selection*, *Crossover* and *Mutation*.

The GA begins its search for optimum from a randomly created initial population of *N individuals*, each of them characterised by a string of binary numbers, 1s and 0s. The string is composed of some concatenated sub-strings, where each sub-string describes a design variable. The design variables encoded into finite length binary strings, are often referred to as *chromosomes* where a single design variable corresponds to one *gene*. They are the *genotypes* that are manipulated by the GA. The *evaluation* routine decodes these structures into some *phenotypical* structure and assigns a *fitness value*. Each individual represents a search point in the space of potential solution to a problem at hand. The idea of survival of the fittest is of great importance to genetic algorithms. GA uses *fitness function* to evaluate the "fitness" of each individual and assigns a fitness value to each individual. According to the fitness measure, the selection process favours better individuals to reproduce more often than those that are relatively worse. Thus the better individuals has a higher chance of forming the new population, some individuals of the new population will undergo transformation by means of genetic operators to form new solutions. So in each *generation*, the GA creates a set of strings from parts of the previous strings, occasionally adding random new data to keep the population from stagnating. All these operations are merely a copying strings and exchanging partial strings. On average, the individuals of genetically evolved new population represent improved points in the solution space. The search continues until a stopping criterion is reached. Despite the simplicity of the operation, GAs have vast and proven potential for solving difficult optimisation problems [8-10].

1.3.2 Representation of individuals

In most applications of GAs, decision variables are coded in binary strings of 1s and 0s. The variables can be integer or real numbers, they are represented by binary numbers of a specific length depending on the required accuracy in the solution. For example, a real number R bounded in the range (a, b) can be coded in a five-bit string with the strings 00000 and 11111 representing the real numbers a and b respectively. With any of the other 30 strings represents a solution in the range (a, b), and the maximum attainable accuracy is $(b\text{-}a)/(2^5\text{-}1)$. The resolution of representation increases with the number of bits in the strings, but too fine a resolution might result in excessive hair splitting in the search for an optima. Too coarse a resolution might result in the optima never being found simply because the string cannot express a value close enough.

Although binary coding has been most popular in GAs, a number of researchers prefer to use Gray coding to eliminate the hamming cliff problem associated with the binary coding [11]. In Gray coding, the number of bit differences between any two consecutive strings is one, whereas in binary strings this is not always true. As an example, consider a five-bit parameter with a range from 0 to 31. If it is encoded using the standard binary coding, then 15 is encoded as 01111, whereas 16 is encoded as 10000. In order to move from 15 to 16 all five bits need to be changed. On the other hand, using Gray coding, 15 is encoded as 01000 and 16 as 11000, differing only by one bit. This will be a problem if two points close to each other in the representation space might be far in the binary represented problem space. As a consequence, GAs using the binary representation is unable to focus the search effort in a close vicinity of the current population. However, as in the binary strings even in Gray coded strings a bit change in any arbitrary location may cause a large change in the decoded integer value.

Although typical, GAs are not restricted to bit-string representation, there are also cases of GAs using other representations [12], such as real valued vectors, permutations and treelike hierarchies. It is clear that choice of representation is inherent to the underlying problem at hand, and a difficult problem can be made simpler by suitably choosing a representation that work efficiently. While there are several options available for representation in the GAs, only the binary string representation will be used throughout the course of this research project.

1.3.3 Mechanics of Genetic Algorithms

In spite of the diversity in GAs, they have common components: selection biased by fitness, crossover, and mutation. Selection is a process in which individuals are selected for mating based on their fitness values. Then new individuals are created by crossover and mutation operations to form the new generation. Each item listed below in the outline of a simple genetic algorithm will be presented in the following subsections.

A simple genetic algorithm would have the following outline:

i) Initialise a population of designs.

ii) Evaluate fitness of each design in a population.

iii) Select designs for "breeding" and designs to be eliminated.

iv) Apply genetic operation, crossover and mutation to create new designs.

v) Replace the eliminated designs with new designs.

vi) Check if stop criteria is met, if not return to step 2 and restart the process.

1.3.3.1 Initialisation

An initial population of designs is generated and this often is accomplished by random sampling from the design space. However, it is also possible for proven designs to be inserted deliberately into the population to avoid possibility of all randomly generated designs being impractical, especially when the initial population of designs is small.

1.3.3.2 Evaluation

The fitness of all individuals is evaluated by the fitness function(s), thus to achieve a successful search result the function should have the capability to identify between better and worse solutions. While what the fitness function would be very much depends on the underlying problem. Within the scope of the thesis, the fitness of all individuals would be measured and ranked based on the results of either FEA or dynamic drive system simulation. Noticeably, this step also consumes the most computational time within the optimisation cycle.

1.3.3.3 Selection

The primary objective of the selection operator is to emphasize better solutions in a population. This operator does not create any new solution, instead it selects relatively good solutions from a population and deletes the remaining not so good ones. In nature, an individual undergoes two different selection pressures before producing its offspring, survival to adult state and finding its mates. Selection operator in GAs models these processes and several different types of selection operator will be discussed.

Fit-Fit selection pairs an individual with the next fittest individual in the population by stepping through the ordered list of individuals. This method of selection ensured that every individual would get an opportunity to breed, not just between the fit so this method could not strictly be said to favour the strong in selection. However, the results of crossover between strong individuals are more likely to be fit than crossover between weak individuals. The weak individuals and their children will probably be replaced quickly from the population. The population does not remain static for the entire cycle through the list, meaning that individuals and their positions in the ordered list are being constantly replaced. Fit-fit selection is highly conservative of genetic information and tends to converge rapidly to one solution.

Random selection, this technique randomly selects two parents from the population for crossover. In terms of disruption of genetic codes, random selection is a little more

disruptive, on average, than roulette selection. A slight modification had been made to this technique, making it suitable for implementing it in the program in this thesis. The modification ensured that every individual would get an equal opportunity to breed.

Fitness-proportional selection - In this method, the selection probability of each individual is calculated by dividing its fitness by the sum of the fitness of all individuals. The *roulette wheel selection* and the *Stochastic Universal Sampling* (SUS) are two methods of this class. Figure 1.1 shows the operation of the roulette wheel selection that assigns a portion of the wheel proportional to the selection probability and starts spinning the roulette wheel; each time a single individual is selected. The most important concern in a stochastic selection is to prevent loss of population diversity also known as *genetic drift*. For instance, in the roulette wheel selection, it is possible that only the individuals A and B are selected to produce all the offsprings. The basic consideration of the SUS is to prevent genetic drift by selecting several parents for each wheel spin. Figure 1.2 shows an example where four parents are selected at a single wheel spin.

Individual	fitness	Selection probability
A	40	0.4
B	30	0.3
C	20	0.2
D	10	0.1

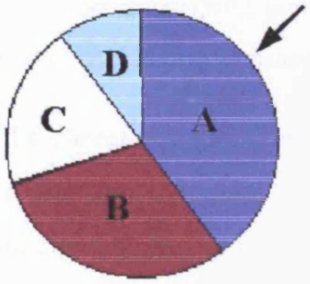


Fig. 1.1 Roulette wheel selection

Individual	fitness	Selection probability
A	40	0.4
B	30	0.3
C	20	0.2
D	10	0.1

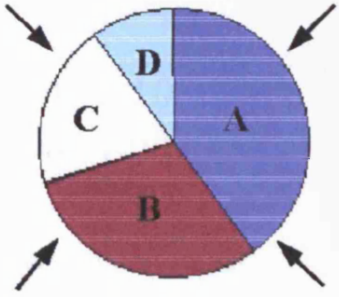


Fig. 1.2 Stochastic universal sampling

Although there are many selection methods being implemented in GAs, the appropriate level of selection pressure is the key to a successful evolution. An evolution under too strong selective pressure leads to premature loss of diversity and results in premature convergence of the GA. It is also known that the used of roulette-selection, could lead to diversity in a population to be lost in early generations due to domination of the entire population by a few "super" individuals that are much better than the average fitness [7,9]. On the contrary, low selective pressure can make the search ineffective. Thus, it is important to find a proper balance between the two.

1.3.3.4 Crossover

In nature, crossover occurs when two parents exchange parts of their corresponding chromosomes. In a genetic algorithm, crossover recombines the genetic material in two parents' chromosomes to make two children. The intuitive idea behind crossover is easy to state: given two highly fit individuals, but for different reasons, ideally what we would like to do is create a new individual that combines the best features from each. Of course, the difficulty is in what physical trait displays that underlying genetic quality, so the best we can do is to recombine features at random. This is how crossover operates.

Single-point crossover, two selected strings from the parent population are lined up, and then creating two new individuals by swapping the bits after a random or predetermined point. By keeping the head sub-string the same and exchanging the tail sub-strings, two children strings, each made up of "genetic information" from both parents are generated. Then these two entirely new strings move on to the new generation. As an example, say that the strings '10000' and '01110' are selected for crossover and the GA decides to mate them. If the GA selects a splicing point of 3. The following then occurs:

Fig. 1.3 An example on single-point crossover

The newly created strings are '10010' and '01100'.

Two-point crossover as the name suggests, they are random or predetermined selected crossover sites. Only the "genetic information" that lay between the two sites is

exchanged. As an example, supposed that '10011001' and '01100100' are selected for crossover and GA selects two splicing points of 3 and 5.

Fig. 1.4 An example of two-point crossover

Then offspring of this crossover are '10000001' and '01111100'

Uniform crossover eliminates the positional bias inherent in single and two-point crossover method. There are two main steps to perform a uniform crossover, first step is to randomly determine n bits to be exchanged where n must be smaller than the string length, l. In the second step, nth location of the string is determined, and the bits after the location are exchanged.

1.3.3.5 Mutation

Selection and crossover alone can obviously generate a staggering amount of differing strings. However, depending on the initial population chosen, there may not be enough variety of strings to ensure the GA sees the entire problem space. Or the GA may find itself converging on strings that are not quite close to the optimum it seeks due to a bad initial population. Mutation is the best-known mechanism for producing variations in the population. It can be performed either during selection or crossover (though crossover is more usual). It is the occasional random alteration of bits within the new strings, based on a mutation probability value, which dictates the frequency at which mutation occurs. In binary coding, this simply means changing a '0'to '1' or vice versa. A commonly used rate of mutation is one over the string length as shown in Fig. 1.5.

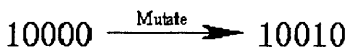


Fig. 1.5. An example on mutation

1.3.3.6 Replacement and Elitist Strategy

Replacement, the idea is simply to remove the weakest individuals in the population with new randomly generated individuals. This technique works well when there are a

high percentage of very unfit individuals in a large population, if they are not deliberately removed they may pass on their "weak" genetic traits for many generations.

Elitist Strategy ensures that the maximum fitness value within a population can never reduce from one generation to the next. This can be assured by simply copying the best individual of a population to the next generation, if none of the individual in the next generation constitutes an improvement of the best value. Such idea is deemed necessary when GAs are used as function optimisers and the goal is to find a global optimal solution. But the elitist strategy tends to make the search more exploitative rather than explorative and may not work for problems in which one is required to find multiple optimal solutions.

1.3.3.7 Termination Criteria

It is important to point out the significance of the termination condition used in the GA. The simplest termination condition would be to check the current generation number; the search is terminated if the total number of generations exceeds a predefined constant. This method of termination limits the amount of computational effort used to a predetermined maximum, assuming user's knowledge on the characteristic of the function. However, in many instances it is quite difficult to justify that enough generations (or function evaluations) have been done in the effort to find the global optimal. It would be much better if the GA terminates the search when the chance for a significant improvement is relatively slim. Therefore, a second termination condition would be to terminate the search if improvement over each generation does not exceed some specified threshold.

1.4 Multi-objective GAs

Many engineering problems require a simultaneous optimisation of multiple objectives; it is often difficult to compute a solution with global accuracy for such complex problem. The difficulty stems from the multiple objectivity of the fitness function, which often comes only at the cost of significant knowledge about the search space. However, finding a solution for multiple objectives is not always possible as some of the objectives may be conflicting. Performance, reliability, and cost are typical examples of conflicting, and incomparable objectives. Improvement in any combination of these objectives will result in a better overall solution, but only as long as no degradation occurs in the remaining objectives. If this is not possible, then the solution

is said to be optimal in the Pareto sense, Pareto optimal or non-dominated. In such a problem, the quality of the solution is better described not by a scalar but by a vector quantity.

Despite that multi-objective optimisation with GAs must ultimately be based on a scalar measure of the objectives, though it need not be a function of the objectives. But it should at least indicate an improvement in the fitness of the solution than those that it dominates in the Pareto sense, whilst taking into account of human preferences in the rating of the solutions. Once a scalar measure of the quality has been derived, the GA may proceed with fitness assignment and selection. Listed below are some of the common approaches of multi-objective optimisation [13].

1) *Weighting approaches.* Objectives are numerically combined into a single objective function to be optimised.
2) *Pareto based approaches.* The population is ranked making direct use of the definition of the Pareto dominance.
3) *Constraint approaches.* Objective function is evaluated to see if any constraints are violated. If not, solution is assigned the fitness value corresponding to the objective function evaluation. If the constraints are violated, the solution is infeasible and has no fitness.

1.4.1 Weighting sum method

The method of objective weighting [7], where n objective functions $f_1, \ldots, f_n$ are weighted by user defined positive coefficients $w_1, \ldots, w_n$ such that $\Sigma w_i = 1$ and added together to obtain a scalar measure of the fitness, F for each individual. This measure can then be used as the basic for fitness-based selection. This approach is widely known, intuitive and simple to implement which can be used with virtually any optimisation problem.

$$F(x) = \sum_{i=1}^{n} w_i f_i(x) \qquad (1.1)$$

The setting of the weighting coefficients w_i is generally dependent on the problem at hand and not just on the problem class. Thus the initial combination of weights usually needs to be fine tuned in order to lead to satisfactory compromise solutions. This usually implies rerunning the GA, although it may also be possible to modify the weights as the GA runs.

1.4.2 Pareto based method

The concept of ranking individuals in a population on the basis of Pareto dominance was first suggested by Goldberg [8]. Fig.1.6 shows an example of Pareto dominance method the objective is to minimize two conflicting objects. As shown on the figure there is no single perfect solution that minimizes both f_1 and f_2. Instead there are three compromised optimal solutions A, B and C; solution A has the smallest value of f_1 compared to the rest, solution C has the smallest value of f_2 compared to the rest, while solution B has smaller value of f_1 compared to C and smaller value of f_2 compared to A. All these solutions are Pareto optimal because there is no better solution on both objectives. On the other hand, one can say that solution D is inferior to B because it has larger values than B in both objectives (f_1 and f_2). The ranking of the solutions is based on the tradeoff information among the different objectives.

As a result all non-dominated solutions (A, B and C) are assigned rank one and temporarily removed from the population. Then the next individuals (D, E and F) are assigned rank two and so forth. Finally, the rank of an individual determines its fitness value.

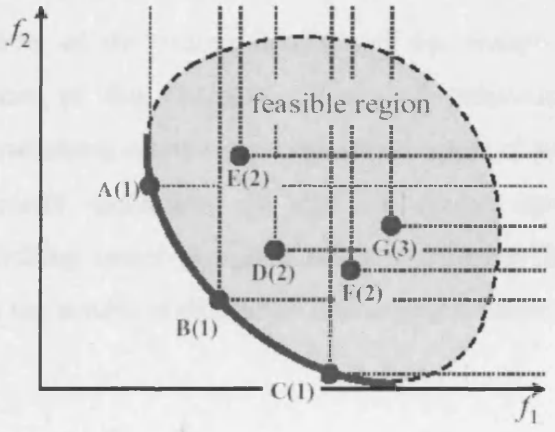


Fig. 1.6 Fitness ranking based on Pareto dominance for a minimization problem

1.4.3 Constraint approach – Penalty method

In a penalty method [10], a constrained problem in optimisation is transformed to an unconstrained problem by associating a cost or penalty with all constraint violations. This penalty is included in the objective function evaluation. For example, a constrained problem in minimisation:

$$\text{minimise } f(x)$$

$$\text{subject to } h_i(x) \geq 0 \quad i = 1, 2, \ldots, n$$

$$f_p(x) = f(x) + r\sum_{i=1}^{n}\Phi[h_i(x)] \qquad (1.2)$$

where Φ is the penalty function and r is the penalty coefficient. There are a number of representations for the penalty function Φ, it can be of the form $\Phi[h_i(x)] = h_i^2(x)$ which means to square the violation of the constraint. While the penalty coefficient r are sized depending on the severity of the violation of the constraints. The penalty function would have a negative sign when applied to a constrained problem in maximization.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces the concepts on the operation and design of the flux switching motor, as well as the implementation of the flux switching motor into a finite element (FE) model. The modelling of flux switching motor drive system in Excel spreadsheet, as a time-stepping circuit along with its switch control algorithms, and the winding optimisation tools is the subject of Chapter 3. Chapter 4 highlights the nature of various electrical and mechanical losses in the flux switching motor. The methods for modelling the various losses will be discussed in brief, while detailed attention is given to the iron loss model. Chapter 5 addresses the breakdown of the coding structure of the design optimisation software written for the purpose of this research. Chapter 6 demonstrates the design and optimisation of flux switching motors for high-speed application using the developed software. For the second application of the GA design optimisation, Chapter 7 introduces a flux switching motor designed using a different fitness function. Lastly Chapter 8 summarises the results of this thesis and suggests future work.

2 Construction, operation & design of the Flux Switching motor

2.1 Introduction

The flux switching motor is a two-phase reluctance machine with the coils pitched over two stator poles. The windings are spread in such way that they cover the same number of stator poles as there are phases; all windings are on the stator, and most commonly the rotor has half the number of poles as the stator. The flux switching motor operates with continuous dc current in one winding, the field winding, and the second winding referred to as the armature winding carries an electronically controlled alternating current, the direction of which is reversed each time a rotor pole passes over a stator pole. This creates the required resultant flux orientation for rotor rotation. The idea of fully pitched windings is originally conceived for use in three-phase switched reluctance motors, and several benefits of using fully pitched windings relative to short pitched windings have been described [14]. Like all switched reluctance motors wound with fully pitched windings, the flux switching motor derives virtually all its torque from a changing mutual inductance between the phases (field and armature) but the method of controlling the magnetic fields is different in that only the armature is controlled. In the lowest cost implementation of the electronic controller the armature winding comprises two closely coupled windings connected to a half bridge inverter with only two power switches.

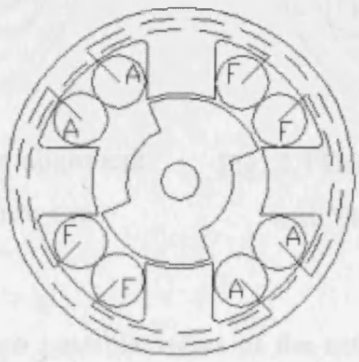


Fig. 2.1 Construction of a 4/2 Flux switching motor

Fig. 2.1 shows an illustration of a 4/2 flux switching motor; it has four stator poles and two rotor poles. For this particular motor construction, a fully pitched winding spreads the whole width of the stator, so that the main active part of the winding is situated in

opposite stator slots. The fully pitched windings are labelled 'F' and 'A' for field and armature windings respectively.

2.2 Operation of the flux switching motor

The principle of operation of the flux switching motor is illustrated with the aid of Figs. 2.2 and 2.3. The 'o' and '+' used in the diagrams show the direction of current flow in the windings, while the dashed lines show the magnetic flux path. As with all reluctance machines, the torque of flux switching motor is developed by the tendency of the rotor to align itself into a position of minimum reluctance with the stator, when the phase windings are energised. The interaction between the two magnetic fields produced by the currents in each of the two-phase windings creates a combined resultant flux. The resultant flux is then oriented into one of the two axes formed by alternate sets of stator poles. The axis selected is dependent upon the direction of the current in the armature winding, which is labelled as 'A' in Figs. 2.2 and 2.3. As mentioned earlier, the current in the field winding labelled as 'F' in the diagrams, remains unidirectional.

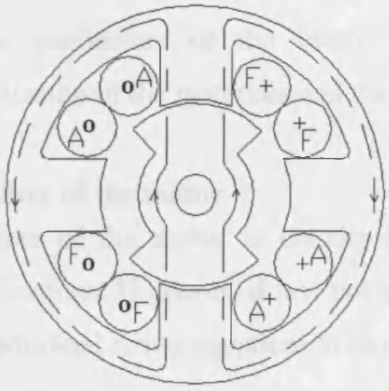


Fig. 2.2 Diagram on the rotor alignment with Field +ve & Armature +ve

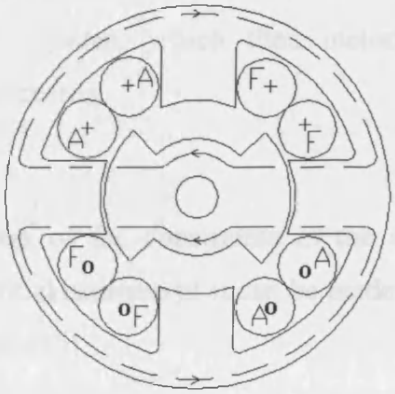


Fig. 2.3 Diagram of the rotor alignment with Field +ve & Armature -ve

Figs. 2.2 and 2.3 show the two possible views of the rotor alignment to the stator. In Fig. 2.2, the field winding creates a flux vector in a north-westly direction. The armature winding excited with a positive current creates a flux vector in the north-eastly direction. As a result the combined resultant flux travels in a northerly vertical direction. In Fig. 2.3, the field winding remains excited in the same direction, meaning that the associated flux vector is still in the same direction. But the current in the armature

winding is reversed, so as a result the associated flux vector is in a south-westly direction (it is 180° changed in direction). The combined resultant flux travels in a westerly horizontal direction. Therefore, the reversal of the current in the armature winding can be thought of as switching the orientation of the flux from one set of stator poles (every alternate stator pole) to the other set of stator poles, hence the name flux switching motor. While rotor rotation at required speed can be realised by sequential commutation of the armature at the correct time with respect to the position of the rotor.

2.3 Design of the flux switching motor

In the past, lamination design was primarily an electro-mechanical consideration. Today, improved performance and efficiency is increasingly important; however, so is reliable, minimal cost stator assembly. Design of the flux switching motor is a process of making choices, which matches the specifications, subject to a series of constraints, imposed by laws of physics, manufacturing processes, economics, end application of the design etc. The most basic requirements in a motor specification would usually include the overall motor dimension, its motoring torque, operational speed, and supply voltage. While the design process aiming to achieve the specifications consists of defining the design parameters of the motor and its drive system, which then include the specification of the materials and manufacturing processes.

2.3.1 Size of the motor

The size of the motor is usually included as one of the constraints in the design specifications. However, if it is not specified, an initial estimate of it can be made using the traditional sizing equation, in terms of the torque [15]

$$T_{em} = C_o D_g^2 L_{stk} \qquad (2.1)$$

where C_o is the output coefficient, D_g is the airgap diameter and L_{stk} is the stack length. As the flux switching motor technology is still a considerably new concept, the values of C_o could perhaps be referenced from the more established technologies liked the switched reluctance motor and the induction motor. The typical values of C_o, used in small totally enclosed motors are 1.96 – 5.5 kNm/m^3, and 5.5-23.5 kNm/m^3 for integral-hp industrial motors, where experiments conducted to verify the value of C_o showed that the winding temperature rise is higher in the switched reluctance motor, 90°C and 68°C in the high-efficiency induction motor [15].

2.3.1.1 Ratio of the rotor to stator diameter

Using constructed flux switching motors [1,16-19] as a reference, the typical ratio of the rotor to stator diameter used in the lamination designs are about 0.5 - 0.6. The ratio depends on several factors, the number of the stator and rotor poles, the stack length, the motor frame size and also on the requirements of the operating speed and torque. The torque that can be produced for a given motor design is proportional to the tangential portion of the magnetic flux flowing across the motor airgap, and to the radius at which forces due to this flux are produced. While the flux that can flow across the airgap is ultimately limited by the cross section area of rotor pole which is proportional to the rotor radius, and saturation flux density of the lamination steel used. In order to develop higher torque, either the radius or axial length of the airgap must be increased. However, large rotor radius restricts the space available for windings, also its size is limited by the motor frame size. Increased motor length requires extra number of laminations, maximum stack length can be limited by capacity of the presses used in motor manufacture, or by rigidity of the rotor and stator assemblies between which tight tolerances on radial clearance must be maintained.

2.3.2 Pole configuration of flux switching motor

A regular flux switching motor is one in which rotor has half the number of poles as the stator, and is a two-phase motor. With the regular flux switching motor the choice of pole configuration is always in the arrangement of 4/2 multiplied by n, where n is an integer (1, 2, 3... n), this leads to several possible pole arrangements 4/2, 8/4, 12/6, 16/8 etc. The appropriate pole configuration for a specified motor frame size can be decided by analysing the torque equations of the flux switching motor and conventional dc motor. The electromagnetic torque of the flux switching motor [1] can be express as:

$$T_{em} = \frac{2N_r k}{\pi \Re} N_f i_f N_a i_a \qquad (2.2)$$

where N_r is the number of rotor poles, k is the fraction of field flux linking the armature winding, $\Re$ is the reluctance of the magnetic circuit, N_f is the number of field turns, N_a is the number of armature turns, i_f and i_a is field current and armature current respectively.

The general torque equation for a dc motor, for each phase is

$$T_{em} = \frac{P_{em}}{\omega} = ei\frac{\partial t}{\partial \theta} = i\frac{\partial \lambda}{\partial \theta} = Ni\frac{\partial \phi}{\partial \theta} \qquad (2.3)$$

where P_{em} is the phase power, ω is the mechanical angular speed, e is the back EMF, i is the operating current, λ is the flux linkage per pole pair, ϕ is the flux, and N is the equivalent number of turns.

The advantage of having larger number of stator poles and rotor poles, N_r is a smaller stroke angle θ, producing a higher torque. Though the stator pole arc as well as rotor pole arc has to be reduced, meaning a reduction of field flux linking the armature. Nonetheless, reduced pole arc could increase the total slot areas which can then be utilised for reducing copper losses. An 8/4 FSM has twice as many poles as the 4/2 but a high fraction of field flux linking the armature winding can be maintained with proper pole design, and the end winding are also shorter as it pitched over a shorter distance, which is beneficial for reducing copper losses. It has shorter flux paths in the stator back iron when compared to the 4/2 motor, short flux paths suggest the magnetic circuit has lower reluctance.

Disadvantages of having too many rotor and stator poles is that the motor structure can become mechanically weak and reduction of coupling coefficient between armature and field because of increase in fringing flux and leakage flux. While to maintain similar operation speed of a motor with more rotor and stator poles would require a higher switching frequency, this higher switching frequency would result in higher iron loss in the motor. Thus it is not worth considering large number of poles unless the motor's radial dimension is increased significantly.

2.3.3 Internal design parameters of flux switching motor

In the flux switching motor, the changing of mutual inductance between the two phases in the stator creates a combined resultant flux, the resultant flux tends to pull the rotor poles toward an aligned position with the energised set of stator poles, thereby applying a torque to the rotor whose profile is itself dependent upon several factors, including the airgap length, the number of the stator poles, the geometry of the stator pole, and the manner in which the stator phases are energised.

2.3.3.1 Airgap length

In the flux switching motor, the radial length of the airgap between stator and rotor is usually made as small as possible. The overall size of the motor, the manufacturing process, the tolerance on the assembly, and the motor operating speed determine what the smallest length can be to ensure that the rotor does not rub against the stator. The advantage for smaller airgap is a lower reluctance in the magnetic circuit as it contributes to the largest portion of the total circuit reluctance, also higher torque could be achieved with the same amount of mmf in the field winding, however building a motor with smaller airgap inevitably increases the manufacturing cost.

The typical airgap length used in construction of flux switching motor ranges from 0.3mm to 0.6mm. Though there is not any established rule of thumb to guide the selection of airgap length but one could use finite element analysis to derive the flux-linkage characteristics of the motor and estimated performances of the motor can be predicted using electromechanical torque equations of a dc motor.

The operation of the flux switching motor is less dependent on airgap length than the switched reluctance machine [19]. The motor is not operating on a change in self inductance but on the switching of dc flux between two sets of stator teeth. As a result an increase in airgap length causes an increase in the mmf required in the field winding to produce a given airgap flux. This is relatively linear as opposed to the more rapid de-rating of a switched reluctance motor as the airgap increases.

2.3.3.2 Pole to slot geometry

The shape of the stator pole and its slot are a compromise between several conflicting requirements. Fig. 2.4(a) illustrates a basic concept of the pole to slot construction of an 8/4 FSM, the problem with the simple design is that as the stator pole arc increases the slot area reduces, this conflicted with the need to maximise the slot area to decrease the copper losses. The concept of the sizing of pole arcs would be discussed on the next section. However, several improvements have been suggested in a reference on the design of the pole to slot geometry of the switched reluctance motor [15]. The improvements to the simple design of pole to slot layout can be illustrated using fig. 2.4(a) & (b). Referring to fig. 2.4(b), the first improvement can be made by filleting the corners and edges in the lamination construction, especially the joints between the poles and the stator back iron; this stiffens the pole against lateral deflection which reduces the mechanical vibration.

Next is the inclusion of the shoulders or overhangs at the pole tips, physically to provide a register for slot wedges. The inclusion of pole shoulders had benefiting effects on softening the torque impulse that may occur at the start of the overlap and also to reduce saturation at the pole tips. Saturation in the pole tips can make the slot opening effectively larger than the physical opening; this reduces the effective area of the stator and rotor poles facing each other, with the reduced area the reluctance of the magnetic path increases, thus resulting in a drop in airgap flux. Another benefit of including the pole shoulder is by undercutting the stator pole the slot area is slightly increased while still maintaining a wide stator pole arc. However, this design will not be suitable if pre-wound windings are to be inserted after the laminations have been stacked.

(a) A simple concept of the construction of an 8/4 FSM

(b) The improvements made to design (a)

Fig. 2.4 (a) & (b) Examples of pole to slot construction of an 8/4 FSM

The pole to slot geometry has a strong influence on the overall stiffness of the mechanical structure of the stator. The choice of having a construction with thick stator back iron and stator pole illustrated in fig. 2.4(a), helps to strengthen the stator structure which is necessary in reducing mechanical vibration, but it compromises on the slot area. However, if slot area is made larger by reducing the thickness of the stator back iron and stator pole; the result is an increase of peak flux density value in the stator leading to higher iron losses.

The flux switching motor is a two-phase reluctance machine with the coils pitched over two stator poles; with the field winding operates with continuous dc current, and the armature winding comprises two closely coupled windings and operates with an electronically controlled alternating current. As the two windings are excited in a different manner, perhaps the slot area need not be of similar shape. In the earlier section 2.2, fig. 2.2 and fig.2.3 are used to illustrate the orientation of the flux paths,

apparently the flux travelling in the field back iron is unidirectional while bi-directional flux travels in the armature back iron. Research on the iron losses in the different parts of FSM have shown that armature back iron generates more losses than the field back iron. The details of the findings are presented in Chapter 4. Thus to expand the slot area it would be more sensible to exploit the depth of the field slot, and undercut into the stator poles to increase the armature slot area without over-compromising on the stator pole thickness. A hybrid design illustrated as fig. 2.5 [19], combining each half of designs featured in fig. 2.4 (a) & (b), could perhaps be a good starting point of a compromised solution. As a matter of fact, the layout between the stator pole shape, the slot area and the thickness of the back iron is a compromise solution, depending on the entire torque/speed range and the number of poles.

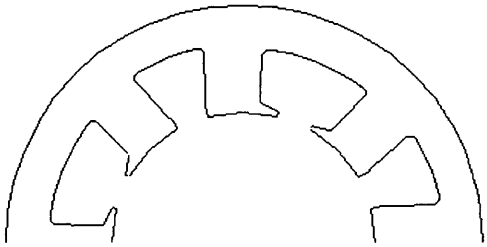


Fig. 2.5 A hybrid design, combining the features from two different stator laminations

2.3.3.3 Ratio between stator and rotor pole arc

In flux switching motor, the optimum pole arcs should be as large as possible to maximise aligned inductance and flux linkage without resulting in the increase of leakage and fringing flux, which causes field flux linking the armature winding to drop thus affecting the development of torque. Torque begins to be developed just when rotor pole corner is about to meet the next stator pole corner, and continues until the poles are fully aligned.

Fig. 2.6(a-c) are used as a guidance on the design of the pole arcs. Fig. 2.6(a) illustrates an 8/4 FSM with its rotor in the aligned position relative to the stator pole, at this point direction of the current is reversed so that the next set of stator poles would become energised and pull the rotor poles toward the newly energised set of stator poles. Noticeably there is still some distance between the rotor pole corner and the next stator pole corner as the current is about to change its direction, at such rotor position the motor would normally produce zero torque. Though the problem could be alleviated by slightly increasing the rotor poles arc, as shown in fig. 2.6 (b-c). However, the example shown below may not be appropriate for all applications, the choices on the poles arc

depend on the entire torque-speed range, the pole configuration and the switching time during armature current reversal.

(a) an 8/4 FSM in aligned position

(b) the lamination design of the original rotor used in fig. 2.6 (a)

(c) a improved rotor design of FSM with a wider rotor arc

Fig. 2.6 (a-c) Example of an 8/4 FSM on the optimum ratio between the pole arcs

2.3.3.4 Rotor profiling + Asymmetry rotor

The main idea of employing an asymmetrical rotor in the flux switching motor is to overcome the starting problem where there are positions of zero torque which is a typical problem for the flux switching motor. Fig. 2.7 illustrates an irregular rotor construction, the rotor poles are asymmetric about their centre-line. With the asymmetric design, the rotor has the advantage of starting assist because of the unbalanced magnetic pull acting on the overlapping corner with a graded airgap on the leading edge of the rotor, the unbalanced magnetic force is a result of the uneven airgap.

While the other advantage of having an asymmetrical rotor in the motor is to simplify the implementation of sensorless commutation control of the FSM. In a FSM with symmetrical rotor, it is very difficult to determine the true rotor displacement, as there are two possible rotor positions (it maybe clockwise or anticlockwise) that shared the same electromagnetic characteristics (like inductances). The introduction of asymmetry rotor in the FSM allows the sensorless control technique to accurately measure the differences in the waveforms generated and thus determine between two potentially ambiguous positions, however the degree of asymmetry must be significant to ensure that the accuracy of the deduction of the rotor position can be achieved.

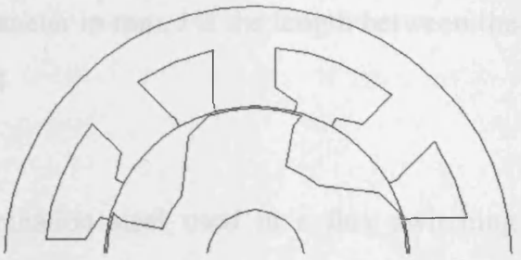


Fig. 2.7 An 8/4 FSM with an asymmetric rotor

Other key features of rotor profiling include varying of the depth of the rotor slot, varying thickness of the rotor core and determining the shaft diameter. These features are similar to the "sizing" process of the switched reluctance motor as both have salient poles on the rotor. The rotor featured in fig. 2.7 is profiled to rotate in the counter-clockwise direction. The rotor slot has deep and steep undercut at the trailing side, this is to reduce fringing flux from unaligned stator pole to enter the rotor pole thus reducing the magnetic pull against the rotating direction. While the overlapping corner at the rotor leading edge with a graded airgap creates unbalanced forces acting on rotor which is useful for starting the motor.

The value for thickness of the rotor core should be a balance between the shaft diameter and rotor slot depth, it needs to be sufficiently thick to carry the rotor flux without saturating. The thickness also needs to be varied relative to the number of rotor poles and rotor diameter, because as the number of rotor poles increases the rotor core surface area decreases leading to an increase in rotor core reluctance, which may result in core saturation. The rotor core thickness should be at least half the thickness of the rotor pole.

In design of the rotor shaft, large shaft diameter is desirable to maximise its radial stiffness, and also minimise acoustic noise. However, having too large a shaft diameter may compromise on the thickness of the rotor core, resulting in core saturation. Thus, it is sufficient just to have shaft diameter large enough to withstand the radial and axial force acting on the rotor at the operating speed. The minimum required shaft diameter could be calculated using the equation of the first critical speed [15]

$$n_c = 203.078\times10^3 \frac{d^2}{l\sqrt{W_r l}} \qquad [rpm] \qquad (2.4)$$

where d is the shaft diameter in mm, l is the length between the bearing in mm, and W_r is the rotor weight in kg.

2.3.4 Lamination steel

The choice of the lamination steel used in a flux switching motor depends on its switching frequency. For high-speed applications low loss steel would be required to achieve high efficiency, while for low speed high power density applications a highly permeable steel is better. The choice of the steel determines the amount of iron losses expected in the motor, though if the problem is carefully addressed in the design of the motor lamination the iron losses can be reduced. A more complete discussion on choice of lamination steel and estimation of its losses is found in Chapter 4.

2.3.5 Winding design

The selection of the winding configuration, e.g. number of turns per coil, series or parallel connection between coils, wire diameter, is based on several factors; the size of the slot and the packing factor determines the amount of copper that can be packed in the slot. While the load torque, voltage and speed control requirement would determine the number of winding turns and how it should be connected. Like dc machines the field of the flux switching motor can be wound in either shunt or series to the armature circuit. Regardless of the winding configuration used inside the FSM, the winding terminals must be brought out and be connected to the drive electronics.

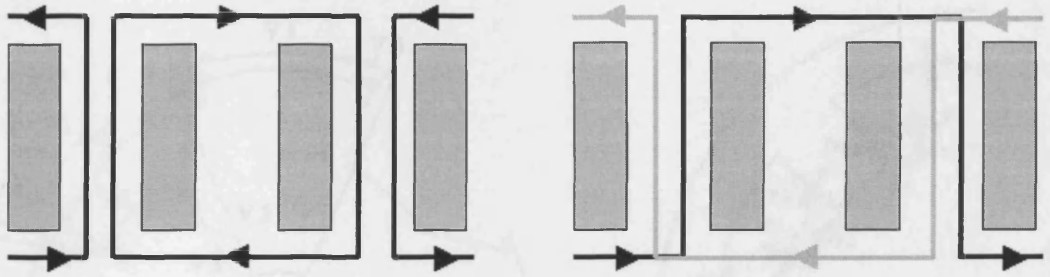
2.3.5.1 Shunt wound or series wound?

The Flux switching motor field winding can be connected in series or in shunt with the electronically controlled armature windings. The studies on the comparison of the results of an identical motor lamination stack, one with a shunt winding and one with a series field winding were made in reference [19], where identical mechanical arrangements were used for the shunt and series stators so that comparisons could be made between both the motors in the same type of housing. In both series and shunt configurations, the armature windings comprise a set of closely coupled bifilar coils that are simply connected to two ground referenced electronic switches. However, in shunt configuration the field winding requires a higher number of turns and an extra ground referenced power mosfet for the control of the field current. The mosfet provides the flexibility for field weakening and also allows the field current to be reduced at light loads to reduce the field losses. The number of turns for the armature windings were

kept the same for both configurations. The results from the experiment in reference [19] demonstrated that using either series or shunt configuration in the flux switching motor could produce identical performance.

2.3.5.2 Winding techniques

Two winding patterns for flux switching motor are illustrated in fig. 2.8 (a) & (b), Option 1 uses multiple coils which requires internal connection to form the complete winding, while option 2 uses two single large coils to form the winding. This method of winding has the advantage of using an external wound coil which is later inserted into the slots.



(a) Option 1, multiple coils to be internally connected in series/parallel

(b) Option 2, two single coils forming a complete winding

Fig. 2.8 (a) & (b) Winding patterns for one winding of the flux switching motor

2.4 Finite element method in modelling the FSM

Electromagnetic modelling using finite element methods is well established for solving problems beyond the scope of analytical methods. By using the finite element method, the FSM with complicated geometries incorporating non-linear magnetic properties of the materials, and arbitrary shaped excitation waveforms can be modelled with good accuracy. Such methods can be used in conjunction with an analytical design process. Electric circuit and design equations could be used to produce the initial design, while the predicted performance could be used to further improve design development. However, there are numerous finite element analysis (FEA) packages available in the market, in which they mainly differ in the complication of constructing the model, the reusability of the constructed model, and the speed at which results can be obtained.

2.4.1 Parameterised finite element analysis of the FSM

Electromagnetic finite element modelling software can simulate the performance of an electromagnetic device, given a full geometric specification and data on materials and excitation. The entry of such data can be greatly simplified through the creation of a parameterised finite element system or "Design Environment" [20]. The complete finite element model is then created though algebraic equations that establish the structure of the whole mesh from the design parameters; the key dimensions and normalized ratios. This parameterised FEA offers significant benefits over a manual design process, but more importantly creates the opportunity for optimisation software to vary the parameters in a structured way to deliver improved performance.

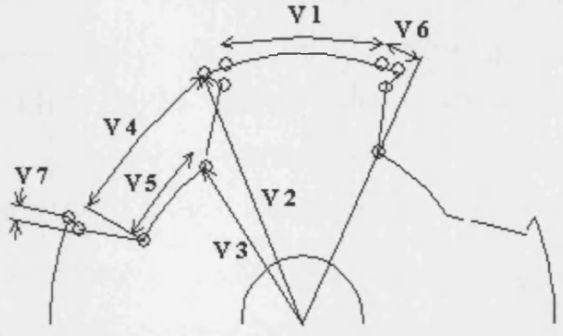


Fig. 2.9 Design parameters controlling the rotor shape

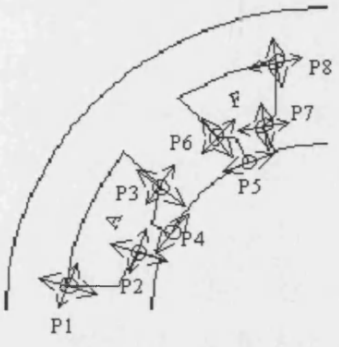


Fig. 2.10 The design variables in the FSM stator

Fig. 2.9 illustrates the design parameters that can be modified on a reluctance machine rotor in addition to the number of poles themselves. Providing the variables V1-V7 are expressed as a percentage of the pole pitches or outer radii as appropriate, the design can be changed to any size and any pole number and the same design proportions will be retained.

Fig. 2.10 illustrates the design parameters that define the shape of the stator lamination and the slots containing the armature and field windings. Note that since the electromagnetic purpose of the field and armature windings are quite different it is beneficial for the field and armature slots to have different shapes. As with the rotor, the design parameters, P1-P8 are expressed as percentage of the pole pitches or outer stator radial depth as appropriate.

This parameterised design entry allows a genetic algorithm control program to have normalized ranges for each parameter. Furthermore, this approach makes the whole

design process very rugged as combinations of inappropriate dimensions are prevented at the outset. The extraction of results from the parameterised finite element model is also controlled by the same parameters. For example if the number of poles or the radial position of the air-gap are altered, the position of any integrals taking place in the air-gap over a pole pitch is altered automatically.

3 Dynamic drive system simulation

3.1 Introduction

When optimising designs of electric machines, tools that enable one to calculate the magnetic field distribution are especially important, providing reliable information about the machine saturation subjected to changes made to its design parameters. The static finite element analysis (FEA) is one of the tools that can accurately calculate the time invariant electromagnetic field solutions of any given lamination geometry. However, it does not perform dynamic analysis of the FS motor. In order to perform dynamic time domain simulations, a time stepping finite element analysis coupled with an electric circuit and mechanical system model would be required. Although the time stepping FEA is capable of accounting for eddy currents and dynamic effects of the motor, and the results would be highly accurate the length of computation time was considered unacceptable, especially for the numerous and varied cases to be considered when incorporated with the genetic algorithms.

As a result, an alternative modelling approach is devised to simulate the dynamic motoring performance of the flux switching motor. The proposed dynamic drive system model is integrated with an electromagnetic finite element modelling software (OPERA-2d). The system model uses initially the static FEA calculated results of winding flux linkages to predict the motor armature back-emf, the variation in the armature back-emf with rotor position was then used in an electrical equivalent circuit model [18] to estimate the field and armature current from a given voltage excitation waveform. The results of the estimated field and armature current relative to rotor position were then fed back to the static FEA to obtain the data of localised flux density distribution in the lamination. The results were then returned to an iron loss model (more details in chapter 4) built within the dynamic system model to predict total iron losses in the motor. In addition the system model incorporated also models to optimise the firing angle which allows advanced, retard and time-delay to be introduced in the control of the power switches, and optimise windings design to meet user-defined motoring specifications. Using the consolidated data from various models, the performance of the motor can thus be estimated, and the developed drive system model is a promising method providing short simulation time and adequate accuracy during initial design stages. The drive system model presented in this chapter describes the drive system for the FSM with field winding connected in series with the armature

circuit, as this has been found to be the better configuration in terms of the power output capability and drive cost [1,19].

3.2 A time-stepping electrical equivalent circuit model

A time-stepping electrical equivalent circuit model [18] has been developed based on circuit diagram shown in fig. 3.1 for a Flux Switching motor. The field winding is shown in series with the incoming supply current with a diode to allow a path for the field current to freewheel while the armature switching occurs. The bi-directional current in the armature slots is achieved by alternate energisation of the two mosfets connected to two oppositely connected, closely coupled, armature windings.

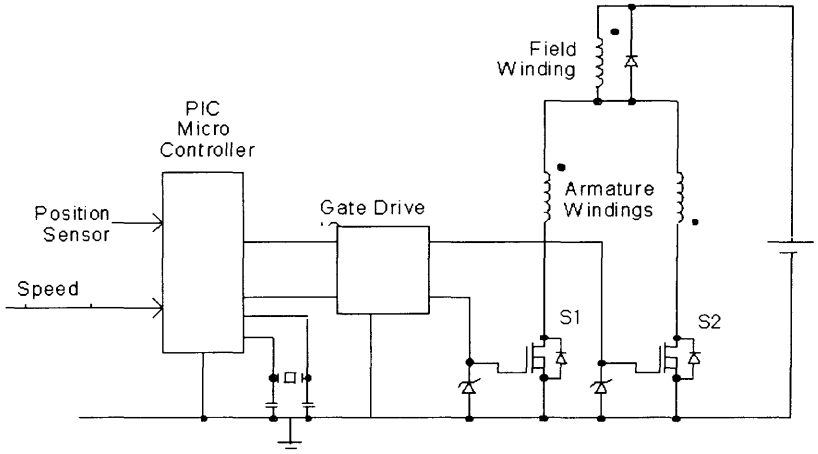


Fig. 3.1 The drive electronics for the flux switching motor used in the simulation and optimization loop

The dynamic drive system model receives the results of winding flux linkages calculated by the FEA software while only the field winding is energised at a range of rotor positions. Examples of the variation in the flux linkage associated with the field and armature windings are shown in Fig. 3.2 and 3.3 over a complete electrical cycle of rotor rotation. Whilst the field flux linkage does not vary substantially with rotor position (Fig. 3.2), the armature flux linkage varies strongly with position. It is the variation in armature flux linkage with rotor position that produces the armature back-emf. The illustrated flux linkage plots represent the electromagnetic characteristics of a prototype flux switching motor (chapter 6). The flux linkage waveforms at other field current levels can either be calculated by either FEA or can be extrapolated along a known non-linear function to represent the machine saturation. It is the later method that has been employed in the drive system model that facilitated the calculation of the armature back-emf and details of the method will be discussed in the following section.

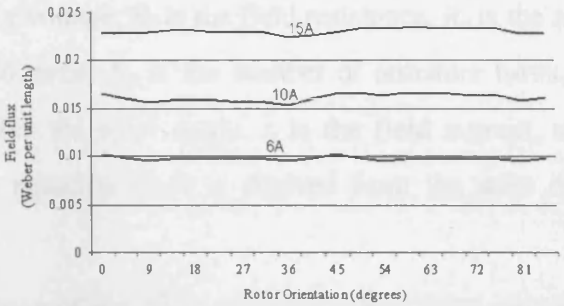


Fig. 3.2 Variation of field flux with rotor position at different field current levels.

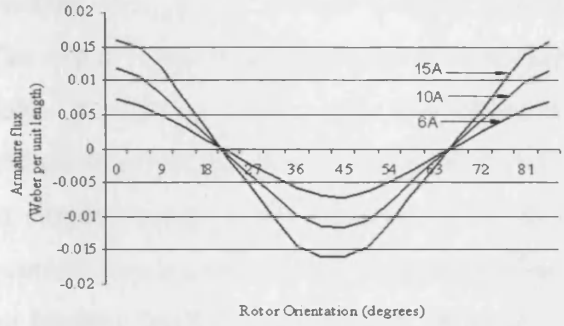


Fig. 3.3 Variation of armature flux with rotor position at different field current levels.

The variation in the armature back emf with rotor position was used with winding inductances and resistances to estimate the field and armature current from one time-step to the next. Equations (3.1-3.5) are the general electrical circuit modelling equations which are solved simultaneously at each time step.

$$V_s = V_f + V_a \tag{3.1}$$

$$V_f = i_f R_f + N_f \frac{d\phi_f}{dt} \tag{3.2}$$

$$\frac{d\phi_f}{dt} = \left(\frac{d\phi_f}{di_a} \cdot \frac{di_a}{dt} + \frac{d\phi_f}{di_f} \cdot \frac{di_f}{dt} + \frac{d\phi_f}{d\theta} \cdot \frac{d\theta}{dt} \right) \tag{3.3}$$

$$V_a = \pm \left(i_a R_a + N_a \frac{d\phi_a}{dt} \right) \tag{3.4}$$

$$\frac{d\phi_a}{dt} = \left(\frac{d\phi_a}{di_a} \cdot \frac{di_a}{dt} + \frac{d\phi_a}{di_f} \cdot \frac{di_f}{dt} + \frac{d\phi_a}{d\theta} \cdot \frac{d\theta}{dt} \right) \tag{3.5}$$

Where V_s is the supply voltage, R_f is the field resistance, R_a is the armature resistance, N_f is the number of field turns, N_a is the number of armature turns, ϕ_f is field flux, ϕ_a is armature flux, and θ is the rotor angle, i_f is the field current, and i_a is the armature current. The sign in equation (3.4) is derived from the state of the switches in the inverter.

During normal operation of the flux switching motor drive system, switch S1 is turned on when field current, i_f is non zero. The current flows from the supply to the armature via the series field winding. The current into the inverter is initially less than the field current (Fig. 3.4a). The excess field current flows in the diode and the diode clamps the voltage across the field. In order to allow faster simulation in the early stages of a design process the field winding internal emf can be ignored and the field current during the free-wheel time is simply represented by equation (3.6). As i_a increases to become equal to i_f there is no current flowing through the field diode (Fig. 3.4b). As a result, the field voltage, V_f is no longer clamped and i_f is then equal to i_a as given by equation (3.7),

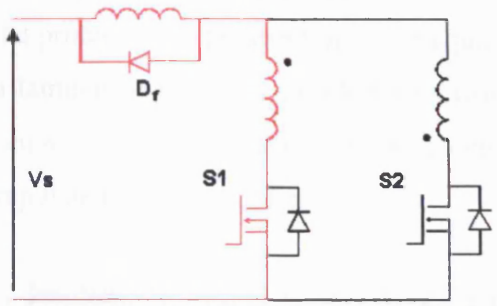
$$i_{f_n} = i_{f_{n-1}} - \left(i_{f_{n-1}} R_f \cdot \frac{\Delta t}{L} \right) \tag{3.6}$$

$$i_{a_n} = i_{a_{n-1}} + \left(\left\{ V_s - i_{f_n} R_f - \left(i_{a_{n-1}} R_a \right) - N_a \frac{d\phi_a}{d\theta} \cdot \frac{d\theta}{dt} \right\} \frac{\Delta t}{L} \right) \tag{3.7}$$

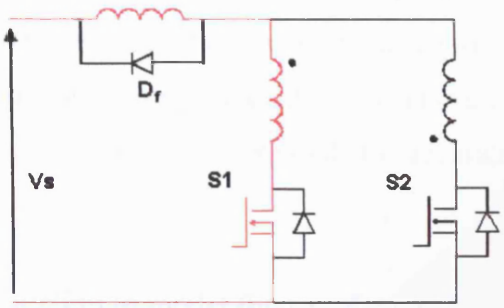
where n denotes the present state, n-1 the previous state, and L is the armature inductance. Winding resistances and inductances are calculated within the circuit model from the specified number of turns and slot sizes obtained from the lamination as drawn in the FEA.

As the point for armature current reversal approaches S1 must be turned off prior to, or at the same time as the second switch S2 is turn on. The smaller the torque load, the earlier S1 is turned off prior to turn on of S2. When S1 is turned off the stored magnetic energy transfers to the second armature winding (Fig. 3.4c), the current now flowing in the second winding is negative as it is wound in the opposite direction to the first armature winding. This current reduces to zero as it flows in the diode parallel to power switch S2 and returned to the supply via the field diode, D_f. When S2 is on, i_a is negative so is the voltage across armature. The reversed in current provide the reverse mmf allowing the rotor to switch to the next pole. Similarly as magnitude of $-i_a$

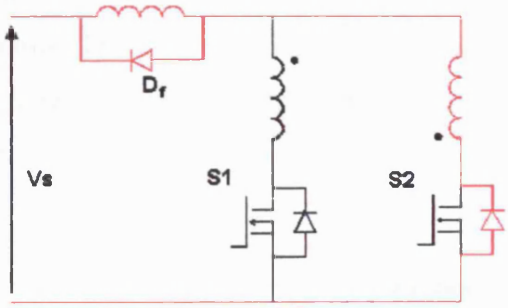
increases to become equal to the field current (Fig. 3.4d), there is no current in the diode parallel to the field and the process repeats to allow the motor to continue to rotate. Thus, the polarity of the armature current flowing in the bifilar winding in the circuit depends on both the state of the switches at that time, and the micro controller controls the state of the switches where at no instances are two switches allowed to be turned on at the same time.



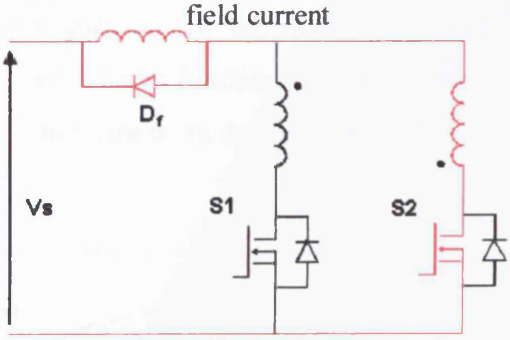
(a) Switch S1 on, positive i_a is less than i_f



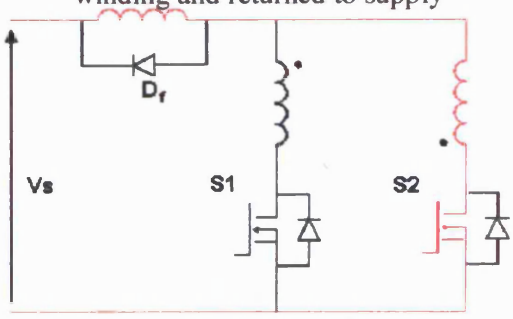
(b) Switch S1 on, positive i_a is equal to field current



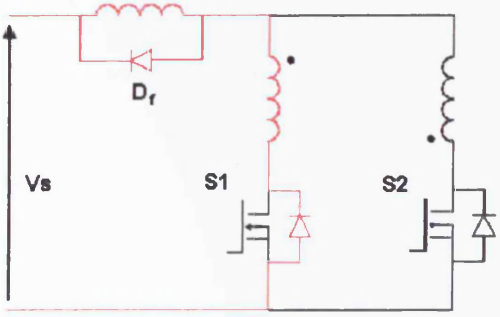
(c) Switch S1 off, i_a transferred to next winding and returned to supply



(d) Switch S2 on, negative i_a is less than i_f



(e) Switch S2 on, negative i_a is equal to field current



(f) Switch S2 off, i_a transferred to next winding and returned to supply

Fig. 3.4(a-f) Operation of flux switching motor drive system in normal operation

3.3 Modelling of the flux-mmf curve

The proposed drive system model uses winding flux linkages vs rotor position waveforms at different field current levels to predict the armature back-emf and other related quantities to estimate the dynamic performances of the FSM. However, this would require multiple solutions to be calculated using FEA and the process requires long computational time yet not all the generated solutions would be used. In the light of the problem, the prospect of using equations to represent the flux-mmf characteristics of a lamination shape in simple terms would be more computationally economical and effective so only minimal FEA computed solutions are required to accurately extrapolate the required data.

3.3.1 Implementation of a piecewise linear function to model flux-mmf curve

In order to model the flux-mmf curve, it is necessary to separate the curve into piecewise-linear parts such that each piecewise part can be represented by a linear function. To simplify the calculations the flux-mmf curve function is divided into two parts, each represented by a linear model and a non-linear model (as illustrated in Fig. 3.5).

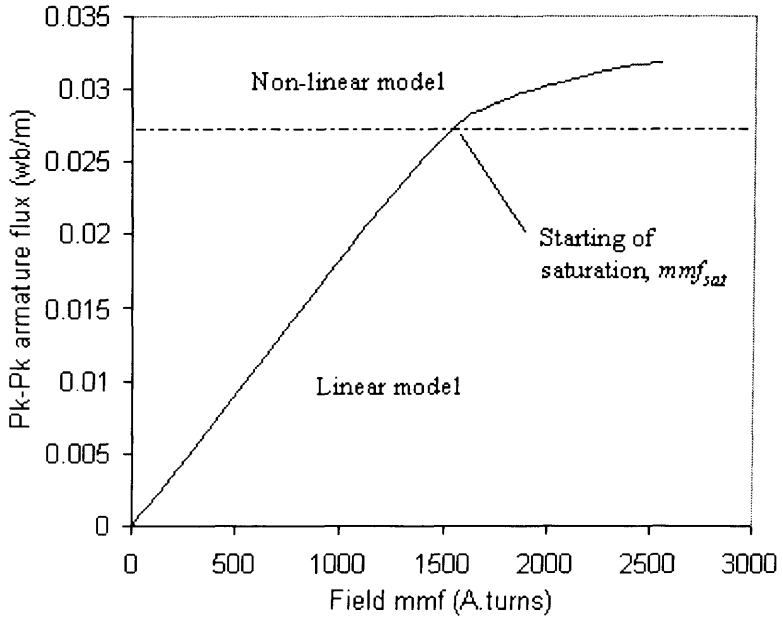


Fig. 3.5 Illustration on the modelling of the flux-mmf curve

In the linear model, the flux-mmf expression can simply be represented as a straight line as in the form of equation (3.8), where ϕ_{actual} is the actual field or armature flux linking its respective windings, mmf_{actual} is the actual field mmf applied during operation, ϕ_{linear}

is the (1st set of) FEA pre-calculated field or armature winding flux linkages in the linear field mmf limit, mmf_{linear} is the field mmf applied in the FEA calculation (set at 1A × N number of winding turns), and the exponent, k_1 has the value of 1 for most grades of lamination steel. In the non-linear model, the flux-mmf curve can be represented using equation (3.9) for the region above the dashed line as illustrated in Fig. 3.5. Noticeably in the non-linear model an mmf dependent exponent was introduced, the exponent, x takes the expression of equation (3.10). The exponent describes the saturation of the machine as the field mmf increases and the rate of saturation is being defined by two terms; in the first term, mmf_{sat} means the starting of the saturation where the (field and armature) flux does not increase in equal proportion to the increase of the field mmf (this is also the starting point of the non-linear model). The second term, k_2 describes the level of the saturation. Both the terms can be determined by curve fitting equations (3.8 & 3.9) against the flux-mmf curve (fig. 3.6).

Linear model,

$$\phi_{actual} = \phi_{linear}\left(\frac{mmf_{actual}}{mmf_{linear}}\right)^{k_1} \tag{3.8}$$

Non-linear model,

$$\phi_{actual} = \phi_{normalise}\left(\frac{mmf_{actual}}{mmf_{non-linear}}\right)^{x} \tag{3.9}$$

$$x = \frac{1}{\left[1+\left\{k_2(mmf_{actual} - mmf_{sat})\right\}\right]} \tag{3.10}$$

Normalisation of non-linear flux,

$$\phi_{normalise} = \phi_{non-linear} \div \left(\frac{mmf_{non-linear}}{mmf_{linear}}\right)^{x} \tag{3.11}$$

Equation (3.11) describes the normalisation of FEA calculated flux linking either the field or armature winding, $\phi_{non\text{-}linear}$ is the (2nd set of) FEA pre-calculated field or armature winding flux linkage where the applied field mmf exceeds the linear model limit (presented by the dashed line in fig. 3.5). The $mmf_{non\text{-}linear}$ is the field mmf applied in the FEA for calculating $\phi_{non\text{-}linear}$, the value of the field mmf has to be in the outside linear model limit (where the limit is determined by machine size, shape and material

used), in this case the value is set at 2600A.turns. To derive the actual (winding) flux linkages when the machine is saturated the non-linear model (eq. 3.9) would be used, it is described as a function of the normalised non-linear flux-linkage, $\phi_{normalise}$ multiplied by a non-linear gain factor.

3.3.2 Determining the constants of the non-linear model

To determine the two saturation constants used in equation (3.10), firstly one have to get a peak-to-peak flux linkage against field mmf plot of a machine lamination. The plot can quickly be obtained using FEA, as only two aligned rotor positions (eg. in a 8/4 configuration, at 45° and 90°) for each chosen mmf range need to be calculated. After the obtaining the flux-mmf plot, the first saturation constant, mmf_{sat} can be determined by curve fitting equation (3.8) against the FEA generated data, and the value of the mmf_{sat} is confirmed at the point when the solution calculated by the linear model starts to diverge from the (FEA) simulated data (where the winding flux linkages do not increase linearly anymore). While the second constant, k_2 can be derived using a least squares fit function, where the non-linear model (with the substituted mmf_{sat} value) is fitted against the FEA simulated data. An example of determination of the two non-linear model constants, k_2 and mmf_{sat} is shown in fig. 3.6, and the determined values of the two constants for this particular example are 1.61×10^{-4} and 1568 respectively.

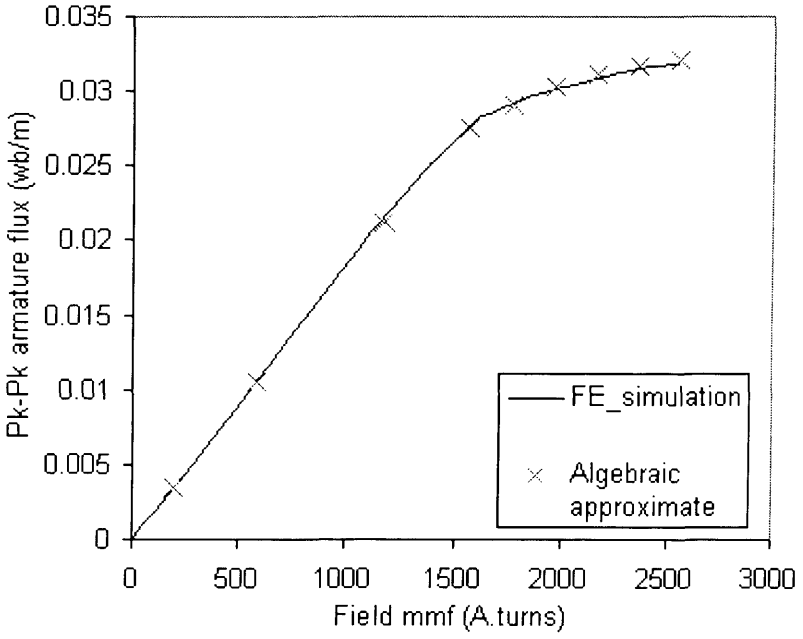


Fig. 3.6 Determination of the non-linear function constants, k_2 and mmf_{sat}

The proposed model of defining the machine flux-mmf characteristics then requires two complete sets of flux linkage vs rotor position waveforms (each represented with 20 discrete points) to be calculated by the FEA; the first one is calculated with "linear" field mmf and the second is calculated with "non-linear" field mmf. While the other waveforms can be extrapolated using the linear and non-linear model and the results of extrapolated waveforms tied up very well with the FEA calculated waveforms (fig. 3.7 & fig. 3.8).

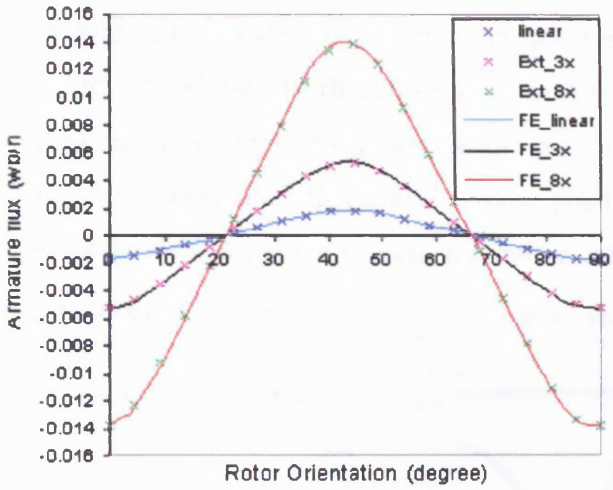


Fig. 3.7 Comparison of extrapolated results using linear section of the model and FEA calculated results.

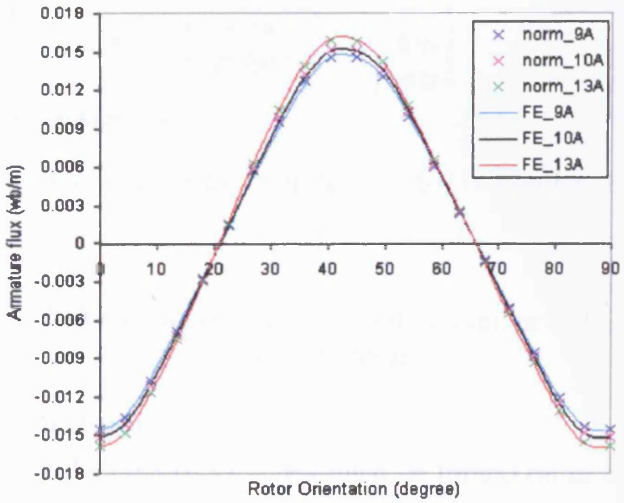
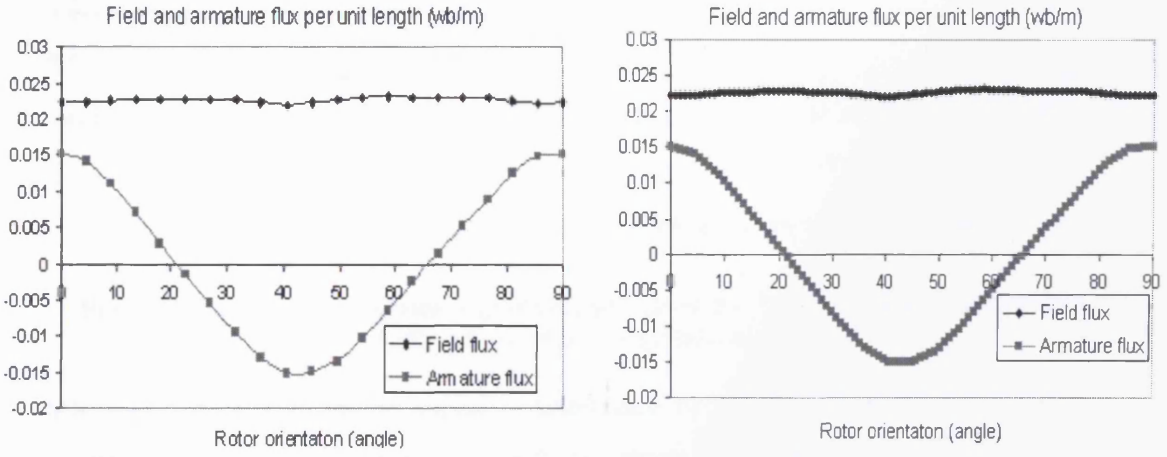


Fig. 3.8 Comparison of extrapolated results using non-linear section of the model and FEA calculated results.

3.3.3 Interpolation of flux-linkage vs. rotor angle

As the number of FEA solutions generated is kept to the minimum to allow fast computation, the number of available data points of flux-linkage vs rotor position is relatively limited (fig. 3.9a). The armature back-emf derived by differentiation of a waveform with only 20 data points could result in numerical error. In order to minimise the numerical error associated with differentiation more data points are required and this can be achieved by taking interpolation between the 20 available data points. Linear interpolation is employed in the model, as it is the simplest type and computationally the least demanding. Fig. 3.9b shows the flux-linkage vs rotor position waveform which has been interpolated onto a finer resolution with 0.9° interval between rotor displacements. Using the interpolated flux linkage-rotor position curve, information liked the windings inductances and current waveforms can be calculated very rapidly.



(a) Original FEA data, with 20 data points

(b) Interpolated FEA data, with 100 data points

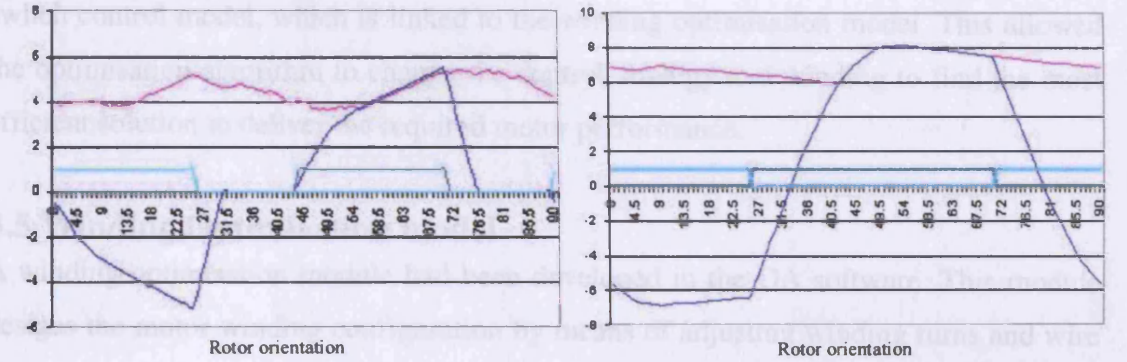
Fig. 3.9 (a) & (b) Example on the increased in number of FEA data point by interpolation

3.4 Switch control model

The decision of when the control switches must be turned on or off to achieve optimal firing position within the electrical cycle for optimum motoring performance is dependent on the speed and load conditions. In practice, the common method to determine optimum firing angles for different operating conditions would involve performing a series of dynamometer tests.

The torque of flux switching motor is mainly produced as a result of the change in (mutual) inductance between field and armature windings as the rotor turns which has a

maximum value when the rotor and stator poles are fully aligned. At this point, the rate of change of the mutual inductance is zero and for this reason the armature back-emf and hence torque passes through zero near the pole alignment positions. As the speed of the FS motor increases the increasing armature back-emf inhibits the fast rise in current for a given supply voltage. Therefore, the commutation of the armature current must occur in advance of the reversal of the armature back-emf (pole alignment) to allow current to build up.



(a) Reduced pulse width, operating on partial load

(b) Advanced commutation angle, operating on full load

Fig. 3.10 (a) & (b) The current waveforms simulated by the drive system model for different speed-load requirements

Efficient operation of the FS motor is determined by the control of the firing angles. During light or no load operation the reduced current input (fig. 3.10a) is introduced to reduce power losses, this was achieved by reducing the turn-on time (and introducing time-delay between turn-off and next turn-on) to the power switches. The time-delay function in the switch control model monitors the state of the armature current (eq. 3.7) as it decays to zero after a power switch is turn-off, and at zero crossing of the armature current hold the current at zero until the next (specified) turn-on of the other power switch.

At high speed conditions, advanced commutation (-9° mechanical) of the armature current as depicted in fig. 3.10b is required to allow current in the (armature) winding to build up to a required level, before the back emf starts rising as the poles starts to align. The developed switch control model enables advanced, retard and time-delay to be

introduced into the control of the power switches (illustrated by fig. 3.10a&b). Thus allowing the drive system model to simulate arbitrary current waveforms that are relatively close to real operational ones. The GA driven optimisation software uses the user specified requirements on the motor's intended speed and load in conjunction with the switch control model (which is integrated to the electrical equivalent circuit model) to seek the optimum firing angle of the motor. The optimum firing angles are sought using a "ranged enumerative" method (to be explained in next section) employing the switch control model, which is linked to the winding optimisation model. This allowed the optimisation algorithm to change the control strategy and winding to find the most efficient solution to deliver the required motor performance.

3.5 Winding Optimisation model

A winding optimisation module had been developed in the GA software. This module designs the motor winding configuration by means of adjusting winding turns and wire size (selected from a table of standard wire gauges), and number of strands to meet design requirements (eg. average torque at specified operational speed), with considerations of maximum packing factor (copper volume allowed in slot, where the size of the slots of different lamination designs are obtained from FEA) and best motoring efficiency. The motor specification is usually based on the full-load and required operational speed. The winding optimisation routine is summarized by fig. 3.11. The optimisation of winding configuration starts with a predetermined firing angle. However, to determine if the designed winding configuration for required torque at operational speed has the best motoring efficiency, a set of solutions of different firing angles would be simulated, and the search range of the optimal firing angles is fixed at ±5° of the predetermined firing angle with 1° step (referred to as "ranged enumerative" method). The results are then compared to give the solution with the best efficiency which conforms to motoring requirements.

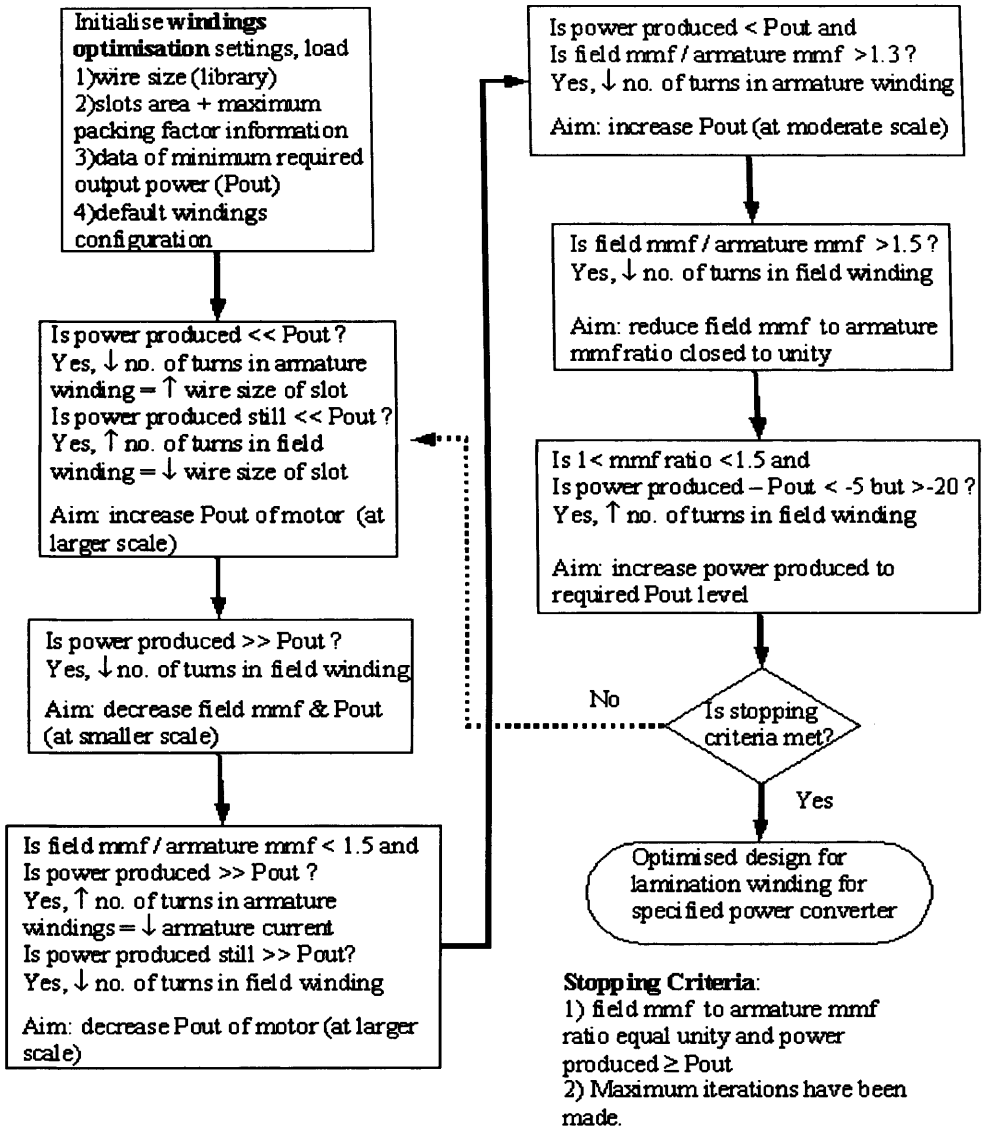


Fig.3.11 The procedure developed for optimising lamination winding for a specified power converter

3.6 Comparison of experimental and simulated results

The accuracy of the predicted motoring performances by the drive system model depends on how closely it can predict the armature back-emfs and the winding current compared to experimental results. Fig. 3.13 shows a comparison of simulated and measured armature back-emf waveforms. The line labeled as experimental, shows the armature back-emf measured on a prototype 8/4 flux switching motor (design details given later and shown in Appendix) while spinning the shaft at a speed of 600r/min. The field winding was excited from a separate dc supply at a current level of 6A. In the experiment, the field winding was connected in series with an external 10mH inductance to minimize the modulation in the field current. This was to ensure that the

measured experimental field current (fig. 3.12) resembled as closely as possible to the constant field current used in the FEA calculations. The other trace labeled as FE simulated is the simulated back emf waveform and comparison between the two are considered to be in good agreement.

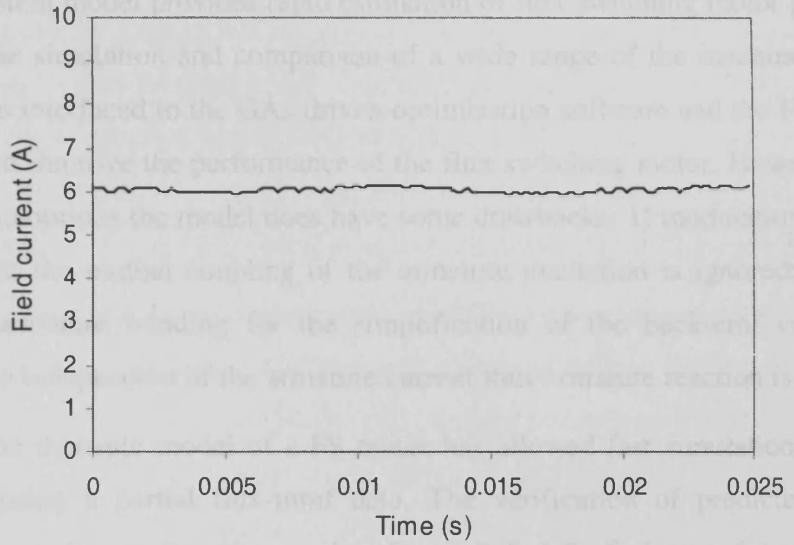


Fig. 3.12The measured current flowing through the field winding

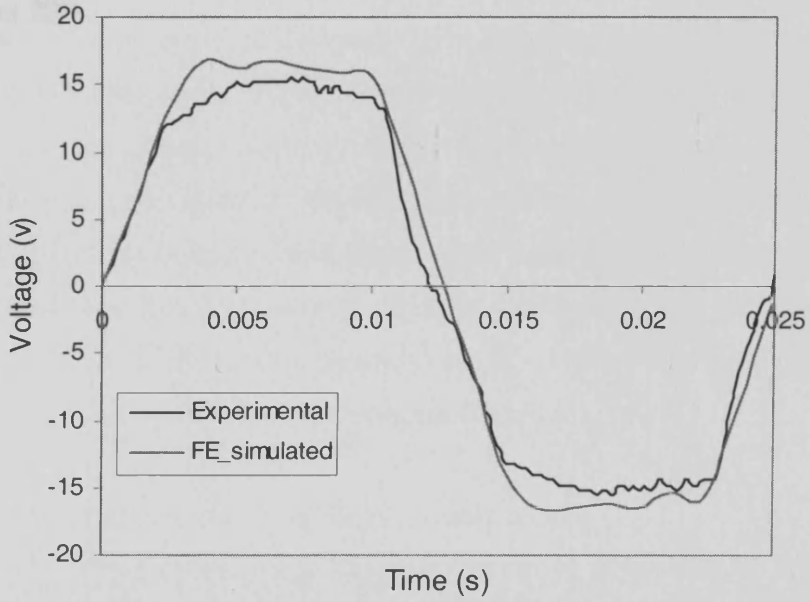


Fig. 3.13 Comparison of measured and predicted armature back emf waveforms at 600r/min with 6A flowing in the field winding

The slight differences between FEA simulated waveform and experimental result is due to the field winding internal emf not being modelled in the equivalent circuit model. The calculation of field winding induced emf in the equivalent circuit is ignored

because of the following reasons: 1) Referring to Fig. 3.2 the field flux linkage does not vary substantially with rotor position, as a result the induced emf in the field winding is almost negligible. 2) The generated flux linkage data are only based on field excitation while the armature is unexcited to facilitate rapid calculation of lamination performance.

The drive system model provides rapid estimation of flux switching motor performance facilitating the simulation and comparison of a wide range of the machine lamination shapes. When interfaced to the GAs driven optimisation software and the FEA solver it can adjust and improve the performance of the flux switching motor. However because of certain assumptions the model does have some drawbacks: 1) modulation of the field current due to the mutual coupling of the armature excitation is ignored; 2) the flux linking the armature winding for the simplification of the back-emf calculation is assumed to be independent of the armature current thus armature reaction is not model.

The simplified dynamic model of a FS motor has allowed fast simulation of different laminations using a partial flux-mmf data. The verification of predicted results to experimental results can be referenced to Fig. 6.8 & 6.9 of chapter 6 (on page 109), where underestimation of the simulated field winding current to experimental waveform is as much as 20%.

4 Losses in a flux switching motor

4.1 Introduction

The design of high-efficiency flux switching motor requires a loss model that can accurately predict the different losses occurring at any motor operating point. In general, the losses appearing in the motor are divided into mechanical and electrical losses. The electrical losses consist mainly the copper losses and the iron losses, while the mechanical losses combine both frictional and windage losses. The loss modelling of the flux switching motor becomes complex when accounting for the iron losses. The chapter focuses mainly on the prediction of the iron losses in the flux switching motor.

The iron losses of the flux switching motor are difficult to estimate and to measure with good accuracy. Most analytical models used to calculate iron losses require detailed information on the machine's geometry and the loss constants of the lamination steel. The models may have to take account of saturation and harmonics, which can both vary substantially in different sections of the machine. The iron losses incurred during operation cannot be measured directly, but have to be determined indirectly by some form of power flow measurement and by separation of the electrical and mechanical losses. However minor errors in the power flow measurement can propagate into large errors on the loss calculation. Electrical and mechanical losses are very dependent on temperature and speed, respectively. Erroneous measurements of these quantities further corrupt the iron loss prediction. In the following section, the magnetic characteristic of the lamination steel is examined and an iron loss model based on the *loss separation* method is introduced. The iron losses in different sections of a flux switching motor are estimated and compared to measurements. Possible sources of errors and the accuracy of both the estimation and the measurement are discussed.

4.2 Magnetic characteristic of lamination steels

For the Flux Switching Motor, the magnetic properties of lamination steel have a strong influence on its motor performance. In order to accurately estimate the motor performance, the information on the variety of lamination steels and the mechanisms determining the quality of soft magnetic material need to be investigated. Determination of the size and distribution of the iron losses is one of the most important fields in the optimisation of an electric motor, as knowledge of the iron losses is essential to the estimation of efficiency and other motor performances. Thus, it is useful to give a

general overview on the different steel types, describe the properties of lamination steel, and those phenomena that determine the steel quality. Some relevant properties of lamination steel are explained, and the section concludes on which electrical steel properties may be used to model the iron loss.

4.2.1 Electrical sheet steel in general

Lamination steels may be defined as magnetically soft, thin, steel sheets, generally in the range 0.15 to 1 mm thick. There are two major types of electrical sheet steel, grain-oriented and non-oriented steels.

Grain-oriented steel is magnetically anisotropic, and has superior magnetic properties in the direction of rolling. These special qualities are due to a combination of a certain chemical composition, rolling and heat treatment [21]. A number of post manufacturing treatments are also applied to enhance certain desirable properties. These include annealing under the influence of applied magnetic field, laser scratching, etc. Grain-oriented steels are more expensive than non-oriented steels and are primarily used in distribution and power transformers, and very large rotating electrical machines.

Non-oriented steel is generally isotropic, having the same mechanical and magnetic properties in all directions. These steels are often used in small electric motors. Non-oriented steel can be supplied fully processed or semi processed. Semi processed steel may be supplied in unalloyed (silicon free) or alloyed (containing low percentages of silicon) form. Alloyed steel sheets with high silicon content (up to 6.5 %) are very hard and brittle, and conventional manufacturing techniques cannot be applied. The steel sheets of motor cores of alloyed steel are produced by the wire cut or laser cut methods that are very expensive. Semi-processed steel is easier to punch, but requires annealing or decarburising, and steam blueing after punching in order that the material may attain its guaranteed magnetic properties. Annealing affects the internal structure and composition of the steel, while steam blueing adds a surface of iron oxide, which serves as an insulation layer between laminations [22].

The main issue affecting motor efficiency is the losses. Reduction of losses will increase the efficiency for given load conditions. Iron losses components of lamination steels can generally be reduced by:

- Reduction in sheet steel thickness. (Eddy current)

- Increasing resistivity of the core material by increasing the silicon content. (Eddy current)

- Reduction in grain size. (Eddy current)

- Increasing the purity of the material. (Hysteresis)

- Reduction in internal and surface strain. (Hysteresis and Eddy current)

Non-oriented steel has been used for constructing the several prototype machines that have been investigated in the work related to this thesis. To illustrate the variety in quality of non- oriented steel fig. 4.1 shows the magnetic properties of different steel grades available from European Electrical Steels.

Grade EN 10106	Specific total loss at 50 Hz		Anisotropy of loss	Magnetic polarization at 50 Hz			Coercivity (DC)	Relative permeability
	$\hat{J}$= 1,5 T W/kg	1,0 T W/kg	%	$\hat{H}$ = 2500 T	5000 T	10000 A/m T	A/m	at 1,5 T
M235-35A	2,25	0,92	10	1,53	1,64	1,76	35	660
M250-35A	2,35	0,98	10	1,53	1,64	1,76	40	630
M270-35A	2,47	1,01	10	1,54	1,65	1,77	40	730
M300-35A	2,62	1,10	10	1,55	1,65	1,78	45	810
M330-35A	2,93	1,18	10	1,56	1,66	1,78	45	830
M250-50A	2,38	1,02	10	1,56	1,65	1,78	30	800
M270-50A	2,52	1,07	10	1,56	1,65	1,78	30	830
M290-50A	2,62	1,14	10	1,56	1,65	1,78	35	800
M310-50A	2,83	1,23	10	1,57	1,66	1,79	40	930
M330-50A	3,03	1,29	10	1,57	1,66	1,79	40	950
M350-50A	3,14	1,33	9	1,58	1,67	1,79	45	960
M400-50A	3,58	1,54	9	1,58	1,67	1,79	50	1020
M470-50A	4,05	1,79	6	1,59	1,68	1,80	60	1120
M530-50A	4,42	1,96	6	1,59	1,68	1,80	70	1150
M600-50A	5,30	2,39	6	1,63	1,72	1,83	85	1620
M700-50A	6,00	2,72	5	1,64	1,72	1,84	100	1680
M800-50A	7,10	3,22	5	1,65	1,73	1,85	100	1680

Fig. 4.1 Typical magnetic properties of non-oriented electrical steels of different steel grade

From fig.4.1 it may be seen that the specific iron losses vary from 2.25 W/kg to 7.1 W/kg. This means that the choice of steel type will be an important factor affecting the efficiency of any electric motor, if iron losses comprise a significant part of the total losses. In general, a complete view must be taken when selecting a material, as materials exhibiting low iron losses sometimes exhibit lower permeability, causing increased magnetising current with associated increased copper losses.

4.2.2 Steel properties suitable for use in modelling iron loss

Steel manufacturers generally supply standard data relevant for modeling of the magnetic behaviour of electrical steel.

- Graph showing the BH Curve of normal magnetization; the BH-characteristic is important to determine the total flux in the motor and the relationship between current and voltage.
- Loss data at different frequencies; loss data at 50Hz or 60Hz for different values of peak magnetic induction in graph or tabular form is data commonly supplied by the manufacturer. The loss vs peak magnetic induction curves of different frequencies are used to determine the constants used in equations for modeling the iron loss.
- Mass density, δ; the mass density of the steel is used to calculate the actual loss in the motor when the specific loss (per volume or per kg) is known.
- Resistivity, ρ; the resistivity of the sheet steel is used in the estimation of eddy current losses.
- Thickness, d; the lamination thickness is a parameter used in defining the nature of the eddy current losses.

In addition stacking factor, hardness, tensile strength, and tolerances etc. are specified by electrical steel manufacturers, but are not used in deriving the total iron losses.

4.3 Iron loss

Before any attempt to analyze the different methods of interpreting the enigmatic iron loss in electrical machines, it would be useful to have a brief overview over the theory of magnetization and the iron losses. The term iron losses generically refer to the various energy dissipation mechanisms taking place when a magnetic material is subject to time-varying external field. As a consequence of the inherent irreversible nature of magnetization process, part of the energy injected into the system by the external field is irrevocably transformed into heat [23]. Serious discussions of magnetization theory and the reasons for iron losses can be found in work of Bertotti [23,24].

The internal structure of lamination steel consists of many magnetic domains which differ from each other by the orientation of their magnetization vectors, and the magnetic domains are separated from each other by the domain walls. During the magnetization process, the large-scale nucleation of magnetic domains, which are reinforced by the external field and annihilation of magnetic domains of opposing magnetization vector (fig. 4.2) result in the damping of domain wall movement by eddy

currents and spin-relaxation and these are the physical origins of the iron losses [23-25]. Impurities and imperfections inside the material hinder the domain wall motion and causes irregularities (like sudden rapid movement or jump) in the motion of domain wall. Therefore, the movement of the domain walls is not regular and the local velocity of the walls is not equal to the change of the rate of the external field.

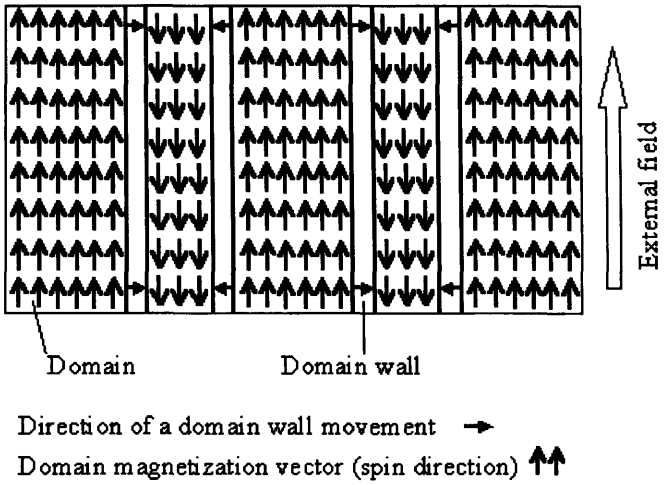


Fig. 4.2 Change in domain structure due to motion of domain walls

4.3.1 Different methods of quantifying iron loss

Given the complexity of the problem, it would be useful to apply simple equations in an iron loss model, and preferably obtained a direct calculation of iron losses as a function of peak induction, magnetization frequency, and the core material characteristics as described in manufacturers core loss data. This would provide a direct means of quantifying the iron losses. Several different methods for determining the size and distribution of iron losses in electric machines already exist. They can generally be classified as the hysteresis models, the empirical equations and the *loss separation* methods.

4.3.1.1 Hysteresis models

The vast number of hysteresis models can be separated into two branches. One is based on the Jiles-Atherton model and the other on the Preisach model.

The Jiles-Atherton model is based on a physical model of energy calculation. It consists of a differential equation that describes the static behaviour of ferro- and ferromagnetic material using four parameters [26]. An iterative procedure has to be used to estimate the parameters of the model where the initial parameters have to be determined from the

graphic derivation of parts of a reference hysteresis loop. The model can be extended to a dynamic calculation but the number of required parameters has to be increased, the extra parameters describe the temperature behaviour and calculate minor hysteresis loops [27].

The Preisach model introduces a statistical approach for the description of the time-and space distribution of the domain wall movement. However the classical Preisach model exhibits an essential drawback for being only a static model. In order to extend the use of the model to be applicable to higher frequencies a weight function needs to be included but the identification of the constants of the weight function requires a tremendous experimental effort [28].

Although the methods mentioned above attempted to model the physical behaviour of the magnetization process, they are of limited practical use. As the suppliers of the magnetic materials do not provide the parameters used in the models, furthermore the temperature dependence of the hysteresis loop is difficult to be accounted for.

4.3.1.2 Empirical equations

Many empirical equations are based on the Steinmetz equation (eqn. 4.1) [29] for direct calculation of the core losses without using the intermediate step of a hysteresis loop description. The equation uses the manufacturers core loss data, based on sinusoidal excitation to calculate the iron loss per volume, P_v as a function of magnetization frequency, f and peak induction level, B. The method seems to be the most practical and direct way to calculate the losses, however the equation is only valid for accounting of iron loss due to sinusoidal excitation. The equation uses three empirical parameters C_m, α, β. Both exponents are non-integer numbers, $1<\alpha<3$ and $2<\beta<3$. However, in modern day technology non-sinusoidal voltages and currents at different switching frequencies are usually used to drive the various electromagnetic devices, for these reasons it made the previous assumptions invalid. In order to estimate the iron loss due to non-sinusoidal excitation, a modified Steinmetz equation (MSE) was developed [25](eqn. 4.2).

$$P_v = C_m f^{\alpha} B^{\beta} \qquad \left[W/m^3\right] \qquad (4.1)$$

$$P_v = (C_m f_{eq}^{\alpha-1} B^{\beta}) f_r \qquad \left[W/m^3\right] \qquad (4.2)$$

where f_{eq} is the equivalent frequency of the non-sinusoidal induction waveform, f_r is the repeated frequency. With the introduction of an equivalent frequency, MSE provides a good fit to experimental measurements of iron loss in a ferrite core under triangular induction waveform [25].

4.3.1.3 Loss separation methods

The concept of loss separation is based on the phenomena of magnetization and uses simple terms to state experimental observations related to the losses. In equation (4.3) describes the existence of three components that contributed to the total iron loss. They are separated into the classical loss, P_{cl} related to the material geometry, the hysteresis loss, P_h as a result of the structural inhomogeneity and domain wall pinning, and the excess loss, P_{exc} due to size of the magnetic domains.

$$P = P_h + P_{cl} + P_{exc} \qquad (4.3)$$

The hysteresis loss is related to the physical phenomenon of the domain wall movement in response to an external field. At very low external fields, domain walls move slightly and remain pinned to defects within the magnetic structure. As the external field increases, the domain walls through sudden jump become unpinned from the defects. Physically the hysteresis loss is caused by very localised irreversible changes during the magnetization process, which make it dependent only on the peak induction, B when no minor hysteresis loops appear in the induction waveform [30]. The loss can be expressed in the form of equation (4.4) where f is the magnetization frequency, k_h is the hysteresis loss constant and the exponent α is a non-integer number which has a range between 1.6 to 3 [31].

$$P_h = k_h B^{\alpha} f \qquad [W/kg] \qquad (4.4)$$

The classical loss (due to eddy currents) is the loss calculated from Maxwell equations based on a perfectly homogeneous conducting medium, that is the conducting medium has no structural inhomogeneity and no magnetic domains [31]. The scale of the classical loss is determined by the material geometry, for example thickness of a lamination, d. The geometry controls the boundary conditions of Maxwell equations, which in turn determine the distribution of the eddy currents in the lamination cross-section and the resulting losses due to the Joule effect. The classical loss acts as a sort of background loss present under all conditions, while other losses are added when

including structural inhomogeneity and magnetic domains into the model. By solving Maxwell equations, the classical loss under sinusoidal induction can be represented with equation (4.5), where d is the lamination thickness in m, ρ is resistivity of the material in Ωm, δ is mass density in kg/m^3, B is the peak induction in Tesla, and f is the magnetization frequency. To extend the used of equation to accounted for the loss due to non-sinusoidal induction equation (4.6) is introduced [30].

$$P_{cl} = \frac{(\pi d)^2}{6\rho\delta}(Bf)^2 \qquad [W/kg] \qquad (4.5)$$

$$P_{cl} = \frac{d^2}{12\rho\delta} f \int_T \left\{\frac{dB(t)}{dt}\right\}^2 dt \qquad [W/kg] \qquad (4.6)$$

The eddy currents tend to concentrate around the moving domain walls, leading to losses which are higher compared to classical loss model which assumed a homogeneous conducting medium and the difference between them is known as the excess loss [24]. The excess loss arises from the smooth, large-scale motion of domain walls across the material cross-section, when sudden jump of domain walls responsible for the hysteresis loss are disregarded [23]. In references [23, 30], the excess loss is represented by equation (4.7) for sinusoidal induction and equation (4.8) for non-sinusoidal induction, where k_e is the constant related to the excess loss.

$$P_{exc} = k_e \sqrt{1/\rho}\,(Bf)^{1.5} \qquad [W/kg] \qquad (4.7)$$

$$P_{exc} = k_e \sqrt{1/\rho}\, f \int_T \left|\frac{dB(t)}{dt}\right|^{1.5} dt \qquad [W/kg] \qquad (4.8)$$

4.3.1.4 Comparison between Steinmetz equation and *loss separation* method

After much effort in examining the different methods of calculating the iron loss, two methods appeared to take the practical and direct approach in estimating the iron loss. They are the Steinmetz equations (as in eqns. 4.1 & 4.2) and the loss separation method (eqns 4.3 to 4.8). Both methods have three coefficients that are required to be extracted using loss data provided by manufacturer. Both methods used rate of magnetization when deriving iron loss under non-sinusoidal excitation which does not require the flux density waveforms to be decomposed into their time harmonic components. Such rate of magnetisation methods are particularly well-suited to time-stepped field solutions,

because the rate of magnetization can be integrated as the time-stepping solution proceeds. A simple analysis was carried out to verify the most suitable method to be employed in calculating the iron loss of the flux switching machine.

In the simple analysis, the unknown coefficients in both iron loss models (Steinmetz, and *loss separation*) are determined by curve-fitting equation 4.1 for Steinmetz, and equations 4.4, 4.5 & 4.7 for *loss separation*, these equations are used to calculate the iron loss based on sinusoidal excitation against a typical loss data of a lamination steel (M660-50) under sinusoidal excitation, this is shown in fig. 4.3. The typical loss data at 50Hz and 60Hz used for extracting the unknown coefficients are data commonly provided by the manufacturer. M660-50, non-oriented semi-processed electrical steel is 0.5mm in thickness and has the following properties, resistivity of 17μΩ cm, and mass density of 7850kg/m^3. The determined values for α , k_h , and k_e used in the *loss separation* method are 2, 15.3×10^{-3}, 8.4×10^{-7} respectively. The determined values for coefficients, C_m, α, and β of the Steinmetz equations are 89, 1.3, and 2.

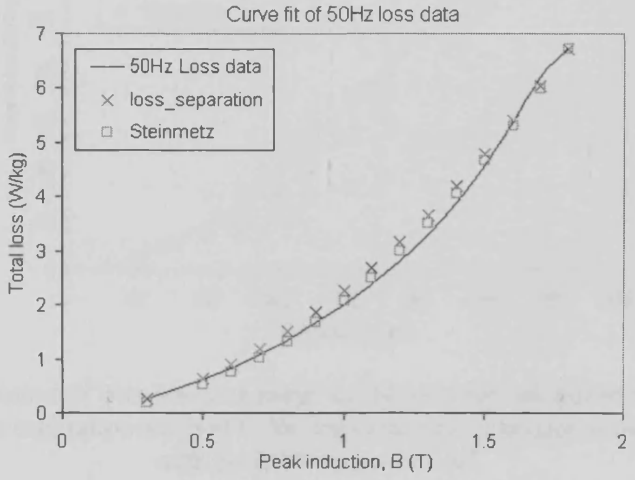


Fig. 4.3 Illustration of using Steinmetz equation and loss separation method to curve-fit the typical loss data of M660-50 at 50 Hz

The typical loss curves of both models are then extrapolated using their respectively equations with the coefficients derived earlier on, the two loss curves are plotted against another set of typical loss data obtained using sinusoidal peak induction of 1.6T at a range of different frequencies (fig 4.4). However, it has to be mentioned that the typical loss data shown in fig. 4.4 cannot usually be obtainable from the manufacturer material data book (data provided by courtesy of Prof. C. Pollock, University of Leicester). The accuracy of the two methods, Steinmetz equation and loss separation method can be

shown using fig. 4.4. The results show that the Steinmetz equation underestimated the total loss by >50% for most cases while the loss separation method understated the total loss by ~10%. A few observations are made from this simple experiment. Firstly the 50-60Hz loss data commonly provided by a manufacturer's material data sheet may not be sufficient to accurately extract the coefficients used in the equations (particularly for the Steinmetz equation). Secondly the Steinmetz equation may require different sets of coefficients to cover a wider range of frequencies. The loss separation method seems to be a more suitable method for calculating the iron losses, as the method is less sensitive to the possible error made in the derived coefficients from curve fitting. Hence, within the context of this thesis all iron loss calculated would be based on the *loss separation* method.

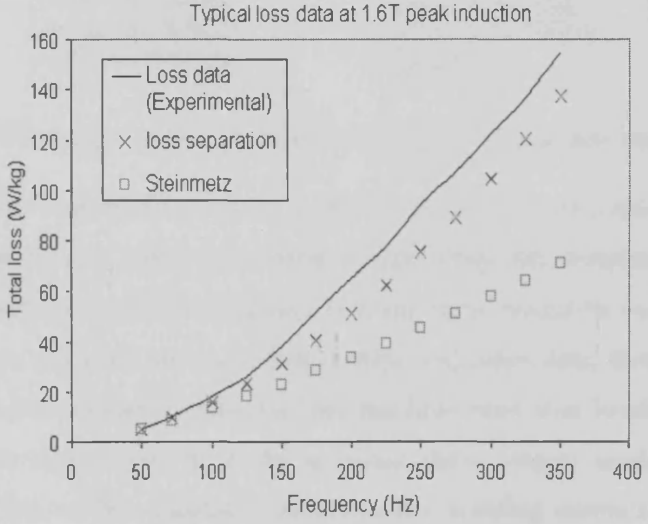


Fig. 4.4 Comparison of iron loss at a range of frequencies calculated using Steinmetz equation and loss separation method to the experimental loss data of material, M660-50 with peak induction of 1.6T

4.3.2 Modelling of iron losses in the FSM

In optimisation of the flux switching motor, the optimum lamination shape of the motor is a compromise between the copper volume and steel volume and this is strongly dependent on the motor specification based on the torque/speed range. In order for the automatic optimisation to perform correctly an accurate model of the iron losses in each design under dynamic conditions is required. The machine iron loss depends on numerous design and fabrication factors, such as the grade of lamination material and the heat treatment used for relieving stress in the material after fabrication, the method

of stacking the laminations, the speed and number of poles etc. However, the procedure described here (summarized in fig. 4.5) applied to flux switching motor, enables the iron loss to be predicted accounting for design and operational factors only, and assumes fabricated lamination stacks have retained pre-fabrication magnetic properties.

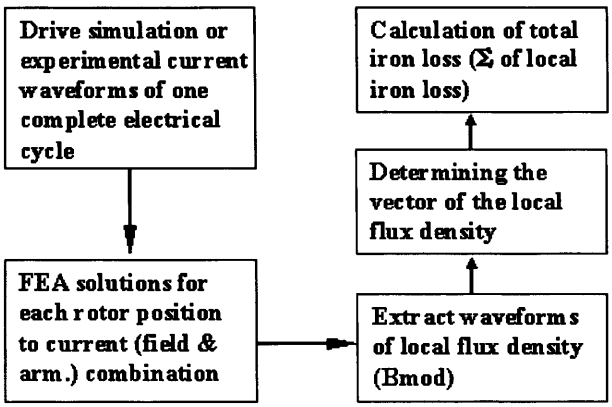


Fig.4.5 Flow chart of the procedure developed for iron loss prediction

The developed iron loss model could be used in two ways: 1) as a standalone model to predict FSM iron loss in which case experimental field and armature current of one complete electrical cycle and the machine winding turns would be used for setting up the FEA model to calculate the local flux density and other data, these data would be used in the iron loss model to calculate the machine total iron losses. 2) The model could be used in conjunction with the dynamic drive system model, and uses the dynamic simulation model calculated instantaneous winding currents (an example of simulated current waveforms is illustrated in fig. 4.6) and winding turns to setup the FEA software and results obtained would be used to derive the FS motor total iron losses. In both cases, the field and armature winding currents corresponding to 20 discrete rotor positions equally spaced over a full rotor pole pitch (full electrical cycle) are sampled and fed into FEA to obtain the data of local flux density (Bmod, the modulus of flux density) in each polygon of the machine model. The machine lamination is formed by a series of smaller polygons (fig. 4.7). The data of FEA calculated local flux densities and areas are then fed into the iron loss model for further processing to derive the machine iron loss. Firstly, the vectors of local flux density in different polygons of the machine are to be determined, and this is done using a simple technique (refer to section 4.3.2.1). An example of the determined solutions of localised flux density waveforms in different sections of the machine under motoring conditions

are shown in fig. 4.8 (a-d). The localized iron loss in each polygon of the model is consequently calculated using the loss separation equations throughout the magnetic circuit. Lastly summation of all the local losses gives the solution to total iron losses in the machine.

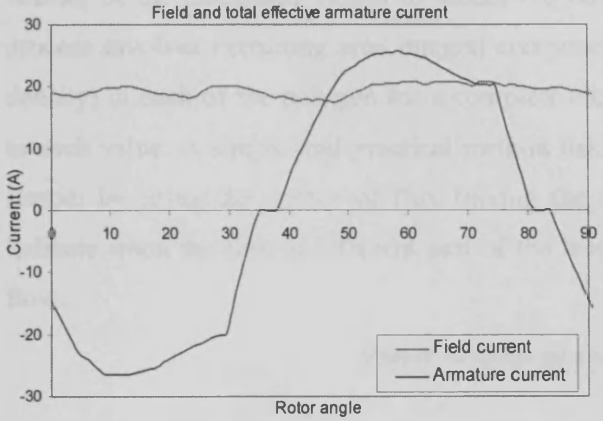


Fig. 4.6 An example of simulated field and armature current waveforms.

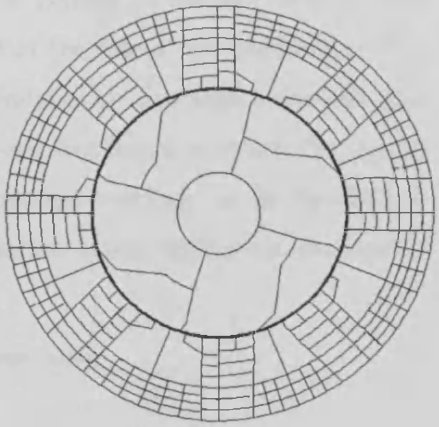
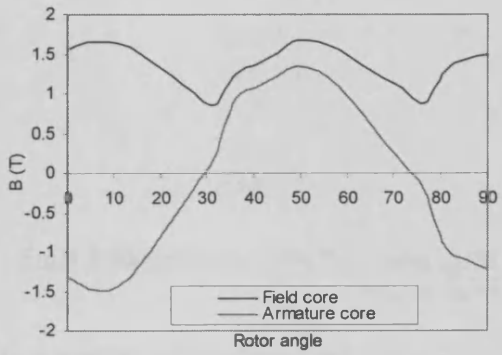
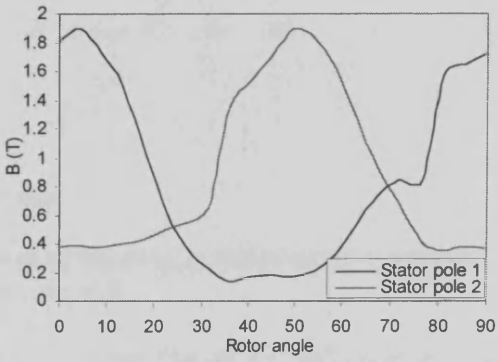


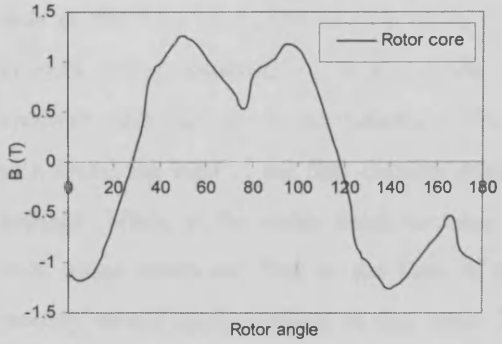
Fig. 4.7 A FS machine model built-up of small polygons.



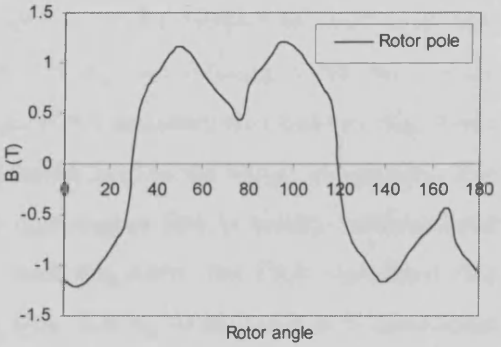
(a) In the stator back iron



(b) In stator teeth



(c) In rotor yoke



(d) In rotor pole

Fig. 4.8 (a-d) Flux density waveforms in sections of a flux switching motor

4.3.2.1 Technique for determining the vector of the flux density

When the FEA calculates local flux density in each polygon of the machine lamination, the output solution is in the form of an absolute value, which means only the magnitude of the flux density is given and not the direction, thus it is necessary to derive the vectors of the calculated values to model the rate of change of the flux density. The process involves extracting area integral component of the Bmod (the modulus of flux density) in each of the polygon for a complete electrical cycle and adds a relevant sign to each value. A simple and practical method has been developed to obtain the sign of Bmod; by using the vector of flux linking the armature winding (as in fig. 4.9) to indicate when the flux in different part of the lamination would change its direction of flow.

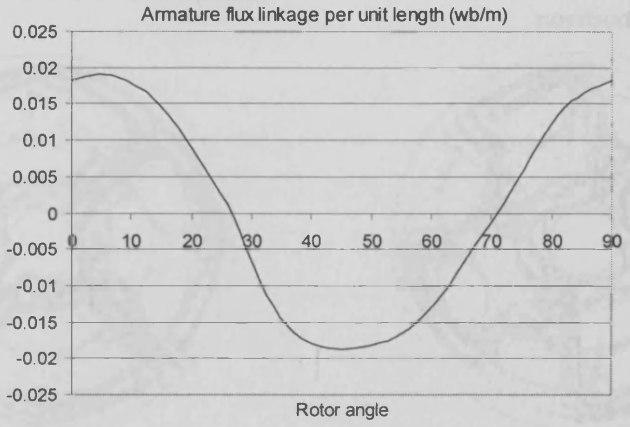
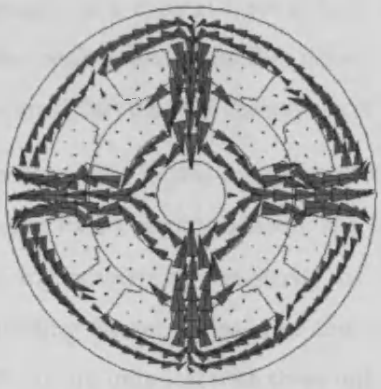


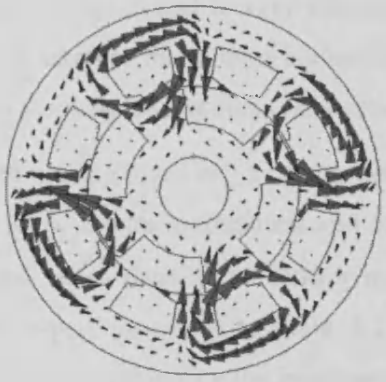
Fig.4.9 Illustration of the flux linking the armature winding at different rotor position over a full electrical cycle.

In a section of the field back iron the orientation of the flux is always unidirectional (either clockwise or counter-clockwise) (fig. 4.8a) thus the FEA calculated flux density data in the field back iron section could be applied straight without the need to include vectors of the components. Whereas the vector of the flux flowing in the back of the armature slot changes in accordance to the sign of the armature flux linkage (fig. 4.8a), as a result the sign of the flux density components is tied to the vector of armature flux linkage. While in the stator tooth section, the direction of flux is mostly unidirectional with some rotational flux at the root of the tooth (fig.4.8b), the FEA calculated flux density would apply straight in this case. The flux flowing in the rotor is bi-directional but its vector only changes at half the rate that of the armature flux linkage (fig. 4.8 c & d), thus the vector of flux density in the rotor would change its sign at the first armature

flux linkage zero crossing (fig. 4.9) and maintain the same direction for a period of a full electrical cycle, and this process is repeated with the next change of flux direction.

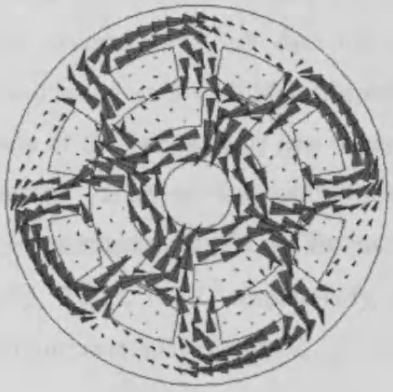


(a) Orientation of flux at 0° rotor position



(b) Orientation of flux at 22.5° rotor position

(c) Orientation of flux at 45° rotor position



(d) Orientation of flux at 67.5° rotor position

Fig. 4.10 (a-d) Illustration on the orientation of the flux in the motor lamination at different rotor positions over a full electrical cycle

4.4 Stages in verification of the iron loss model

Up to this stage, the development of an iron loss model had been completed, while the accuracy of the iron loss model would be verified against experimental results of a prototype flux switching motor constructed with Scotsil 800.5 lamination steel. The following sections describe the procedures used in verifying the accuracy of the developed iron loss model.

4.4.1 Curve fit method to determine the loss coefficients

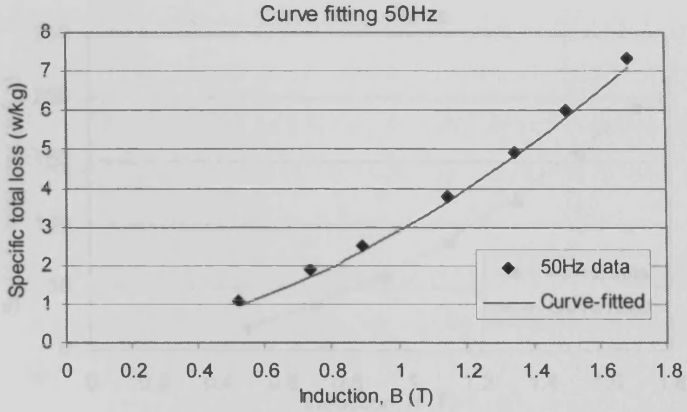
Based on experiences in the earlier section, we learned that the coefficients determined would be more accurate when curve fitted against a wider frequency range of loss data

(data used is provided by courtesy of Mr. R.T Walter, Black and Decker Inc.). The sets of loss data at various frequencies are collected based on a Ring test (stack of stamped out rings). In a typical Epstein test, the flux density is controlled to vary sinusoidally and the peak value of the flux density wave shape is taken as the reference condition. It is assumed that the flux density wave shape used in the Ring Test is sinusoidal as well.

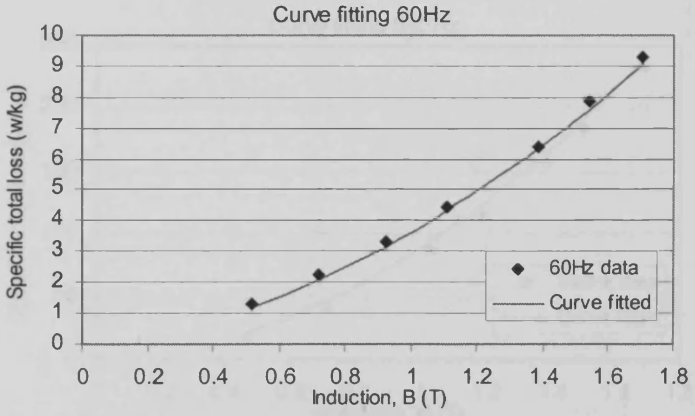
The motor used in this experiment was constructed using Scotsil 800.5, a non-oriented fully-processed electrical steel. The lamination sheet is 0.5mm in thickness and has the following properties, resistivity of 25μΩ cm, and mass density of 7800kg/m^3. By substituting these values into the sinusoidal loss equations (refer to table 4.1), the equations are only left with three unknown coefficients. To calculate the iron loss in the flux switching motor, the values of unknown coefficients used in the iron loss model needs to be determined. The coefficients α, k_h , and k_e can be determined by curve fitting the (sinusoidal) loss equations against the experimental loss data curves at different peak induction level over a range of frequencies (from 50-540Hz). Illustrations on determining the other three unknown coefficients of the (sinusoidal) loss separation equations are shown in fig. 4.11 (a-e), the extracted values of the coefficients are determined by the best-fitted curves for a range of magnetization levels and frequencies. The determined values for α , k_h , and k_e are 1.65, 0.046, 1×10^{-7} respectively, these values would be applicable for both sinusoidal and non-sinusoidal equations.

	Sinusoidal	**Non-sinusoidal**
Hysteresis loss	$P_h = k_h B^{\alpha} f$	$P_h = k_h B^{\alpha} f$
Classical loss	$P_{cl} = \frac{(\pi d)^2}{6\rho\delta}(Bf)^2$	$P_{cl} = \frac{d^2}{12\rho\delta} f \int_T \left\{\frac{dB(t)}{dt}\right\}^2 dt$
Excess loss	$P_{exc} = k_e\sqrt{1/\rho}\,(Bf)^{1.5}$	$P_{exc} = k_e\sqrt{1/\rho}\, f \int_T \left\lvert\frac{dB(t)}{dt}\right\rvert^{1.5} dt$

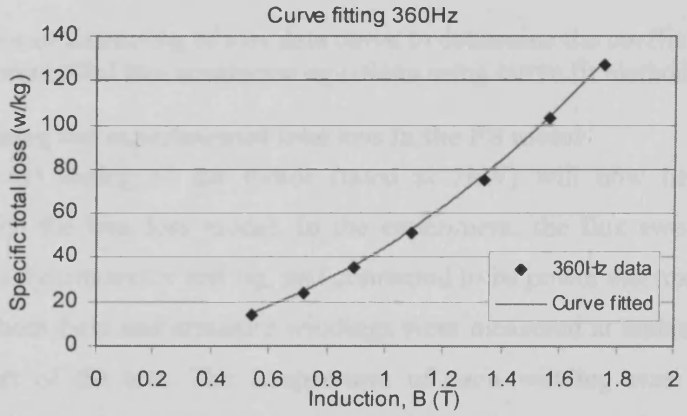
Table 4.1 Sinusoidal and non-sinusoidal equations of the loss separation method



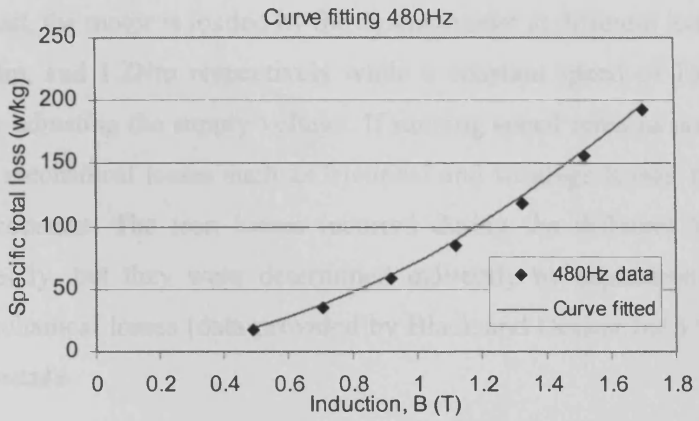
(a) Curve fit of sinusoidal equations to the 50Hz loss data of material, Scotsil 800.5



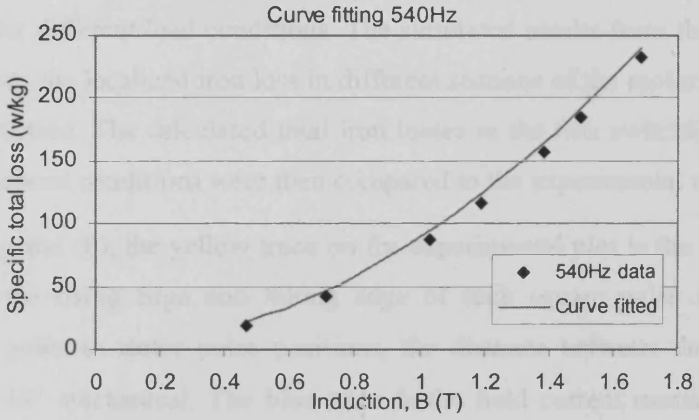
(b) Curve fit of sinusoidal equations to the 60Hz loss data of material, Scotsil 800.5



(c) Curve fit of sinusoidal equations to the 360Hz loss data of material, Scotsil 800.5



(d) Curve fit of sinusoidal equations to the 480Hz loss data of material, Scotsil 800.5



(e) Curve fit of sinusoidal equations to the 540Hz loss data of material, Scotsil 800.5

Fig. 4.11 (a-e) Extracting of loss data curve to determine the coefficients from sinusoidal loss separation equations using curve fit method.

4.4.2 Determining the experimental iron loss in the FS motor

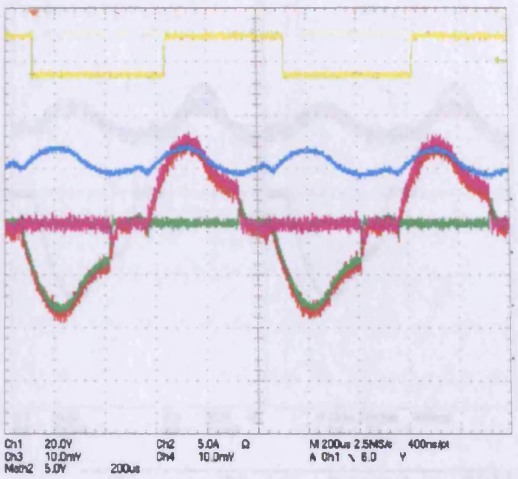
Full experimental testing of the motor (rated at 2kW) will now be presented for comparison with the iron loss model. In the experiment, the flux switching motor is mounted onto a dynamometer test rig, and connected to its power electronic circuit. The resistances of both field and armature windings were measured at ambient temperature before the start of the test. The temperature of each winding was also measured immediately after each test using thermocouples embedded in the coils to allow correction of winding resistances for running conditions. DC link voltage, current, speed, power input and mechanical output power are also recorded using a power analyser connected to the drive circuit and the dynamometer while winding current waveforms and shaft mounted optical sensor signal are recorded using a oscilloscope.

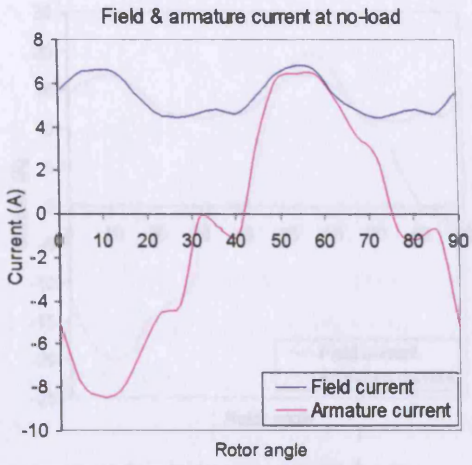
During each test, the motor is loaded by the dynamometer at different load conditions at no load, 0.5Nm, and 1.2Nm respectively while a constant speed of 15,000rev/min is maintained by adjusting the supply voltage. If running speed remains constant between tests then the mechanical losses such as frictional and windage losses, may reasonably be assumed constant. The iron losses incurred during the different tests cannot be measured directly, but they were determined indirectly by separation of the copper losses and mechanical losses (data provided by Black and Decker Inc.) from the power flow measurements.

Real values of current waveforms recorded from the experiments (fig. 4.12 a-c) are used in the FEA to obtain the instantaneous flux density distribution in various sections of the motor under different load conditions. The simulated results from the FEA are then used to calculate the localized iron loss in different sections of the motor as described in the previous section. The calculated total iron losses in the flux switching motor at the different load-speed conditions were then compared to the experimental results.

In figs. 4.12(a) and (b), the yellow trace on the experimental plot is the position sensor signal where the rising edge and falling edge of each square pulse represented the aligned rotor poles to stator poles positions, the distance between the two edges is equivalent to 45° mechanical. The blue trace is the field current measured at 5A/div, and the red trace is the effective armature current measured at 5A/div. In fig. 4.12(c), the yellow trace on the experimental plot is the position sensor signal, the blue trace is the field current measured at 10A/div, and the green trace is the effective armature current at 10A/div. Beside each experimental waveforms plot is an illustration of the extracted current waveforms corresponding to 20 discrete rotor positions of a full electrical cycle. This information is fed back to the FEA to calculate the localised flux density in the motor lamination at different load conditions.

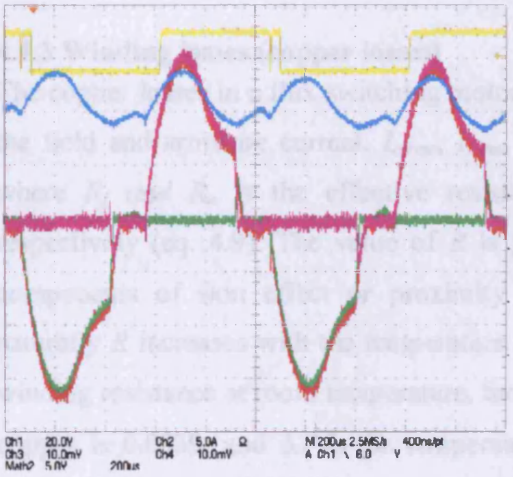
The experimental power flow measurements at different load conditions measured by the power analyser and the digital oscilloscope are also shown in the fig 4.12 (a-c). Noticeably, when the dynamometer was set to zero load the power analyser still records a small reading for the output power this is due to zero offset error in the load cell. The measured iron loss at each load condition was derived when the measured windings copper losses, the mechanical loss and output power are separated from the input power. Thus, within this context the measured iron loss could also mean other losses including the iron losses.

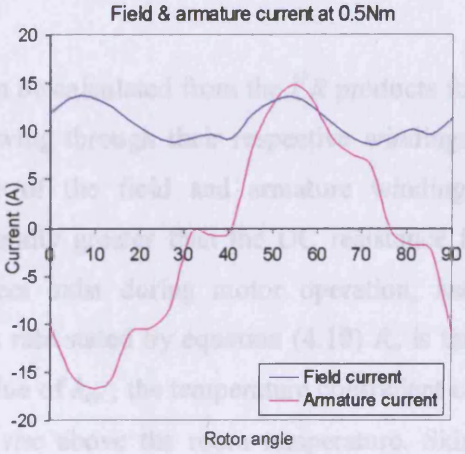




Torque (Nm)	Pin (w)	Pout (w)	Field copper loss (w)	Armature copper loss (w)	Mech. losses (w)	Measured iron losses (w)
No load	459.5	36.26	33	10.6	108	271.64

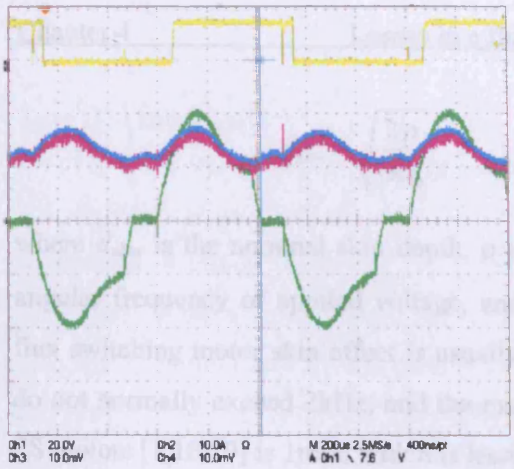
Fig. 4.12(a) Experimental measurements for the 8/4 FS motor running at 15,000rev/min with no external load and the extracted current waveforms to be used in FEA

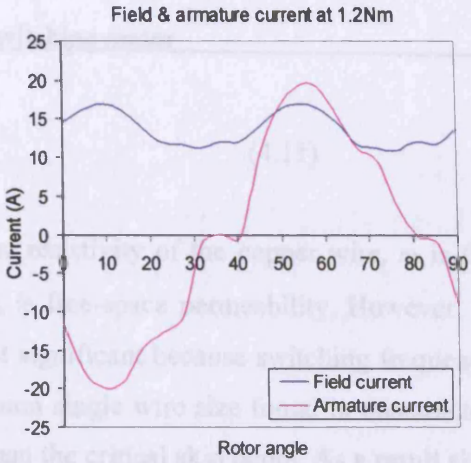




Torque (Nm)	Pin (w)	Pout (w)	Field copper loss (w)	Armature copper loss (w)	Mech. losses (w)	Measured iron losses (w)
0.5	1608.4	792.62	130	41.9	108	536

Fig. 4.12(b) Experimental measurements for the 8/4 FS motor running at 15,000rev/min with 0.5Nm externally applied load and the extracted current waveforms to be used in FEA





Torque (Nm)	Pin (w)	Pout (w)	Field copper loss (w)	Armature copper loss (w)	Mech. losses (w)	Measured iron losses (w)
1.2	3152.7	1862.7	236	78.26	108	867.77

Fig. 4.12(c) Experimental measurements for the 8/4 FS motor running at 15,000rev/min with 1.2Nm externally applied load and the extracted current waveforms to be used in FEA

4.4.3 Winding losses (copper losses)

The copper losses in a flux switching motor can be calculated from the I^2R products for the field and armature current, $I_{a,rms}$, $I_{a,rms}$ flowing through their respective windings, where R_f *and* R_a is the effective resistance of the field and armature windings respectively (eq. 4.9). The value of R is generally greater than the DC resistance if components of skin effect or proximity effect exist during motor operation, and naturally R increases with the temperature at a rate stated by equation (4.10) R_o is the winding resistance at room temperature, the value of k_{cu} , the temperature coefficient of copper is 0.0039, and ΔT is the temperature rise above the room temperature. Skin effect is likely to be a problem in very high speed motors wound with small numbers of turns (as bigger wire size is likely to be used to reduced copper losses). A rough estimation on the significance of the AC resistance due to skin effect can be made using equation (4.11).

$$P_{cu} = I_{a,rms}^2 R_a + I_{f,rms}^2 R_f \qquad (4.9)$$

$$R = R_o(1 + k_{cu}\Delta T) \qquad (4.10)$$

$$d_{skin} = \sqrt{\frac{2\rho}{\omega\mu_0}} \tag{4.11}$$

where d_{skin} is the nominal skin depth, ρ is the resistivity of the copper wire, ω is the angular frequency of applied voltage, and μ_o is free-space permeability. However, in flux switching motor skin effect is usually not significant because switching frequency do not normally exceed 2kHz, and the maximum single wire size found in constructed FS motors [1,18,19] is 1mm, which is lesser than the critical skin depth. As a result skin effect is not necessary to be accounted for in the loss model.

Proximity effect is the tendency for current to flow in undesirable patterns due to the presence of magnetic fields generated by nearby conductors. In reference [33], the proximity effect losses of a cylindrical conductor in a magnetic field is calculated with the following expression,

$$P_{pe} = \frac{\pi \cdot f^2 \cdot \left|\overline{B}\right|^2 \cdot l \cdot d^2}{128 \cdot \rho} \tag{4.12}$$

where f is the frequency of the magnetic field, l is the length of the conductor, d is the diameter of the conductor, ρ is the resistivity of the conductor, and $\left|\overline{B}\right|$ is the average value of the flux density over the region of winding (this value could be obtained using the static FEA). The calculated proximity effect losses of the test FS motor (mentioned in the previous section) at various load conditions are negligible, and the calculated results are shown in table 4.2.

Load condition	Loss due to proximity effect (W)
No load	0.179
0.5Nm	0.665
1.2Nm	0.932

Table 4.2 Calculated losses due to proximity effects in the FS motor at different load conditions

4.4.4 Dynamic losses

The frictional and windage losses are independent of the load of the machine. Frictional loss is the term used to represent the losses that occur in the machine bearings. The frictional loss can be very small if the bearings are anti-friction bearings. Windage loss is the term used to represent the power dissipated in the machine by friction with air. This loss is speed dependent as well as on the length of air gap, geometry of the rotor, and geometry of the stator bore [34]. The windage loss can be expressed as a function of rotational speed, ω, in the form of,

$$P_{windage} = k_w \omega^{\alpha} \qquad [W/m] \qquad (4.13)$$

where k_w is the windage coefficient, α is the curve fitting parameter experimental results of the mechanical losses are required to determine the value of the parameter. The test for the determination of the mechanical losses consists of measuring the motor deceleration rate as it stops to rest following the disconnection of the excitation source. The motor inertia needs to be determined when using this method. As the mechanical (or dynamic) losses cannot be easily determined by any analytical method thus it is not being modelled in the drive system. However the magnitude of the loss can be considered as a constant when the running speed is kept the same.

4.4.5 Comparison of iron loss between measured and calculated

Using the developed iron loss model, the iron losses in different sections of the motor model (fig. 4.7) running at 15,000rev/min under different load conditions were calculated. In Table 4.3 the iron losses of the motor under different load conditions were calculated using the "non-sinusoidal" equations (4.4, 4.6 & 4.8) and the calculated results were compared with the experimental ones. The results in Table 4.4 were computed using the "sinusoidal" equations (4.4, 4.5 & 4.7). Both sinusoidal and non-sinusoidal equations used the same set of values for the determined coefficients in the earlier section (4.4.1). The motor iron loss at 1.2Nm calculated using the "non-sinusoidal" equations was able to agree with measured results, however the iron losses at other loads were overestimated by as much as 55%. While motor iron losses at no load and 0.5Nm calculated using the "sinusoidal" equations shows good agreement with the test results but underestimate the loss at 1.2Nm by nearly 30%.

	Field back	Armature back	Stator pole	Rotor yoke	Rotor poles	Total loss (W)	Measured iron loss[1] (W)	% of deviation
No load	74	123	110.2	54.86	61.36	423.42	271.64	**+55.87**
0.5Nm	123.32	259.36	204.04	100.06	141.5	828.28	536	**+54.5**
1.2Nm	108.54	275.06	208.71	109.88	174.9	877.09	867.77	**+1.07**

Table 4.3 A breakdown of calculated iron loss using "non-sinusoidal" equations in different sections of the motor at 15,000rev/min with different externally applied load.

	Field back	Armature back	Stator pole	Rotor yoke	Rotor poles	Total loss (W)	Measured iron loss[1] (W)	% of deviation
No load	31.46	89.2	66.5	47.1	46.3	280.56	271.64	**+3.28**
0.5Nm	58.5	178.1	129.9	91.9	92.6	551	536	**+2.79**
1.2Nm	58.3	201.4	140.5	100.5	99.3	660	867.77	**-30.8**

Table 4.4 A breakdown of calculated iron loss using "sinusoidal" equations in different sections of the motor at 15,000rev/min with different externally applied load.

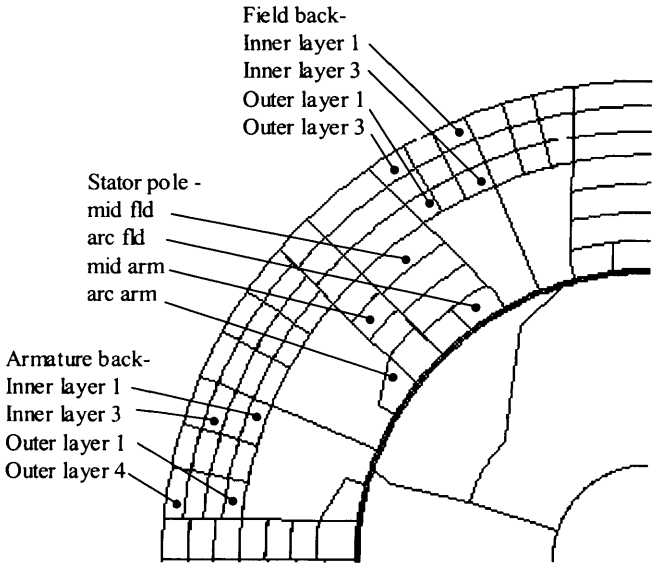


Fig. 4.13 Illustration on the polygon locations in the machine lamination where the various flux densities shown in figs. 4.14, 4.15 & 4.16 are found

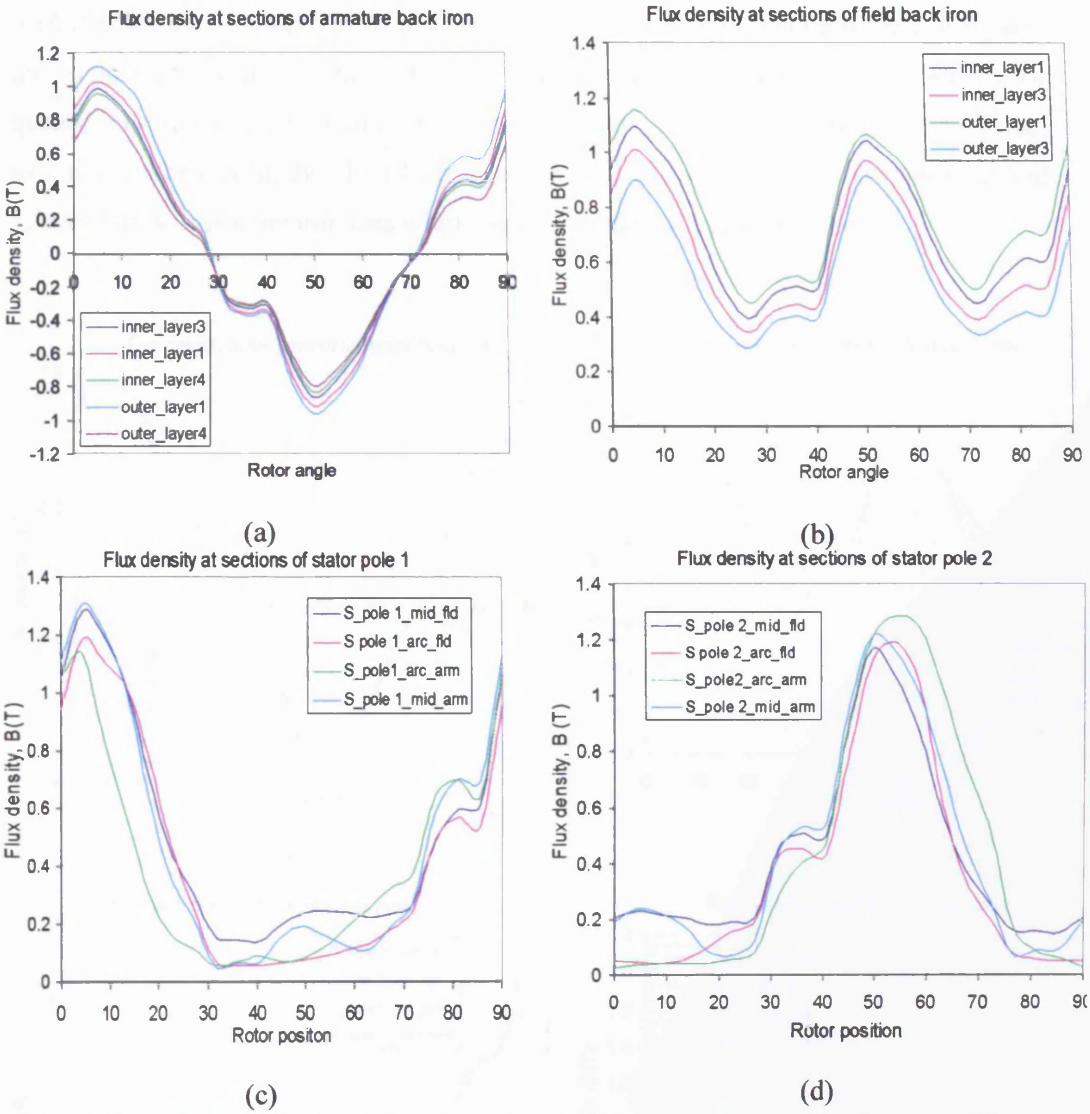


Fig. 4.14 (a-d) The calculated flux densities at different sections of the flux switching motor at 15,000rpm and no external load.

The discrepancies in the calculated results can be explained with the aid of figs. 4.13, 4.14, 4.15 & 4.16, the figures show the flux density waveforms at different sections of the motor when operating at 15,000rpm with no external load applied, with 0.5Nm load and 1.2Nm load respectively.

Firstly, a comparison between figs 4.14 & 4.15 the shape of the flux density waveforms in different parts of the motor between the two figures are almost similar while there are significant differences in their magnitudes. Therefore, the iron losses calculated using either sinusoidal or non-sinusoidal equations with their respective local flux densities could demonstrate the difference. In the other comparison was between figs. 4.15 &

4.16 the flux density waveforms in different parts of the motor between the two figures are almost identical in terms of wave shapes and magnitudes. Most methods of quantifying iron loss are based on the rate of magnetization or the peak induction value, as a result when using the almost identical flux density waveforms at 0.5Nm load and 1.2Nm load to calculate iron loss would lead to an almost identical solution.

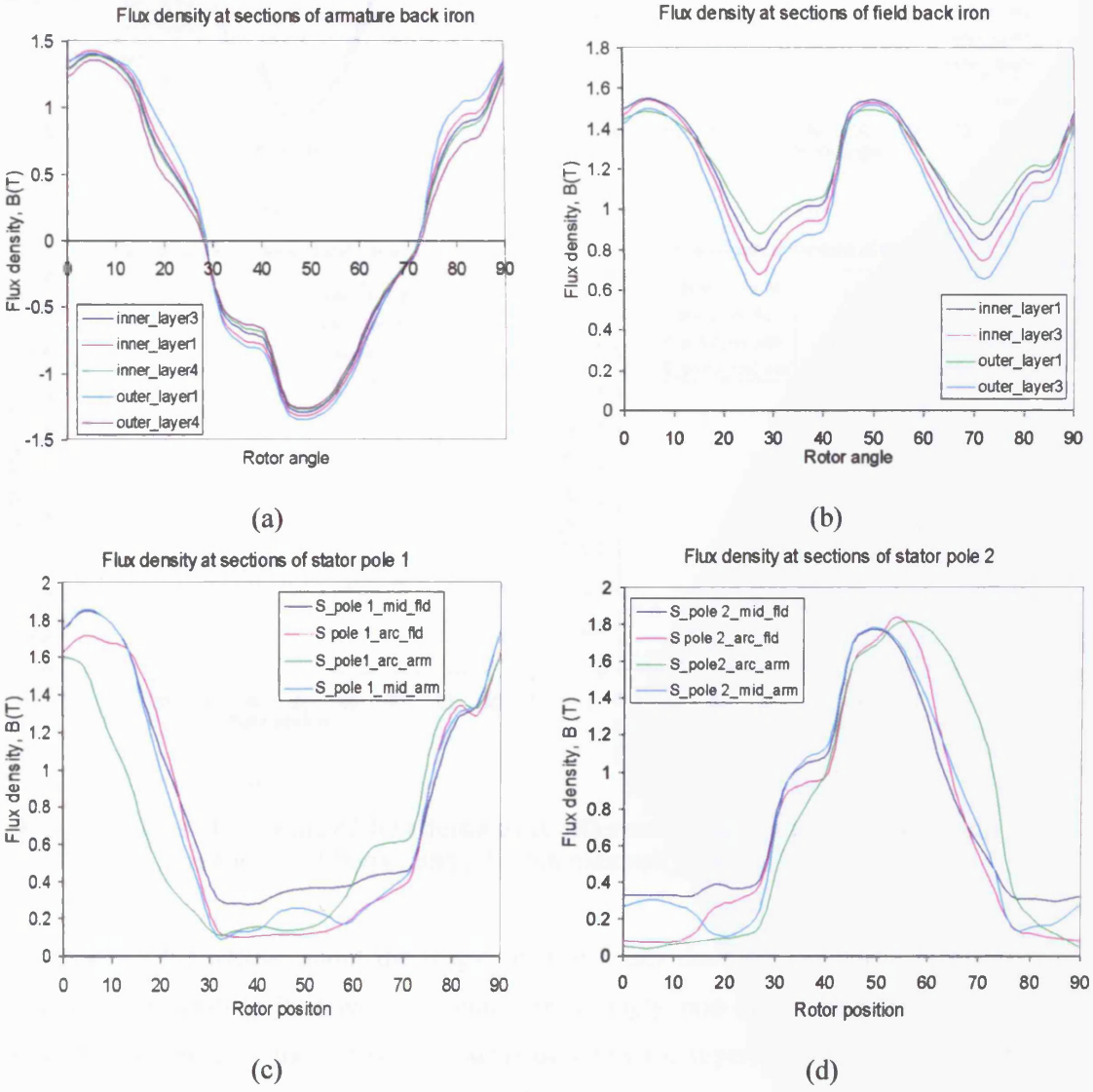


Fig. 4.15 (a-d) The calculated flux densities at different sections of the flux switching motor at 15,000rpm and a 0.5Nm externally applied load.

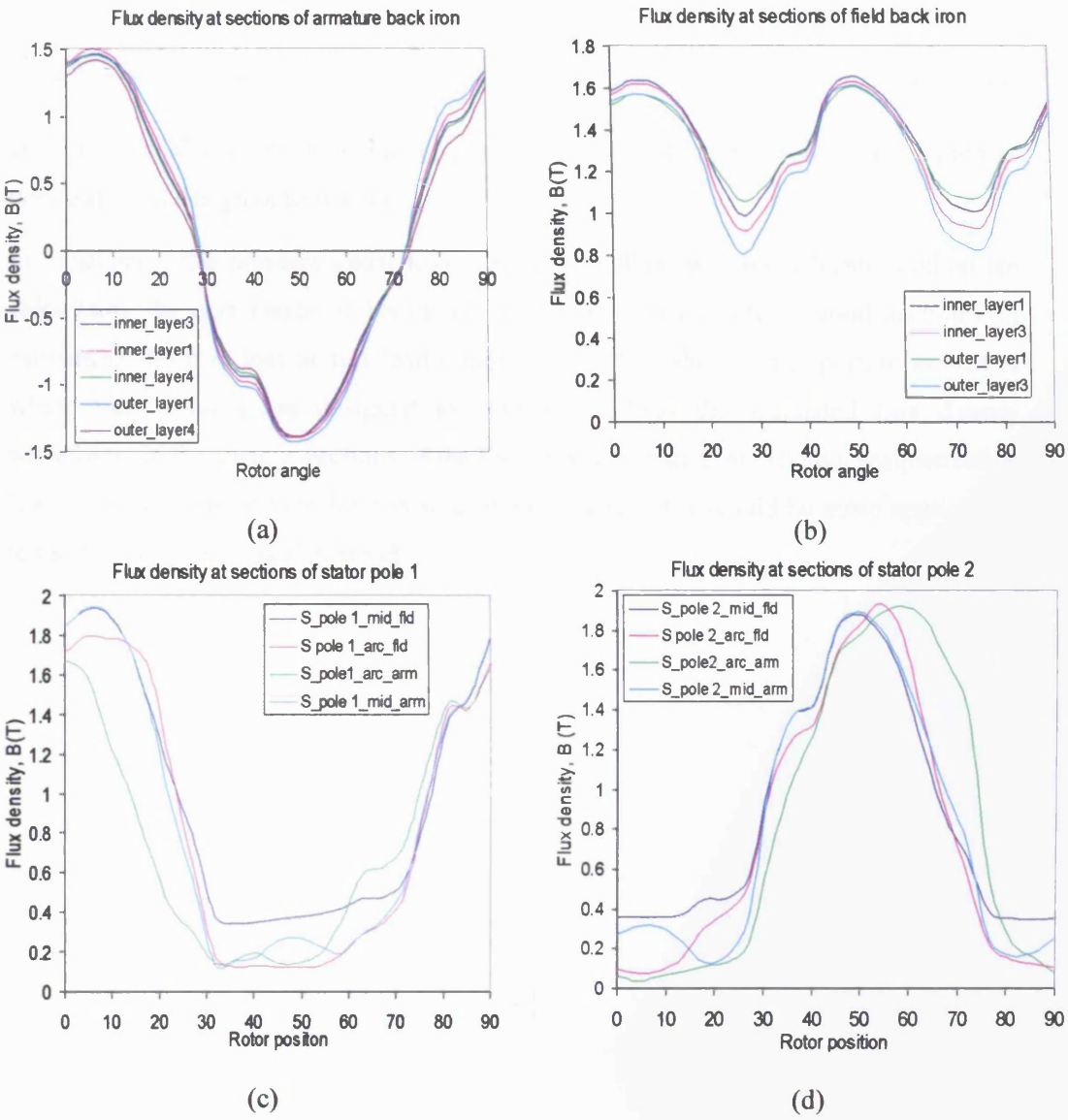


Fig. 4.16 (a-d) The calculated flux densities at different sections of the flux switching motor at 15,000rpm and a 1.2Nm externally applied load.

The detailed knowledge about the origin of iron losses does not provide a practical means of calculating it. Given the complex strongly non-linear character of the magnetization process, there is no obvious reason why the superposition law expressed by the *loss separation* equations should hold true under broad conditions. In fact, the equations are only statistical methods to the analysis of the stochastic magnetization process that one is able to show that the three scales (classical, hysteresis, and excess) affect the loss in an approximately statistically independent way. However, the

equations enabled a fast and direct approach of estimating the iron losses, which is necessary in a design optimisation cycle.

In conclusion, the non-sinusoidal loss separation method was the selected method for calculating the iron losses of FS motor, as the method has shown good accuracy in estimating the iron loss at full load condition which is the most important as that is where most motors are designed to operate at. Also the simulated flux density waveforms in the various sections of the FS motor lamination at different magnetization level have demonstrated to be non-sinusoidal, as a result it would be more appropriate to used the non-sinusoidal method.

5 Planning and development of the optimisation software

5.1 Introduction

This chapter describes the outline of the software development for the GA optimization software. The optimisation software is planned and developed using a commonly employed software development method, the method can be classified into six stages as illustrated in fig. 5.1, starting with project planning and ends with (final) installation. However, only the first three stages of the method are described in this chapter. The relationship of each stage to the others can be roughly described as waterfall, where the output from a specific stage serves as the initial inputs for the following stage. During each stage, additional information is gathered or developed, combined with the inputs, and used to produce the stage deliverables. The development of software projects using such method allowed the software to be built more systematically, and also enabled timescale involved within each stage to be more manageable.

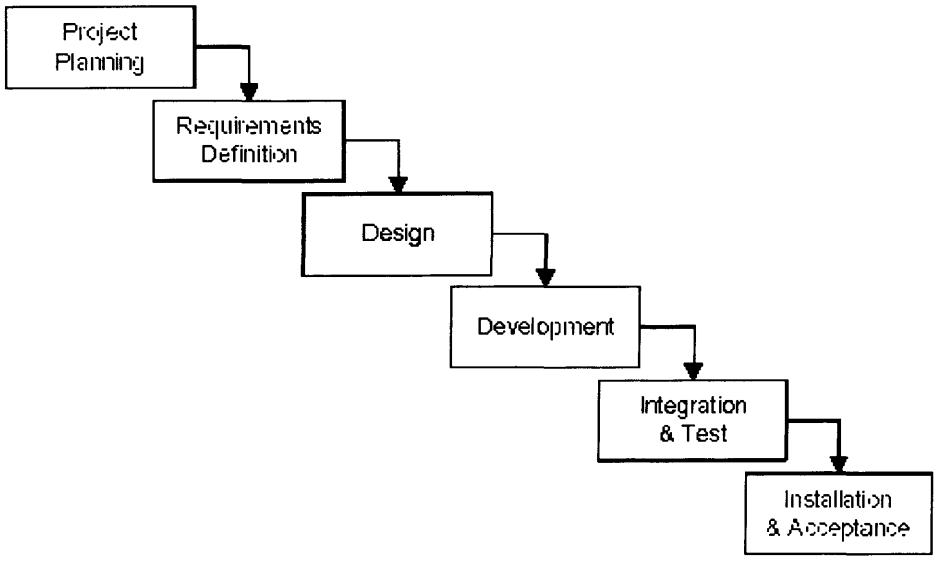


Fig. 5.1 The six stages involved in software projects development

5.2 Project planning of the GA optimisation software

In the planning stage, it was identified that the FEA program (OPERA-2D) could be run offline without user interaction by executing a readily prepared command, referred to as a control script. The control script contains details the type of analysis to be carried out for a particular model and the changes to be made to the configuration for each case, for

example different excitation, geometric modifications etc. Furthermore standard executable files (with extension liked .exe, .bat, and .com) could also be executed in OPERA-2D design environment if required. The openness of the OPERA-2D program allowed the software to more easily incorporate other design tools or analyses that are separately developed using other development programs.

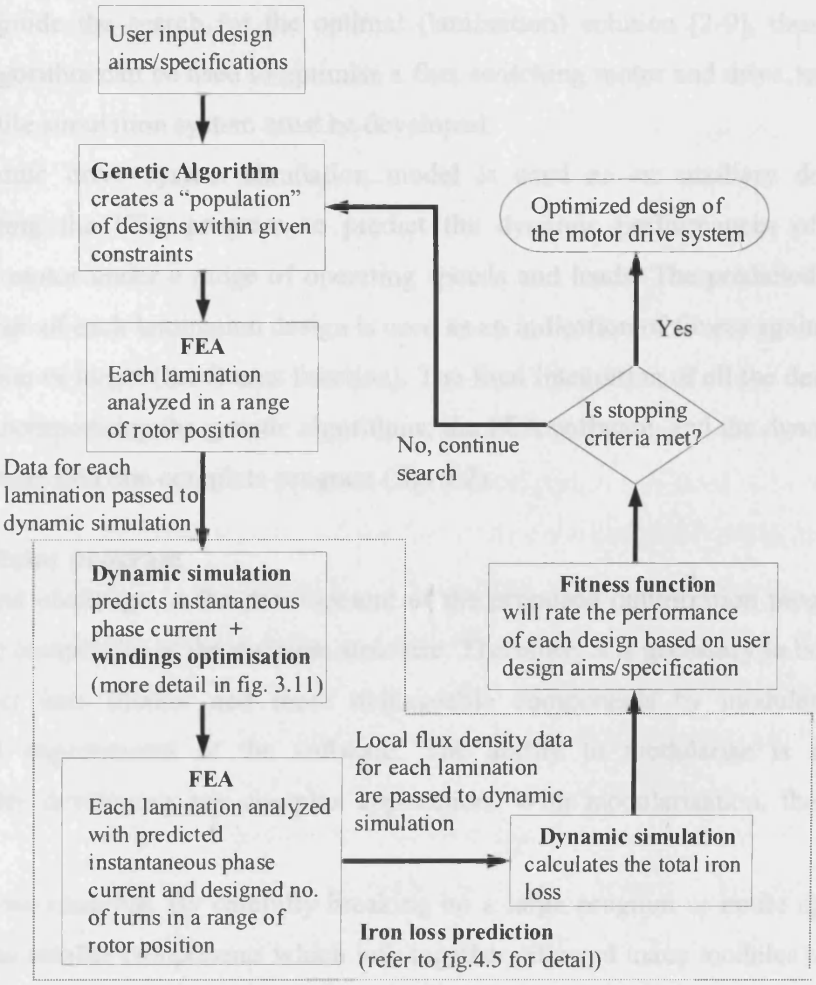


Fig. 5.2 The outline of the functional requirements of the optimization program for flux switching motor and drive.

As mentioned in the first chapter of the thesis, the FEA program is only a design tool for estimating electromagnetic performance of flux switching motor for a given configuration. On the whole, the improvement to the design was still dependent on the designer feedback to the program and such reliance of the designer interactions with the

software could be substituted with the Genetic Algorithm. This would require the development of a general-purpose genetic algorithm, which offers versatility and flexibility to its user, as well as its capability to interface with the FEA software. The interface of the genetic algorithms with OPERA-2D, allows multiple design variables of the flux switching motor to be tuned simultaneously, and produces a wide range of motor laminations. However, it is understood that the genetic algorithms used fitness value to guide the search for the optimal (lamination) solution [2-9], thus before a genetic algorithm can be used to optimise a flux switching motor and drive, an accurate and versatile simulation system must be developed.

The dynamic drive system simulation model is used as an auxiliary design tool incorporating the FEA program to predict the dynamic performances of the flux switching motor under a range of operating speeds and loads. The predicted motoring performance of each lamination design is used as an indication of fitness against a given specification or target (the fitness function). The final integration of all the design tools, involves incorporating the genetic algorithms, the FEA software, and the dynamic drive system model into one complete program (fig. 5.2).

5.2.1 Modular program

The biggest challenge in the development of the proposed optimization program is to reduce the complexity of the program structure. Therefore, it is necessary to break down the project into smaller and more manageable components by modularising the functional requirements of the software. The ability to modularise is central to successfully developing any complex application. With modularization, the program becomes:

- *More reusable.* By carefully breaking up a large program or entire application into smaller components which link together, allowed many modules to be used in more than one other program in the application.
- *More manageable.* This allowed the program to be tested and debugged on a smaller scale (unit test) before individual modules are combined for a more complicated system test.
- *More readable.* It is always easier to work with a smaller scale program, where each module is named and has its functions properly described. Thus making it easier it to read and understand what that program is doing.

- *More reliable.* The code produced will have fewer errors, also if errors are found it will be easier to fix because they will be isolated within a module. In addition, the code will be easier to maintain since there is less of it and it is more readable.

5.2.2 Problems in interfacing design tools built in different development environments

During the process of identifying the software functional requirements, it was established that the least technical demanding method of developing the optimization software would be program modularisation, by incorporating the OPERA-2D software with externally developed design tools. However, to integrate design tools developed using external development environments with the OPERA-2D software were not as simple as first thought. The incompatibility of source and target languages and the requirement for automatic file naming conventions were the problems identified that hindered a smooth integration of the optimization program. The possibilities of successfully integrating the individually developed design tools into a complete optimization program were largely dependent on resolving the two problems mentioned above. The structured information generated by each design tool need to be transferred in the correct format from one tool to another while a unique name has to be assigned for each data transfer. The uniqueness of the object names (used in every structural information transfer) was necessary to avoid unexpected effects such as replacing a previous object by a new one, and also in this way information generated by each design tool pertaining electromagnetic characteristic of a (lamination) design could be retained as a form of record. Therefore, information transfer would require the development of ad hoc transformations to interface the different design tools within the optimization program. In this case information transfer means transformation of the output of one design tool into the input of another design tool. These ad hoc transformations were built in the form of permanent interfacing mechanisms (executable files) between the different design tools.

5.3 Defining the program requirements of each modular program

This section defines how the software was developed from analysing the general requirements (or goals) of the program (fig. 5.2) to the identification of individual component programs (or modules as in fig. 5.3).

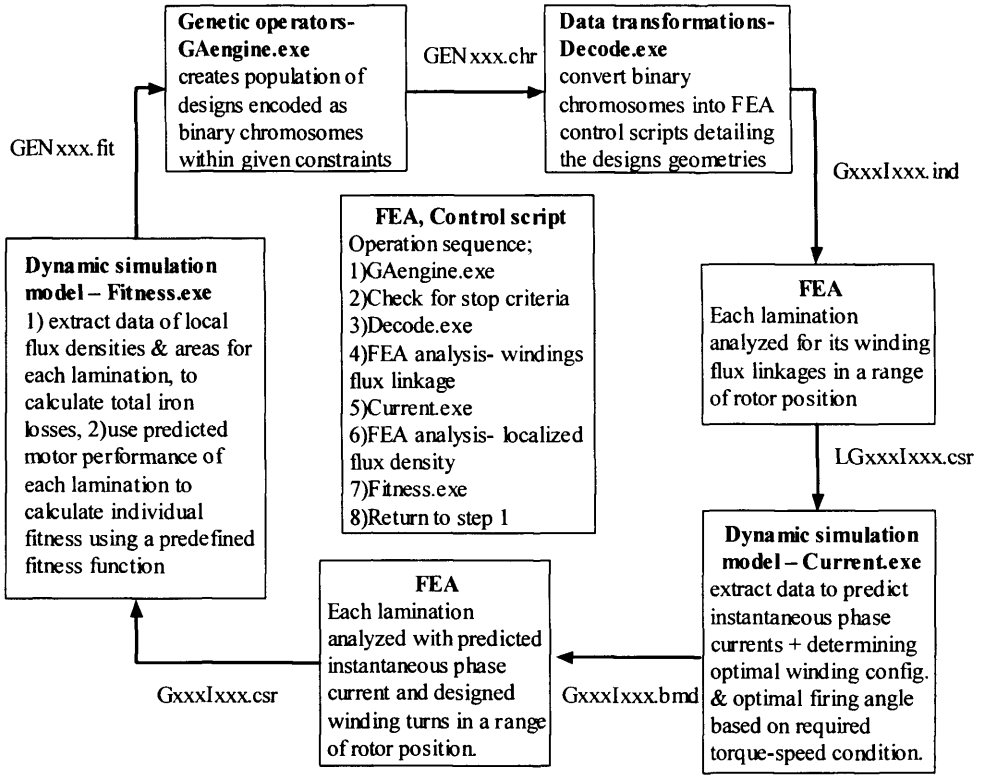


Fig. 5.3 Modularisation of the optimization program into five program modules

As shown in figure 5.3 the structural breakdown of the optimization software consists of five separate program modules, with the communication between them by means of formatted text files. The order of running the external program modules is managed by a Control script (written in OPERA scripting language) which can be loaded in the OPERA-2D Design Environment [20]. The script also details the operations in the FEA program, like simulating the rotation of the rotor; invoking the pre-processor to read the *.ind* files which contain the data of different motor lamination shapes (in the form of OPERA script language); calling the solver and post-processor to create the *.csr* files which contain the electromagnetic characteristics of the analysed lamination designs, etc. The external program modules are the genetic operators module which creates the population of designs by means of genetic operations, the decode module which converts binary chromosomes (encoded designs created by Gaengine.exe) into OPERA script for the input of the FEA program, the instantaneous current module for determining the required phase currents and number of windings turns to deliver the user specified torque for each lamination design (output in script language format), and the fitness module calculates the overall motor performance (including total iron losses)

of each lamination design these information were then used in a predefined fitness function to determine the fitness of each design.

The identified general requirements in each program module are refined into a set of more detailed requirements. These detailed requirements define the major functions of the intended application(s), operational data areas and reference data areas, and the initial data entities. Each detailed requirement contains a requirement title and is further elaborated in the form of textual and graphical descriptions. The following sub-sections would refine the set of more detailed requirements for each of the identified program module.

5.3.1 Defining program functionalities – Genetic operators module

Ref #	Function	Category
R1.1	Customising of the GA engine.	Evident
R1.2	Stored customised GA engine into a setup file.	Hidden
R1.3	Read customised GA engine from setup file.	Hidden
R1.4	Checked if number of generations has exceeded (only if selected by user).	Hidden
R1.5	Find most recent generation of system and the length of a chromosome.	Hidden
R1.6	Copy and ranked the latest generation of chromosomes based on their fitness.	Hidden
R1.7	Checked if convergence in the fitness has occurred (only if selected by user)	Hidden
R1.8	Selection by user choice	Hidden
R1.9	Crossover by user choice	Hidden
R1.10	Mutation by user choice	Hidden
R1.11	Replacement of weaker chromosome	Hidden
R1.12	Create new generation.	Hidden

Fig. 5.4 Table on functional specifications of GA operators module

R1.1 Customising of the GA engine

The function checks whether the genetic operators have been defined, the function would return a '1' or '0' to indicate if the parameters are already set. If the function returns '0', a set of questions would be asked, providing the user with choices of GA tools to be selected from. The user-selected genetic operators form the GA engine, and this customised GA engine is used to perform future GA operations, until the terminating criteria are met.

Here are the set of questions:

- Size of the population.
- Percentage of replacement for weak chromosomes.
- Type of optimization problem 1) Maximization, 2) Minimization.
- Method of selection 1) Rank order (Fit-fit), 2) Random order, 3) SUS.
- Method of crossover 1) Single-point, 2) Two-point.
- Method of mutation 1) Flip-bit, 2) Random.
- Method of termination 1) Fixed number generation, 2) Convergence.
- Defining of convergence criteria 1) the number of past generations to be compared with the current one, 2) the value of fitness threshold.

R1.2 Stored customised GA engine into a setup file

This function is only executed when the GA engine has not been setup previously. The function would compile all GA operators selected by user into a setup file. After this function is executed, the GA module terminates its operation.

R1.3 Read customised GA engine from setup file

This function would only be executed, if function R1.1 found that GA operators have already been defined and stored in a setup file. R1.3 would then read the stored information from the setup file and initialise the setup.

R1.4 Checked if number of generations has exceeded

This function is only activated when a fixed number of generation termination method was selected. The function would determine whether maximum generations have been reached, if so, the function would write a termination file. The termination file would give an indication to the system (control script) to stop all its operation. Thus the whole iteration ends.

R1.5 Find most recent generation of system and the length of a chromosome

This function would search for the latest generation in this iteration, and stores that information. The search starts from generation 1 until it finds the latest generation. A subsequent task is to verify the length of a chromosome in that generation. To find the length of the chromosome, a chromosome is copied, then the number of '1' and '0' within the chromosome is counted, and the information is stored.

R1.6 Copy and rank the latest generation of chromosomes based on their fitness

This function would copy all the strings of '1' and '0' with their assigned fitness value into an array, and sort the strings based on their fitness in descending order. The assigned fitness value of each individual gives an indication of how the strings should be sorted. For example in a maximization problem, individual A with fitness value 0.75 would be ranked below individual B with fitness value 0.99, and vice versa if the optimization problem is to minimise the fitness function.

R1.7 Checked if convergence in the fitness has occurred

Convergence function would check the maximum fitness value of current generation and do a comparison with maximum fitness value of previous generations. If this comparison shows that the maximum fitness value had not improved for a certain (user-defined) generations, the function should write a termination file. The termination file would give an indication to the system (control script) to stop all its operation. Thus the whole iteration ends.

R1.8 Selection by user choice

The method of selection is determined in function R1.1 when the user customised the choices of genetic operators. Only the chosen selection function would be activated. Here are the possibilities of selection:

- Rank-order (fit-fit) selection, the function would pair an individual with the next fittest individual in the population by stepping through the order of the list, and the process should stop when it reached the last pair of strings. The percentage of replacement set in function R1.1 would determine whether the selection has reached its last pair of individuals.
- The random selection, as suggested the function randomly selects any pair within a mating pool, where the size of the mating pool is determined by percentage of replacement and size of population. However to maintain the idea of population diversity, each individual (string) can only be picked once. Thus, a shuffling method was implemented to rearrange the order of the individuals. The process involved the random selection of two strings within the sorted population (according to their fitness values) and swapping their positions. This iterative process of strings swapping their positions is completed in 200 iterations and the process would disarrange the order of the fitness in the

population. After the shuffling process, the function would pair the first two strings in the mating pool, followed by the next two and so forth until the last of them.

- Stochastic Universal Sampling (SUS), the function would firstly calculate the sum of all individuals' fitness in the population, *S* and this step is similar to the roulette wheel selection (refer chapter 1, section 1.3.3.3). However, unlike the roulette wheel selection the SUS selects multiple individuals per spin, *N* (this value is preset in function R1.1) to prevent loss of population diversity. To determine the positions of the *N* pointers, a random number first has to be generated between the range of 0 and *S/N*. This randomly generated value would be the first pointer or the firstly selected individual while the subsequently selected individuals would all happen at an interval of *S/N* until the last individual is selected. Figure 5.5 shows an example of the SUS, where 6 individuals were being selected for the mating pool. After the placing all selected individuals into the mating pool, the function would pair the first two strings in the mating pool, followed by the next two and so forth until the last of them.

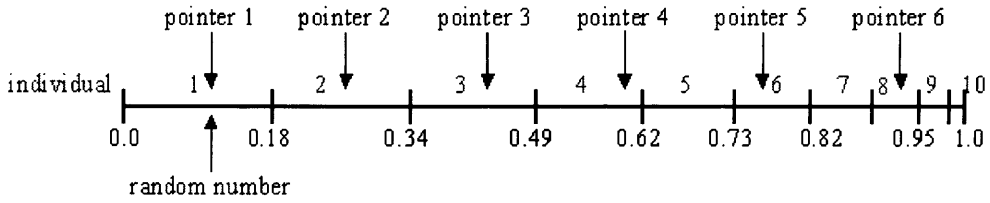


Fig. 5.5 An example of Stochastic Universal Sampling

R1.9 Crossover by user choice

The method of crossover is determined in function R1.1 when the user customised the choices of genetic operators. Only the chosen crossover function would be activated.

- Single-point crossover, the function begins by splicing two selected individuals at a random point. The splice-point is chosen randomly using a random number generator within the range of 1 to maximum string length-1. By keeping the head sub-string the same and exchanging the tail sub-strings, two new strings, each made up of "genetic information" from previous pair are generated. The iteration of swapping tail sub-string of two chromosomes ends when the last pair of individuals in the mating pool exchanged their "genetic information".

- Two-point crossover, the function firstly chooses two random number within the range of 1 to maximum string length-1, with the condition that value of first random number has to be less than that of the second one. These two randomly selected (integer) numbers then becomes the splicing points, where only "genetic information" that lay between them is exchanged. The iteration of swapping the middle section of two chromosomes ends when the last pair of individuals in the mating pool exchange their "genetic information".

R1.10 Mutation by user choice

The method of mutation is determined in function R1.1 when the user customised the choices of genetic operators. Only the chosen mutation function would be activated.

- Flip-bit mutation, the function randomly altered bits within the new strings, based on a mutation probability value, which dictates the frequency at which mutation occurs. For example if the mutation probability is set by the user to be 0.004, string length of each individual is make-up of 20 bits, and there are 100 individuals in a population. Based on the selected parameters, the function would randomly select eight bits out of the possible 2000 bits in each generation, and changed a '0' to '1' or vice versa of the selected bits.
- Random mutation, the function would randomly alteration of bits within the new strings, based on a mutation probability value. But this time the selected bit(s) should have a chance to remain unaltered or be inverted.

R1.11 Replacement of the weaker chromosomes

The function randomly generates a predetermined number of strings, containing '1' and '0' of a specific length to replace the strings, which didn't qualify for selection. The number of strings to be created would be determined by the value of "percentage of replacement", and length of the string is found in function R1.5.

R1.12 Create new generation

This function creates two files, first a file that contains all the new strings reproduced from the process of GA. Then followed by a file that held the information indicating to the system (control script) to continue the iteration.

5.3.2 Defining program functionalities- Decode module

Ref #	Function	Category
R2.1	Set design variables, constraints, etc.	Evident
R2.2	Stored design variables, constraints, etc into a setup file.	Hidden
R2.3	Read design variables, constraints, etc from setup file.	Hidden
R2.4	Create initial population.	Hidden
R2.5	Find most recent generation of system.	Hidden
R2.6	Conversion of binary coding into design variables.	Hidden
R2.7	Write design variables into OPERA script format for design analysis.	Hidden

Fig. 5.6 Table on functional specifications of the Decode module

R2.1 Set design variables, constraints, etc

This function checks whether the design variables, constraints, etc. have been defined by the user, if it does, it ends the function. Otherwise, a set of questions regarding the design would be asked, prompting the user to update the necessary data for design.

The set of questions is shown:

- Number of variables the user wants for the design.
- Number of bits in each variable.
- The upper and lower limit (constraints) for each variable.
- To label or name each variable.
- Assigned an expression code to each variable.
- Give a description to each variable.
- If user wants any additional information for each variable.
- Assigned an expression code to the additional information.

R2.2 Stored design variables, constraints, etc into a setup file

Function R2.2 is only executed, if the design variables, constraints, etc. had not previously been stored. The function would store all information entered by the user into a setup file.

R2.3 Read design variables, constraints, etc from setup file

This function would be executed, if function R2.1 found that design variables, constraints, etc. had already been defined and stored in a setup file. R2.3 would read the information from a setup file and store them in the memory.

R2.4 Create initial population

First the function would check for any existing generation in the system, if a previous generation existed the function ends. Else the function should randomly generate a predetermined number of strings containing '1' and '0' of a specific length. Information on number of strings to create and length of each string should be gathered from the setup files of Genetic operators module R.1 and Decode module R.2 respectively.

R2.5 Find most recent generation of system

The function would search for the latest generation in iteration, and stored that information. The search would start from generation 1 and increase to the next generation until it finds the latest generation.

R2.6 Conversion of binary coding into design variables

This function would open the latest generation of *.chr* file, reads in the first *chromosome* (string of binary numbers) in that generation, and then starts to decode each *gene* in the *chromosome* with the specified data (liked number of bits, upper limit, lower limit of each variable). The *chromosome* has now been converted into solution set, and this process starts with reading in *chromosome* (string) again. But this time, it would read the second string and decodes it, the iteration continues until the last string of the generation is decoded. In Fig.5.7 shows that each variable in the solution set corresponds to a gene in the chromosome.

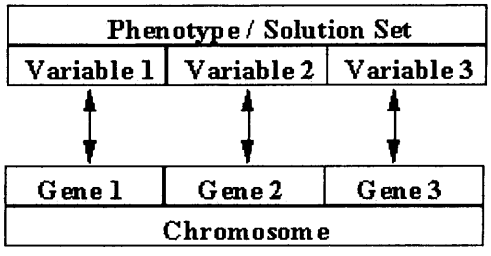


Fig.5.7 Diagram on relation between solution set and chromosome.

A *gene* is defined as a string of *N* bits. It is a binary representation of the number of intervals from lower limit. The design constraints set an upper and lower limit, without setting this constraint, searching the range from -∞ to ∞ would be impossible. This range between the upper and lower limit is divided into the number of intervals, as defined by the length of *gene*. A simple expression of the bit string of *N* length is (2^N-1), and the size of an interval would be (upper limit – lower limit)/ (2^N-1). Fig. 5.8 illustrates how the conversion from *chromosome* to solution set is being done.

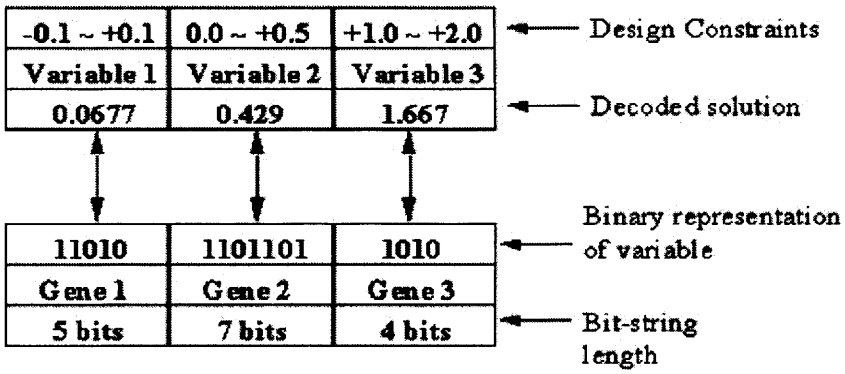


Fig.5.8 Illustration of conversion from chromosome to solution set.

For example:

Variable 1 has a design constraint of -0.1 to +0.1.

The bit string (gene) is 5 bits long.

The interval is 6.45×10^{-3}

The binary number '11010' is 26 in decimal.

While bit string is '11010', the value is $[\{(0.1+0.1) / (2^5-1)\} * 26] - 0.1 = 0.0677$

Therefore, general equation for converting chromosome to solution set is,

$[\{(\text{upper limit} - \text{lower limit})/ (2^N-1)\} * \text{gene}] + \text{lower limit}$

R2.7 Write design variables into OPERA script format for design analysis

The function would transform the decoded design variables into corresponding OPERA script format. The format is similar to figure 5.9 with a description of the variable, an address of the variable, a value to which the variable is to be set, and finished with an indicator, '|'. This function allowed as many design variables as there are in a particular design model to be written in the configuration script.

```
/Armature slot      <-- Descriptor
RUNP slot_2         <-- Variable name
RUNP PA01=0.56      <-- Expression
|                   <-- End indicator

RUNP slot_2
RUNP PA02=0.56
|
```

Fig.5.9 An example of input script format of the OPERA-2D software

5.3.3 Defining program functionalities – Instantaneous current module

Ref #	Function	Category
R3.1	Find the most recent generation of the system.	Hidden
R3.2	Determine the number of designs in the current generation.	Hidden
R3.3	Extract flux linkage information from *.csr* file of each analysed lamination design.	Hidden
R3.4	Start the dynamic simulation model...	Evident/ Hidden
R3.5	Activate the winding optimisation model ...	Evident/ Hidden
R3.6	Write instantaneous phase current data and optimal winding configuration into OPERA script format for design analysis.	Hidden

Fig. 5.10 Table on functional specifications of the instantaneous current module

R3.1 Find most recent generation of system

The function would search for the latest generation of design in the design cycle, and stored that information. The search would start from generation 1 and increase to the next generation until it finds the latest generation.

R3.2 Determine the number of designs in the current generation

The function would check the total number of result files created by OPERA-2D in the current generation, and store that information. Each result file contains the windings flux linkage information of a lamination design.

R3.3 Extract flux linkage information from .csr file of each analysed lamination design

The function would open the *.csr* file of the most recent generation extracts all the meaningful numerical data and store them in an array. The function reads only one *.csr* file at each time, starting with *LGxxxI001.csr* (the first lamination design file of most recent generation), and the process finishes when all necessary data in each design file within the generation are extracted.

R3.4 Start the dynamic simulation model...

The function would first input the extracted flux linkage data into the dynamic drive system model then manipulate the model to calculate the instantaneous phase current and other related motor performance without considering the iron loss using the default winding and firing angle settings.

R3.5 Activate the winding optimisation model...

The function manipulates the winding optimisation model built within the dynamic simulation model to achieve required speed-torque specification with best possible efficiency (refer to section 3.5 for details).

R3.6 Write instantaneous phase current data and optimal winding configuration into OPERA script format for design analysis

The function would transform the simulation data of instantaneous phase currents and optimal winding configuration into corresponding OPERA script format. The format is similar to figure 5.11, where the (FEA pre-processor) instructions of the motor instantaneous phase currents and winding turns at 20 discrete equally spaced rotor positions are logged in *.bmd* file. Useful information of selected wire sizes and firing angle but had no meanings in the FEA environment are included as comments.

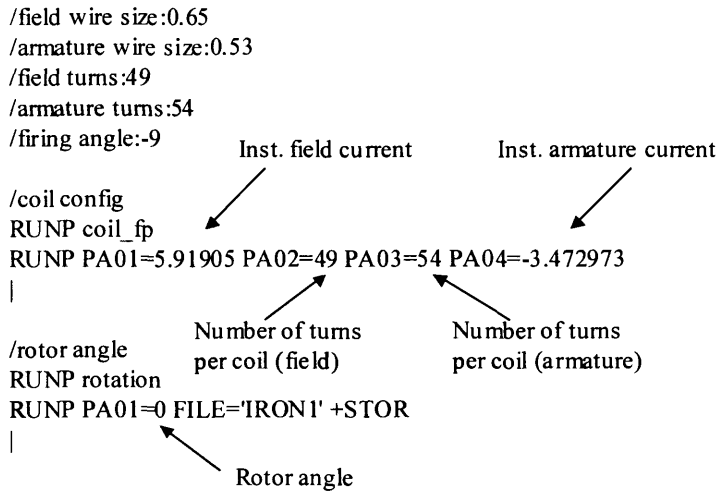


Fig.5.11 Another example of input script format of the OPERA-2D software

5.3.4 Defining program functionalities – Fitness module

Ref #	Function	Category
R4.1	Find the most recent generation of the system.	Hidden
R4.2	Determine the number of designs in the current generation.	Hidden
R4.3	Extract flux linkage information from *.csr* file of each analysed lamination design.	Hidden
R4.4	Extract local flux densities and areas data from *.csr* file of each analysed lamination design.	Hidden
R4.5	Extract optimal windings configuration and firing angle information from each *.bmd* file.	Hidden
R4.6	Start the dynamic simulation model, input all extracted data and calculate motor performance of each lamination design, derive the fitness value of each design using the predefined fitness function.	Evident/ Hidden
R4.7	Write individual designs in the form of binary chromosomes and their assigned fitness values into a *.fit* file.	Hidden

Fig. 5.12 Table on functional specifications of the fitness module

R4.1 Find most recent generation of system

The function would search for the latest generation of design in the design cycle, and stored that information. The search would start from generation 1 and increase to the next generation until it finds the latest generation.

R4.2 Determine the number of designs in the current generation

The function would check the total number of result files created by OPERA-2D in current generation, and stored that information. Each result file contains the windings flux linkage information of a lamination design.

R4.3 Extract flux linkage information from .csr file of each analysed lamination design

The function would open the *.csr* file of the most recent generation extracts the flux linkage vs. rotor position data and store them in an array. While only one *.csr* file is opened each time starting from the first file of most recent generation, *LGxxxI001.csr* and until the last file of the generation.

R4.4 Extract local flux densities and areas data from .csr file of each analysed lamination design

The function would open the *.csr* file of the most recent generation pertaining the information of local flux densities and areas extracts the data and store them in an array. Only one *.csr* file is opened each time starting from the first file of most recent generation, *GxxxI001.csr* and until the last file of the generation.

R4.5 Extract optimal windings configuration and firing angle information from each .bmd file

The function would open the *.bmd* files of the most recent generation (but one at each time), extracts the data pertaining the information of the optimal windings configurations and firing angles. Each .bmd file contains information of the optimal windings configuration and firing angle of a particular lamination designed to best achieve design objective(s).

R4.6 Start the dynamic simulation model ... derive the fitness value of each design using the predefined fitness function

The function would first input the all extracted data from functions R4.4 to R4.6 into the dynamic drive system model then manipulates the model to calculate individual lamination design motoring performances like efficiency, total copper and iron losses, average torque etc. The function would then use the information to derive the fitness of each individual design using the preset *fitness function*.

R4.7 Write individual designs in the form of binary chromosomes and their assigned fitness values into a .fit file

The function would append a fitness value to each lamination design determined in function R4.6 while each lamination design would be represented in the form of a binary chromosome. For example, lamination design 1 of generation 1, G001I001 before it is being decoded into OPERA scripting language takes the form of string '100101111' and has a fitness value of 0.987. So the appended string would look like 100101111, 0.987. All the appended strings would then be written into a *GENxxx.fit* file (fitness file), the fitness file should have all the strings with assigned fitness value of that generation.

5.3.5 Defining program functionalities – Control script module

Ref #	Function	Category
R5.1	Determine the most recent generation of the system and introduce an appropriate start-up strategy.	Hidden
R5.2	Determine the number of designs in the current generation.	Hidden
R5.3	Execute the genetic operators module- *GAengine.exe*	Hidden
R5.4	Interpret instruction from GAengine.exe to determine if stopping criteria has been met	Hidden
R5.5	Execute the data transformation module- *Decode.exe*	Hidden
R5.6	Analyse each lamination design for winding flux linkages in a range of rotor positions.	Evident
R5.7	Execute *Current.exe* to activate the dynamic simulation model and winding optimisation model.	Evident/ Hidden
R5.8	Analyse each lamination design with predicted instantaneous phase current and designed winding turns in a range of rotor positions.	Evident
R5.9	Execute *Fitness.exe* to activate the dynamic simulation model, iron loss model and fitness function.	Evident/ Hidden

Fig. 5.13 Table on functional specifications of the control script module

R5.1 Determine the most recent generation of the system and introduce an appropriate start-up strategy

The function would search for the latest generation of design in the design cycle, and store that information. The search would start from generation 1 until it finds the latest generation. This information would be used as a form of guidance to allow system to operate in the correct sequence; as the design process can either start afresh or restart from where it last left off, the different scenarios would require a slightly different operation, like the sequences of calling up the various program modules.

R5.2 Determine the number of designs in the current generation

The function would determine the total number of *.ind* files created by *decode.exe* in current generation, and stored that information. This information would provide the system the knowledge of the numbers of design laminations that required to be analysed in the current generation.

R5.3 Execute the genetic operators module- GAengine.exe

The function passes two instructions one to execute *GAengine.exe* file and the other to the OPERA-2D Design Environment to hold all proceedings till *GAengine.exe* finishes its operation.

R5.4 Interpret instruction from GAengine.exe to determine if stopping criteria has been met

The function would read a text file named, *finished* where the context of the file would determine if the system would continue its search for better design solutions or terminate the design cycle. The context can be simply in the terms of a '0' or a '1', where '0' means to continue search and '1' to end the search. The decision of continuing or ending the design and optimisation cycle is made by the genetic operators module- *GAengine.exe.*

R5.5 Execute the data transformation module- Decode.exe

The function passes two instructions one to execute *Decode.exe* file and the other to the OPERA-2D Design Environment to hold all proceedings till *Decode.exe* finishes its operation.

R5.6 Analyse each lamination design for winding flux linkages in a range of rotor positions

The function would load an *.ind* file to obtain the motor lamination shape, and then stores the data of the lamination at a range of rotor positions with prefixed field winding excitation. The FEA post processor then calculates and writes the results of the winding flux linkages at each rotor position into a *.csr* file, this process would repeat till all design laminations have been analysed. The post processing commands of the parameterised model would determine what information is written to the *.csr* format files and they have to be predetermined in the model.

R5.7 Execute Current.exe

The function passes two instructions one to execute *Current.exe* file and the other to the OPERA-2D Design Environment to hold all proceedings till *Current.exe* finishes its operation.

R5.8 Analyse each lamination design with predicted instantaneous phase current and designed winding turns in a range of rotor position

The function would load an *.ind* file to obtain the motor lamination shape, and then update the parameterised model with motor instantaneous phase current relative to a rotor position and its winding turns from a *.bmd* file. All information is stored for further analyses using the FEA post processor, the post-processing results of the localised flux densities and areas are then logged on a *.csr* format file with its unique name, this process would repeat till all design laminations have been analysed.

R5.9 Execute Fitness.exe

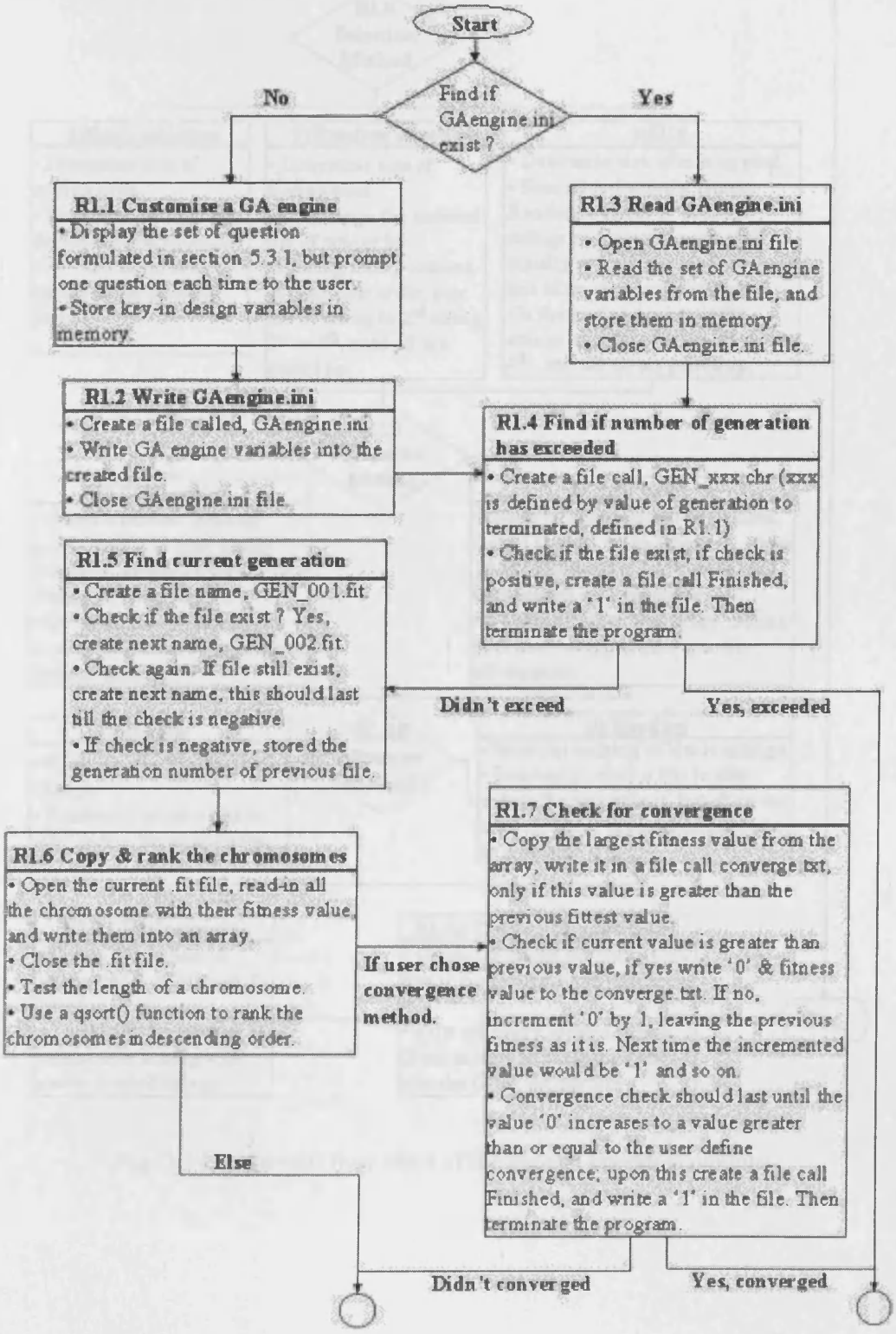
The function passes two instructions one to execute *Fitness.exe* file and the other to the OPERA-2D Design Environment to hold all proceedings till *Fitness.exe* finishes its operation.

5.4 Designing the optimization software

This section presents the design structure of the optimization software using several sequential flow charts, where each flow chart describes the sequential and logical appearance of a program module. The flow charts are used as pre-coding tools to help

the visualisation of the logical structure of each modular program and its required program functionalities; it gives a clear map out of how the program should be coded.

5.4.1 Designing of genetic operators module



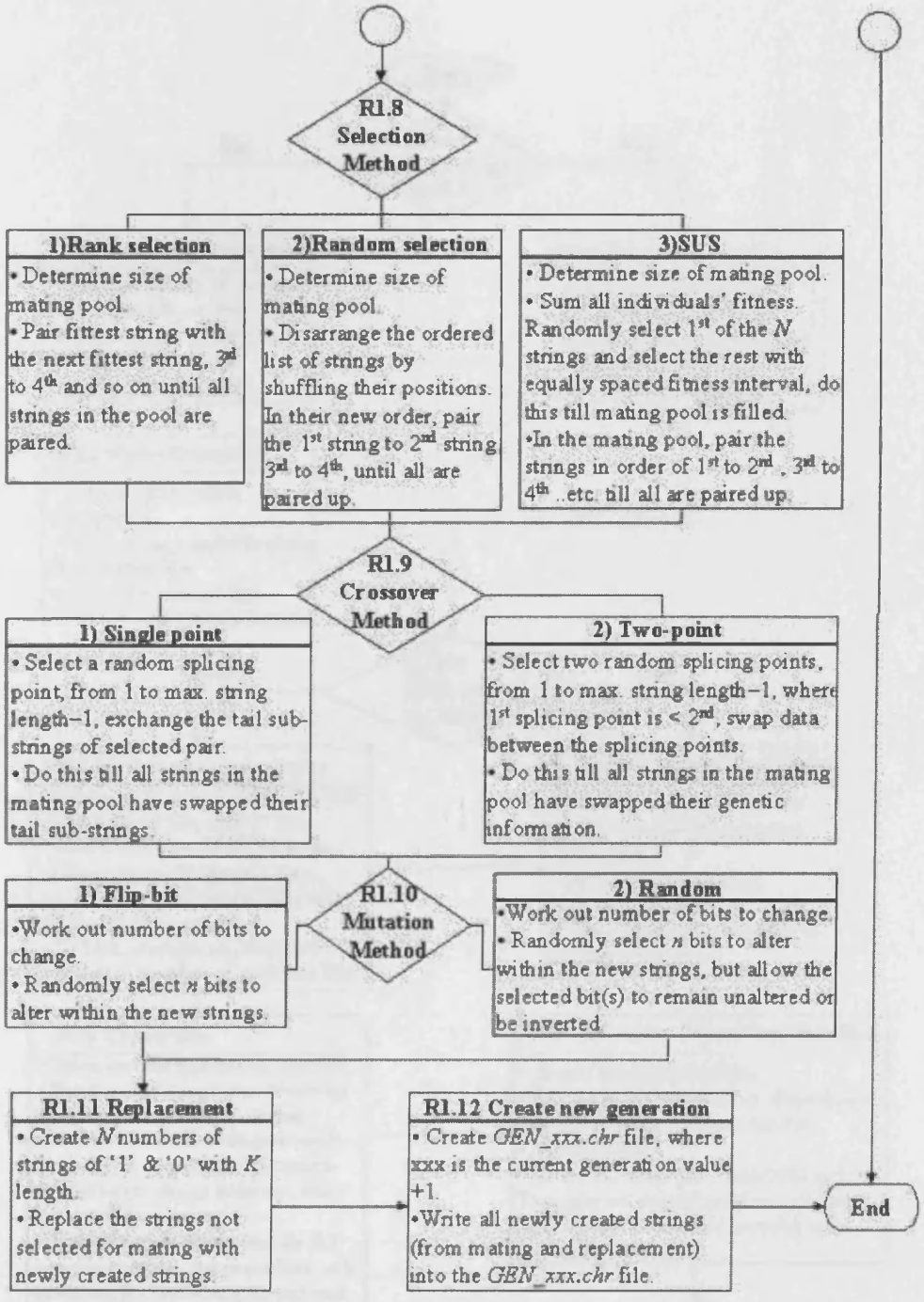


Fig. 5.14 Sequential flow chart of the genetic operators module

5.4.2 Designing of decode module

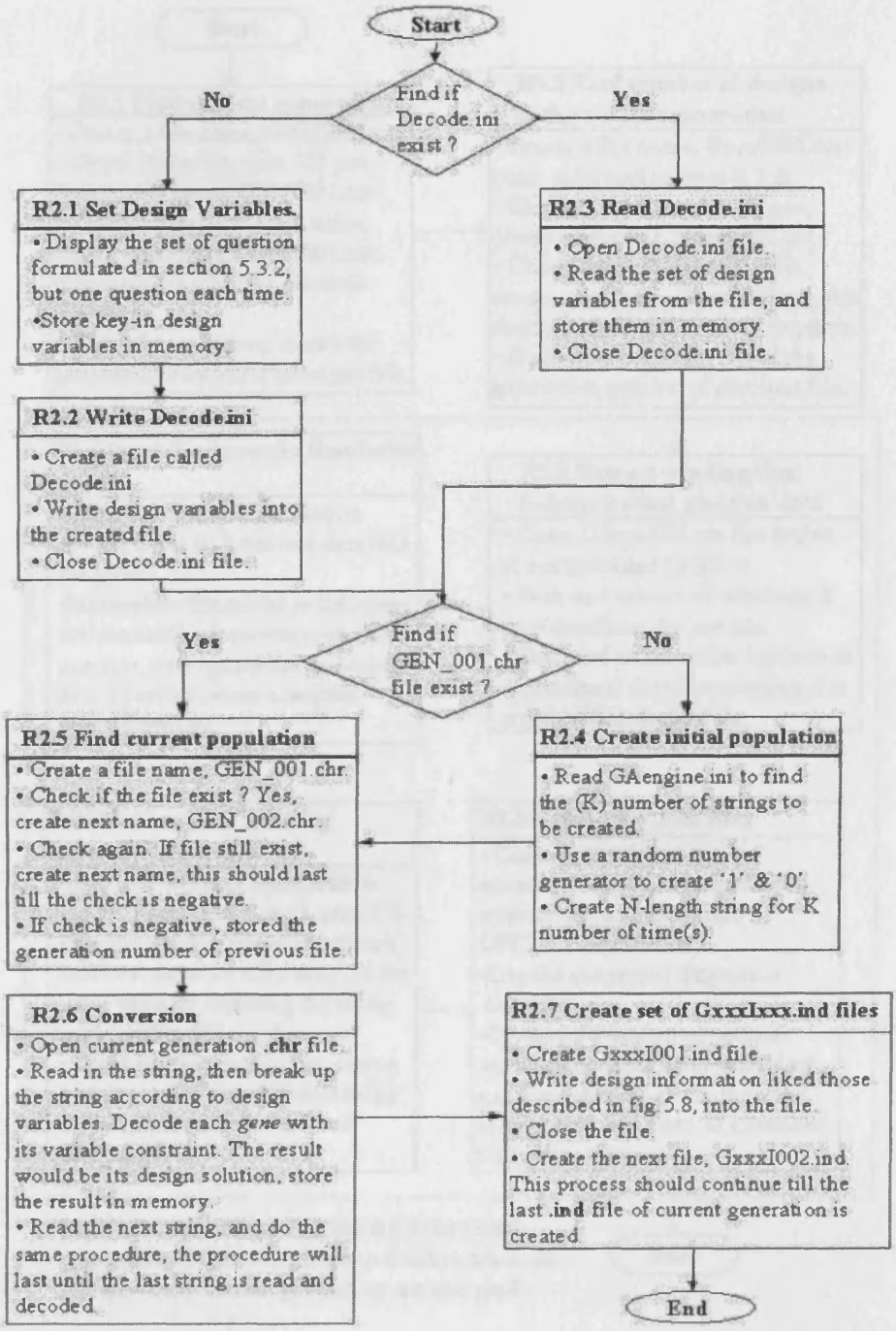


Fig. 5.15 Sequential flow chart of the decode module

5.4.3 Designing of instantaneous current module

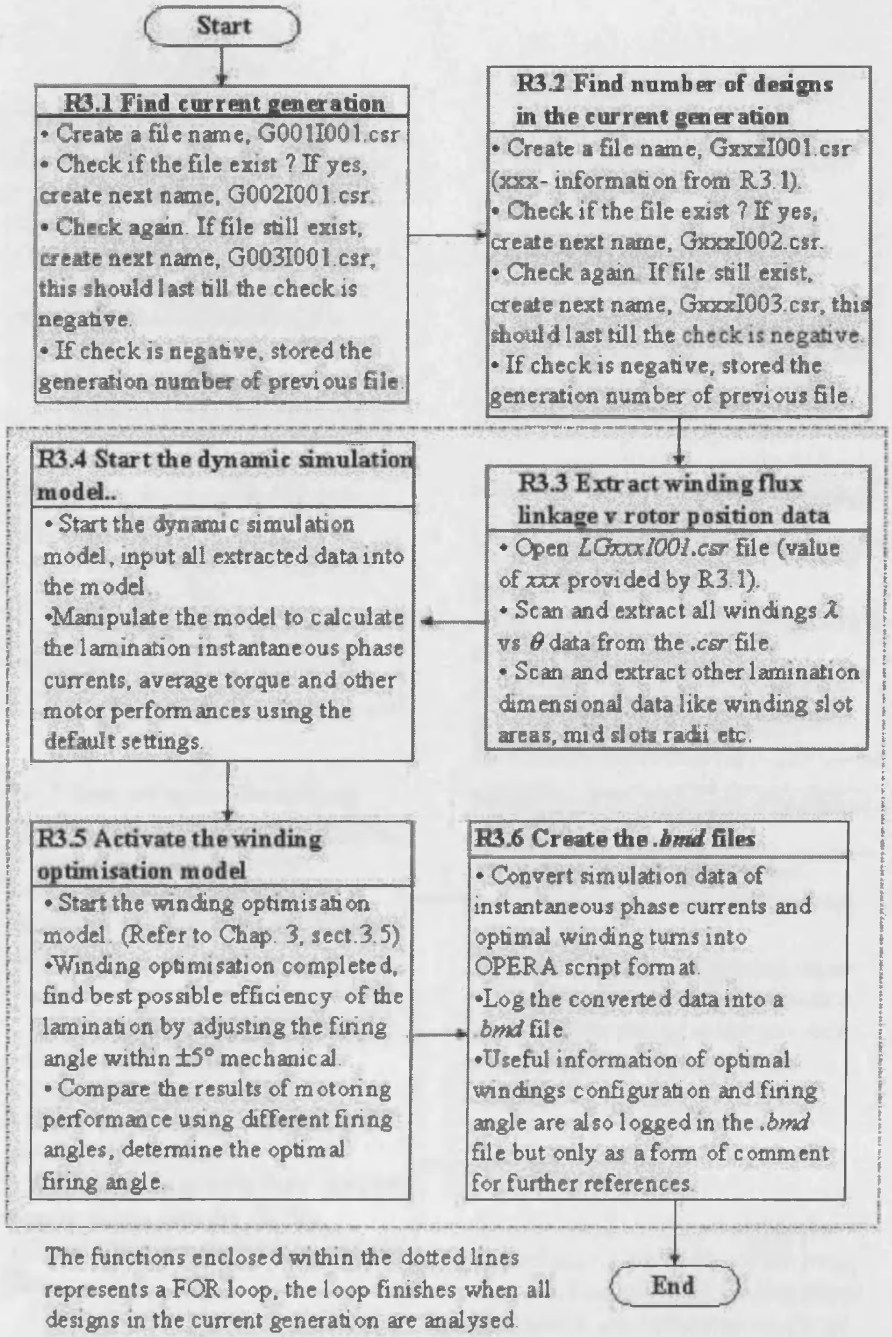


Fig. 5.16 Sequential flow chart of the instantaneous current module

5.4.4 Designing of fitness module

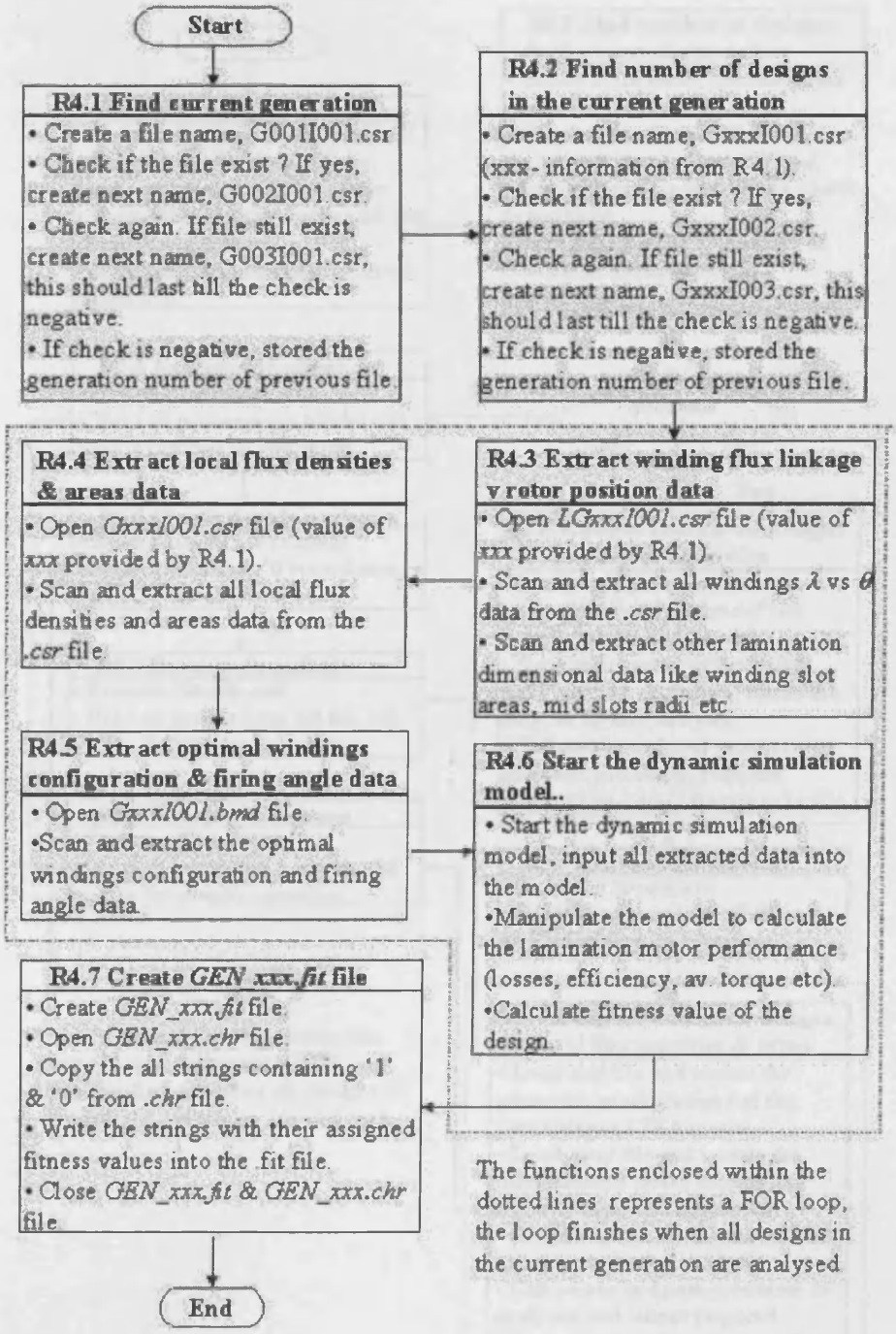


Fig. 5.17 Sequential flow chart of the fitness module

5.4.5 Designing of control script module

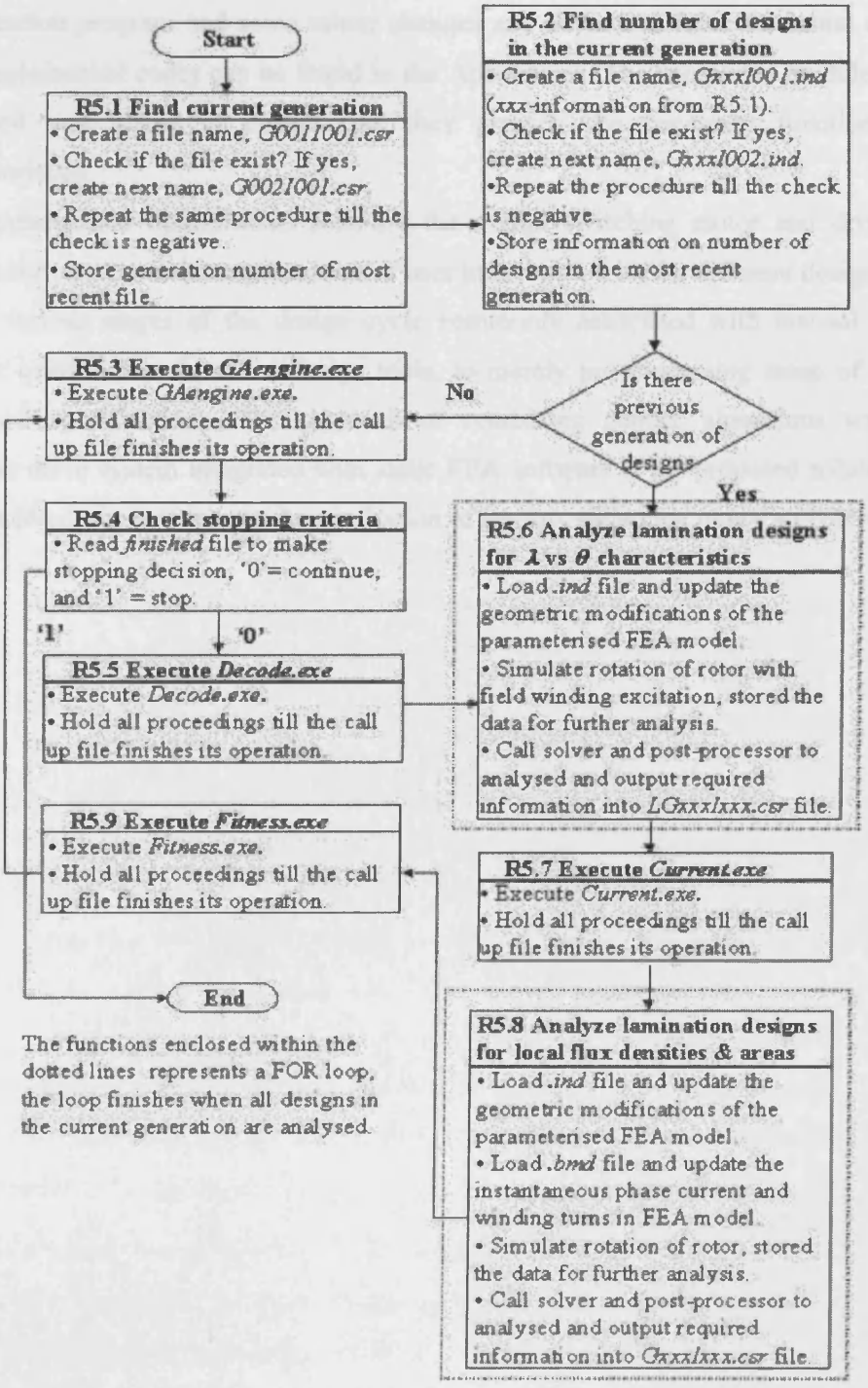


Fig. 5.18 Sequential flow chart of the control script module

A significant amount of decision making and creative work were accomplished during the analysis and design phases. From here onwards generation of the code was a relatively straightforward translation process. However, the implemented codes for the optimization program had some minor changes and deviations from the initial design. The implemented codes can be found in the Appendices. The completed modules were reviewed and tested to ensure that they provide the necessary functions and characteristics.

The implemented optimization software for a flux switching motor and drive had significantly reduced the requirements of user interaction with the different design tools, at the various stages of the design cycle commonly associated with manual design process using computer aided design tools, to merely pre processing setup of design aims and specifications. The approach of combining genetic algorithms with the dynamic drive system integrated with static FEA software is the proposed solution for unmanned computer-automated optimization of the flux switching motor and drive.

6 FSM drive system optimisation for high speed application using a genetic algorithm

6.1 Introduction

The design optimisation of a flux switching motor and drive system for a given specification is a multi-dimensional problem. Achieving the design objectives often requires tens or even hundreds of mechanical and electrical parameters to be tuned, and the strong interdependencies of the various parameters on each other and on performance made the optimisation problem virtually unsolvable by analytical techniques. Furthermore, the flux switching motor technology is still a considerably new concept, where there is not a large body of design experience to guide the designer. Therefore, the proposed design and optimisation software for the flux switching motor has been used to design a flux switching motor for high speed application without human intervention. The capability of accurately predicting dynamic performances of the full system (motor and drive) allows the genetic algorithms to confidently adjust the design within reasonable boundaries while steadily improving the fitness of the designs. The motor and drive has been constructed and test results have verified the accuracy of the simulation.

6.2 Defining the optimisation problem of an 8/4 FSM drive system

Traditionally, the design process commonly employed by engineers on electric machine design involved the procedures shown in figure 6.1. Starting with defining the design aims and specifications, while several possible solutions will be identified by the engineer(s) to achieve the design aims. During the implementation stage the best available design tools are used firstly to gain a thorough understanding of the design problem and then manually optimise the design. Usually, several software tools will be used at this stage, ranging from simple spreadsheet programs to sophisticated analytical and numerical CAD packages.

The same initial design procedures of defining the design aims/specifications and identification of possible solutions would apply when designing a flux switching motor using the proposed software, while the manual design process during the implementation stage is being automated with the optimisation program. With a clear definition of the design aims and specifications with possible design solutions allowed

the optimisation problem to be mapped out as a multi-dimensional problem space in which the genetic algorithms thrive in finding the optimal solution.

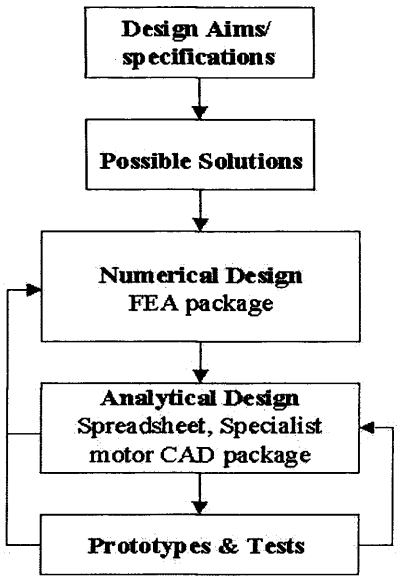


Fig. 6.1 Flow diagram of electric machine design process commonly used by engineers

6.2.1 Design aims and specifications

The design aims and specifications in this optimization problem of an 8/4 Flux Switching motor for high speed application are:

1) To deliver output power of 400W at 8000rpm, using 220-240V ac supply.
2) To maximize the efficiency of the FS motor while still achieving specification 1.
3) All specified motor dimensions are to be complied with; stator outside diameter of 87.5mm, with 35mm stack length and the air-gap length is 0.5mm.

6.2.2 Identification of the motor parameters to be varied

The identification of possible solutions is an important part of the design process. The number of suitable solutions identified depends largely upon the ingenuity and experience of the design engineer(s). As the FS motor is a new class of electric motor, which has not been widely documented there is little background material for this design problem. For the design aims described above the following possible design variations have been identified, and they are shown in Table 6.1 the parameters variation are illustrated in figs. 6.2 & 6.3.

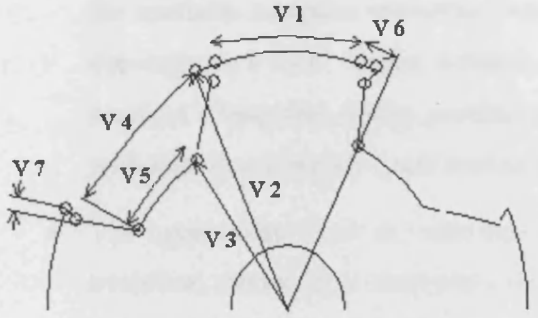


Fig. 6.2 Design parameters controlling the rotor shape

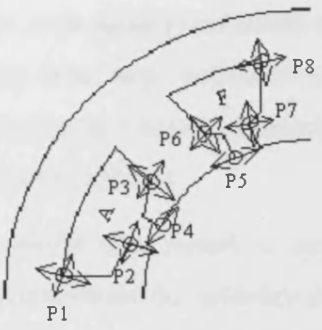


Fig. 6.3 The design variables in the FSM stator

Design Parameter	Mapping range		
	Lower limit	Upper limit	Encoded bits
Width at bottom of armature slot (P1)	0.75	0.5625	4
Width at middle of armature slot (P3)	0.7	0.5125	4
Width at armature slot entry (P2)	0.4	0.1	4
Armature slot gap (P4)	0.4	0.1	4
Field slot gap (P5)	0.4	0.1	4
Width at middle of field slot (P6)	0.5	0.3125	4
Width at field slot entry (P7)	0.4	0.1	4
Width at bottom of field slot (P8)	0.55	0.3625	4
Rotor arc, as a fraction of slot pitch of 90° (V1)	0.5	0.35	4
Rotor outer diameter (V2)	57	50	3
Rotor inner diameter (V3), as a fraction to rotor OD	0.7	0.625	2

Table 6.1 Design variables with their given dimensional limits and resolutions

6.2.3 Selection of genetic operators

Finding a good combinational of genetic operators that enable a successful evolution (eventually leading to locating the global optima) is not a trivial task. Robust GA (operators) settings had to be found for population size, number of generations, selection method, crossover method, frequency of mutation, while all these are done to ensure a good balance between quality of search results and convergence time.

- The size of the population is one of the keys that governs the quality of the evolution, and this is dependent on the bit string length used to represent the given problem, the number of generations to run the simulation and ultimately

the available computer resources. Normally GA with small populations tends to converge to a local optima solution, while GA with large populations would required a long time (many generations) to converge to a region of search space with significant improvement (not necessarily global optima).

- The appropriate level of selection pressure would also ensure a successful evolution, the key is to maintain a proper balance between the selection pressure and population diversity. However, level of selection pressure to apply is also subjected to the population size.
- In artificial system, crossover is the primary tool for the search and recombination of extant notions, however, occasionally this may become overzealous and lose some potentially useful genetic material ('1's or '0's at particular locations). The mutation operator is used to protect against such irrecoverable loss. When used sparingly with crossover it is an insurance policy against premature loss of important notions. In empirical genetic algorithm studies, the frequency of mutation that obtained good results is on the order of one mutation per thousand bits [8,10].

In reference [3], a genetic algorithm is employed to optimise of an induction machine and it has demonstrated good results. The genetic algorithm initially creates 200 machine designs in an initial population and the design iteration is terminated after 20 generations. A large population size is used to ensure possible design solutions are distributed over the large design problem space represented by 220 bits. The algorithm uses roulette wheel selection, single point crossover, mutation rate of 0.005 and elitism. The simulation was run on eight separate occasions using the same setting, and the results all demonstrated very similar and consistent outcome.

In this optimisation problem, a desktop computer with Pentium 700MHz processor was used to run the developed optimisation software, and the FEM analyses constitute up to 95% of the total computation time. Each lamination design takes roughly 45 minutes to be completely analysed at all positions for its motor performance and be appraised for its fitness. The long computational time required for each lamination and winding evaluation has posed a problem for using a large population GA (over 50 lamination design for each generation). As it has been acknowledged that traditionally genetic algorithms worked best using large population. Thus, the knowledge of selecting an

appropriate genetic algorithm that manipulates small population size and still yields good results has to be drawn from past experiences.

In reference [2] optimisation of a PM brushless DC motor using genetic algorithms has been reported. The algorithms were examined with different population size (at 25, 50 & 100 individuals), and mutation rate (0.01, 0.05) in the experiments. For all the simulations, the algorithm uses 60 bits to represent the problem space, stochastic sampling (aka roulette wheel selection), single point crossover, bit mutation and the design iteration terminates after 50 generations. The optimal solution is found using an algorithm that has the followings: 1) population size of 100, and 2) mutation rate of 0.05. However, it is worth to mention that the solution found using a smaller population size of 25 individuals, and mutation rate of 0.01 yields also a good result compared to the optimal solution.

Also in the same reference, a genetic algorithm that uses small population size of 25 individuals with mutation rate of 0.01 has been used in a simple problem with only two motor design variables. This experiment is used to observe the nature operation of a genetic algorithm. The 25 individual designs in the initial population are randomly chosen, and they are distributed over a given design problem space (in feasible region and also impractical one) specified by the two design variables (while others are set to be constants). It was reported that the individuals gradually moved away from the impractical region as the generation progresses. After 20 generations 80% of the population is in the feasible region, and a few of the individuals actually have the (two) design variables very close to the optimal design found earlier on. At 50^{th} generation, about 75% of the population has converged to results very close to the optimal one.

Using the information in references [2,3], a more unconventional GA with small populations and moderate number of generations was implemented, and as much effort was taken to balance between the selection pressure and population diversity. The following shows the genetic algorithm that has been implemented for the problem mentioned above.

- Population size of 20 individuals.
- Seeded initial population (refer to section 6.3).
- 20% replacement rate (replacing 4 least fit individuals within the generation with 4 new randomly generated individuals).

- Rank (Fit-fit) selection
- Uniform single point crossover
- Flip-bit mutation with frequency of mutation set at 0.004 (four mutations per thousand bits).
- Termination after 20 generations.

6.2.4 Defining the fitness function

The fitness function should express how good a design lamination is from the point of view of satisfying all the defined design aims and specifications. In this optimisation problem the fitness function is the motor efficiency at 8,000rpm producing 0.5Nm.

6.3 Optimisation of the 8/4 FSM drive system using the GAs driven optimisation software

The execution of the program starts with the specification of 1) the population size, 2) the GA operators (eg. replacement, method of crossover, rate of mutation and method of selection, choices of termination), 3) the constraints of the design limits, and 4) the objective function. In this experiment, the initial population consists of 20 (individuals) lamination designs. The algorithm uses binary representation, each (chromosome) design consists of 11 genes representing randomly selected values for the 11 design variables (refer to Table 6.1), with each gene is made up of two to four bits. The genetic algorithm is formed with the following genetic operators, rank (fit-fit) selection, uniform single-point crossover, flip-bit mutation, replacement, and predefined generation termination. Each set of design variables is passed to the Design Environment to create the finite element model and the design is first evaluated for its electromagnetic performance under static conditions. The static electromagnetic solutions of each design are used to obtain an initial dynamic performance (unaccounted for the core loss), these initial evaluated results are then used as a guide for the design of the winding configuration, with the optimised winding configuration determined. The core loss is then estimated for every individual of the population at the target operating point. At this point, the fitness value of each design will be calculated based on the user defined objective function. In this design exercise, the motor efficiency at 8,000rpm producing 0.5Nm was selected as the fitness function, the optimization problem is to maximise the motor efficiency while constrained by the specified speed-load

requirement. The calculated fitness values would be assigned to the corresponding designs, and based on the fitness information of each design, the genetic algorithm can perform its genetic operation to produce a new and improved generation of designs. The algorithm terminates after 20 generations. At this point, the designer can view the performance characteristics of the proposed solutions. Values for the eleven design variables and values of winding configuration chosen for each design can be found in case files with their unique names, for example, the design variables of design 1 of generation 1 are stored in *G001I001.ind*, and the values of the winding configuration are in *G001I001.bmd*. The naming convention can be interpreted as such, the number after the letter '*G*' presents the generation in which the file is created, the number after letter '*I*' is the design number with the total number of designs in a generation being determined by the population size, and the extension of the file details what information it contains.

6.3.1 Simulation results

The evolution of the designs is being illustrated in fig. 6.4 and fig. 6.5 demonstrating how the genetic algorithm starts its search for better designs from a feasible region as well as from the impractical ones. In generation 15, the genetic algorithm found the solution with the highest fitness value assigned amongst all other designs in the generations, noticed that most of the design structure of the best solution in generation 1 has been retained in generation 15, while some features have been modified. It is important to highlight that the best solution in generation 1 is not a randomly generated design, it is purposely inserted (as a seed) to the other randomly generated designs to avoid the possibility of all solutions starting in the impractical region, as there is only a small initial population of 20 individuals. As the designs evolved through the generations, transformation in the seed design had occurred through changes in lamination shape as illustrated in fig. 6.4 while the specific change in the design variables in term of their dimensional limits are shown in Table 6.2. The optimised design (as shown in fig. 6.4b) had the following changes compared to the initial one (fig 6.4a):

- Stator pole width and pole arc of the optimised design are wider.
- The field and armature slot opening of the optimal design are reduced.
- The rotor outer diameter has reduced.

- The optimal design had wider rotor pole arc with a reduced graded air-gap at leading edge.

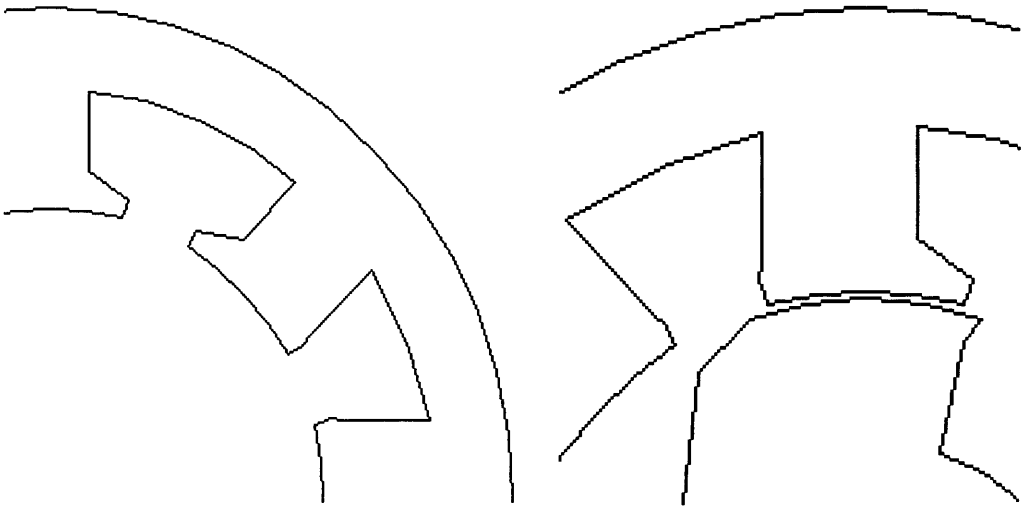


Fig. 6.4(a) Seed design inserted in generation 1 - illustration of zoom in view of the rotor and stator lamination design

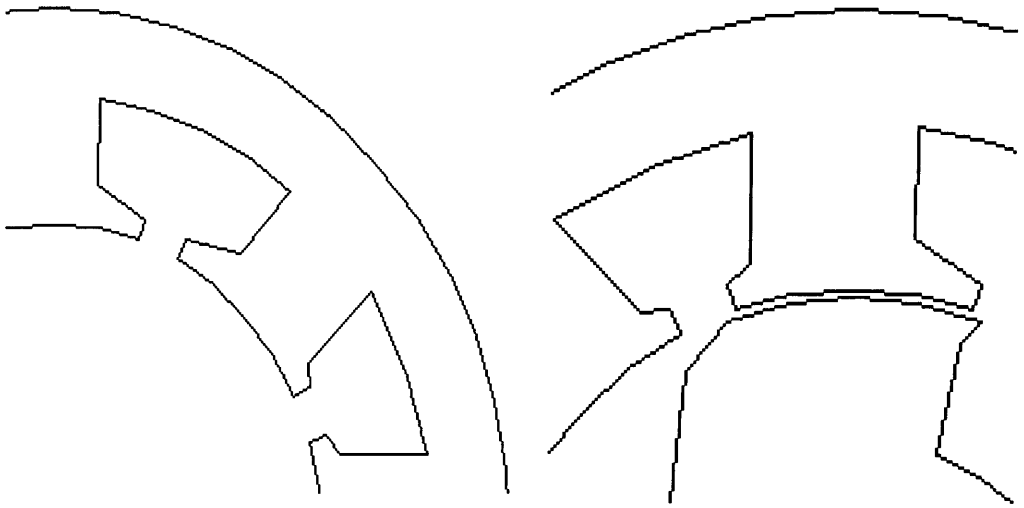


Fig. 6.4 (b) Design with highest fitness value (found in generation 15) - illustration of zoom in view of the rotor and stator lamination design

Design parameters	**Seed**	**Optimised** (*G15i1*)
Width at bottom of armature slot (P1)	0.75	0.7
Width at middle of armature slot (P3)	0.7	0.65
Width at armature slot entry (P2)	0.34	0.2
Armature slot gap (P4)	0.34	0.2
Field slot gap (P5)	0.34	0.2
Width at field slot entry (P7)	0.34	0.2
Rotor arc, as a fraction of slot pitch of 90° (V1)	0.4	0.44
Rotor outer diameter (V2)	53	51

Table 6.2 List of specific change in the design variables of the seed design and optimised design

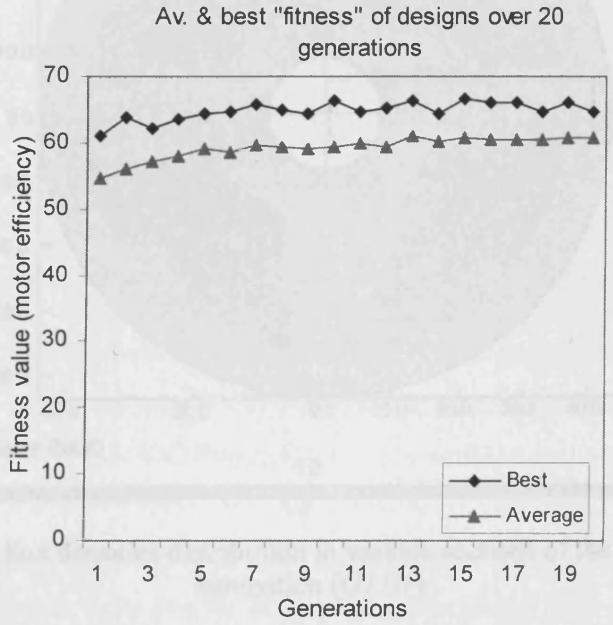


Fig. 6.5 The trend of design fitness over the 20 design generations

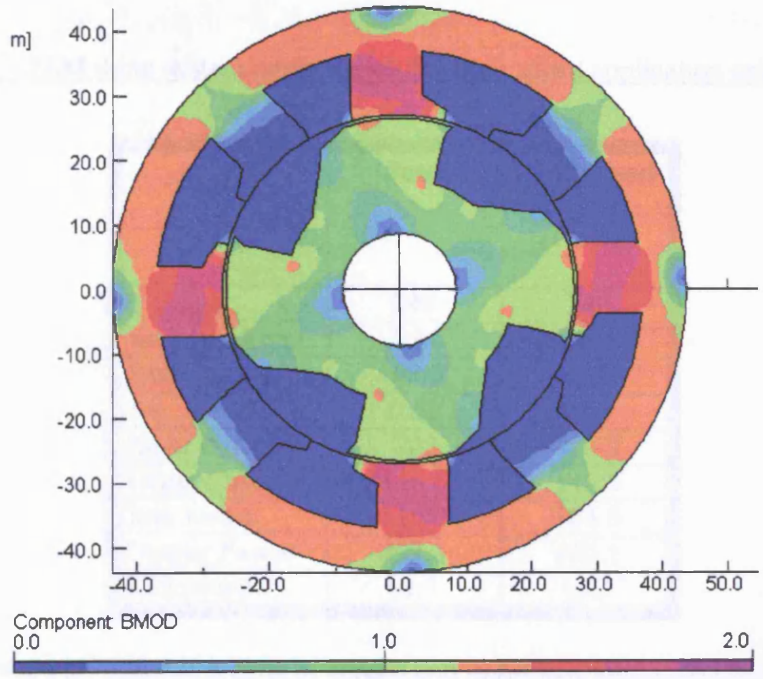


Fig. 6.6 The flux densities distribution in various sections of the seeded design lamination

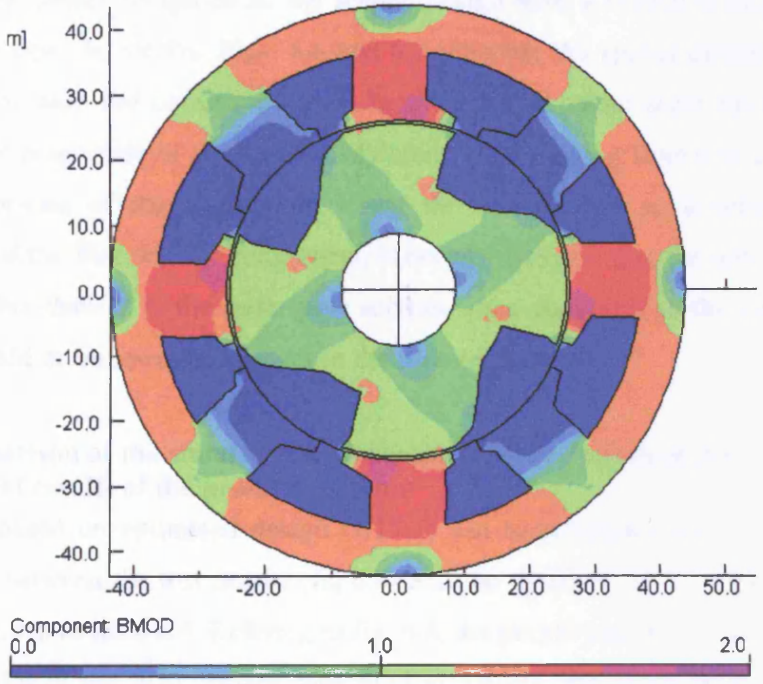


Fig. 6.7 The flux densities distribution in various sections of the fittest design lamination (*G15i1*)

	Seed	Optimised (*G15i1*)
Voltage	240	240
Supply current	2.94	2.93
Rotor speed	8000	8000
Av. Torque	0.52	0.55
Input Power	705	704.2
Copper loss	156.5	140.6
Iron losses	116.2	103.5
Output Power	432.3	460.1
Efficiency	61.3	65.3

Table 6.3 Predicted results of seeded and optimised lamination designs

The predicted dynamic performance of both seeded and optimised lamination designs are illustrated in table 6.3, the data shows that the optimised design can produce a higher output power compared to the seeded design with a reduction in power losses (copper and iron) by 10.5%. Figs. 6.6 and 6.7 illustrate the spatial distribution of flux density in the seed and optimised design laminations. A colour scale has been used to represent the magnitude of the flux density component ranging from 0 to 2 Tesla in the different sections of the lamination. Using the colour scale as a reference to the magnitude of the flux density component, it became obvious that the optimised design has lower flux density in the stator pole section when compared to the seeded design, and this would mean lower iron losses in the pole sections.

6.3.2 Comparison of the simulated performances of the optimised design with experimental results of the prototype motor

The motor based on optimised design (*G15i1*) has been constructed and tested. The comparison between the test results (fig.6.8) and the dynamic model simulated results (fig.6.9) is given in table 6.4. Refering to fig. 6.8, the purple trace (ch3) on experimental waveform plot is the field current measured at 2A/div, the green trace (ch4) is the armature current measured at 2A/10mV div, and brown trace (ch1) is the opto signal the starting edge represents the 0 degree of the rotor orientation. The measured armature current waveform from the prototype motor appeared to tie up quite well with the one estimated by the dynamic simulation model (fig. 6.10). However, there still exists difficulty in simulating the experimental field current waveform, as the modulation of field current due to the mutual coupling of the armature excitation is not included in the

model. The dynamic simulation model represented here is used in conjunction with a genetic algorithm. Thus, the model complexity must be a counterbalance with the program execution time, as a result the electromagnetic interactions of the flux switching motor drive system are substantially simplified and idealised.

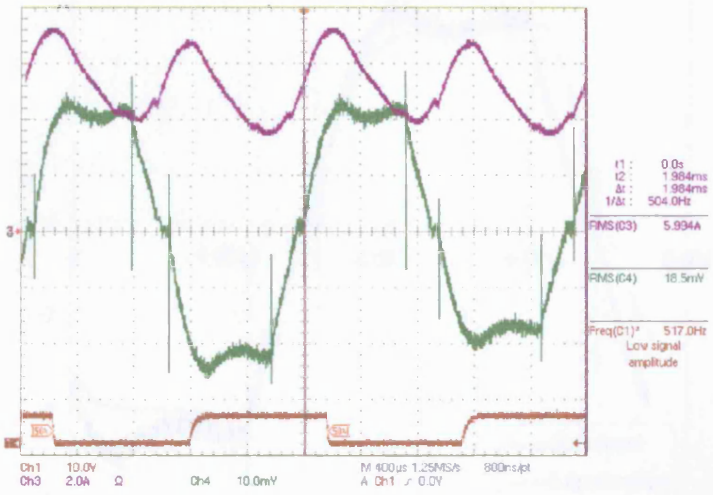


Fig. 6.8 Experimental waveforms for the constructed 8/4 FS motor running at 7840rpm with 0.5Nm externally applied load.

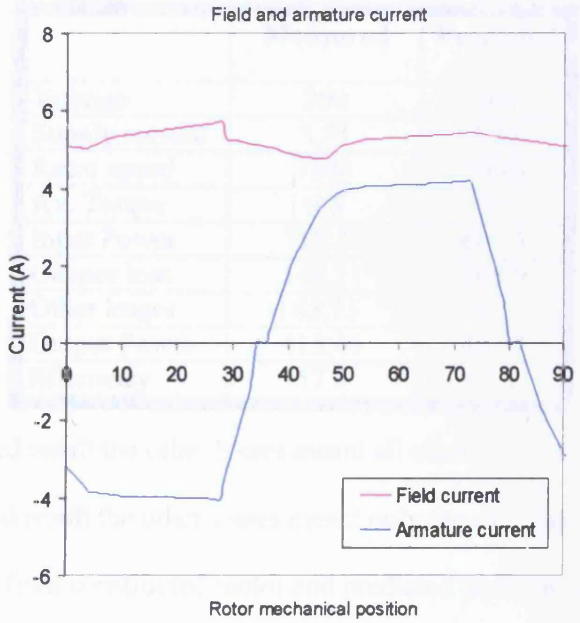


Fig. 6.9 Simulated current waveforms for the constructed 8/4 FS motor running at 7840rpm, 0.5Nm.

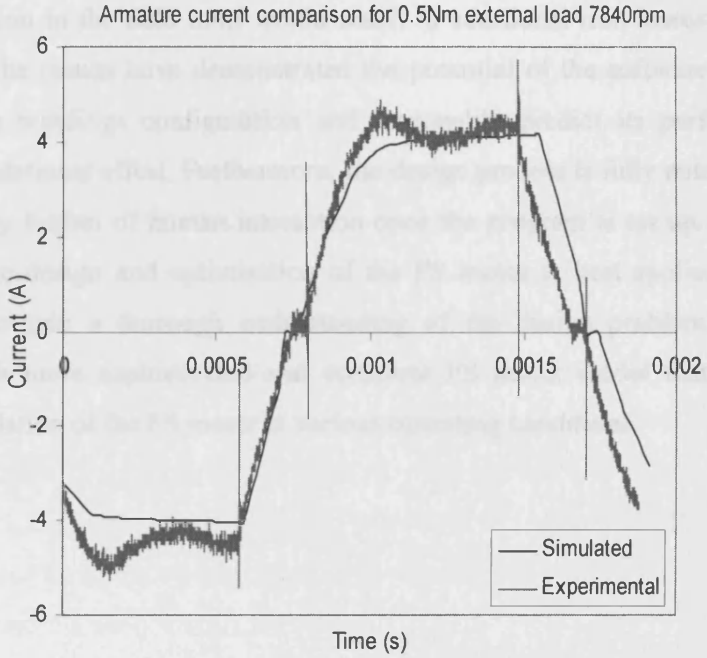


Fig. 6.10 Comparison of simulated and experimental armature current waveforms for the constructed 8/4 FS motor

	Measured	Predicted
Voltage	200	200
Supply current	3.51	3.03
Rotor speed	7840	7840
Av. Torque	0.5	0.5
Input Power	719.32	607.3
Copper loss	142.11	108.9
Other losses	163.75*	91#
Output Power	413.46	407.4
Efficiency	57.5	67.1

* In measured result the other losses meant all other losses excluding copper loss.
# In predicted result the other losses meant only the calculated iron loss.

Table 6.4 Results from constructed motor and predicted performance from software

Referring to table 6.4, the differences between predicted and actual motor efficiency can be explained as such, the predicted results of the optimised motor do not include any mechanical losses, while the iron loss is simulated with the instantaneous current waveforms shown in fig. 6.9 in which the modulating field current is not included. This

extra modulation in the field mmf would result in additional iron losses in the motor. Nonetheless, the results have demonstrated the potential of the software to design the motor and its windings configuration and reasonably predict its performance with modest computational effort. Furthermore, the design process is fully automated and do not require any further of human interaction once the program is set up. The proposed method for the design and optimisation of the FS motor is best applied as an initial design tool to gain a thorough understanding of the design problem and be used alongside with more sophisticated and complete FS motor model that supports full dynamic simulation of the FS motor at various operating conditions.

7 Design optimisation of FS motor for improved starting capability using a genetic algorithm

7.1 Introduction

This chapter describes a second application of the GA design optimisation, in which a FS motor is designed using a different fitness function. The design objective in this particular optimisation problem is to improve the starting capability of an existing Flux switching motor (fig. 7.1), while complying with some of its existing motor dimensions. The task involves redesigning the lamination over the existing motor to increase the torque available in the overlap region between positive and negative armature currents (fig. 7.2) using GA design optimisation. The method of starting a flux switching motor involves pulling the rotor into a stable equilibrium position of zero torque for one polarity of armature current. At that position the other armature current polarity must be capable of producing enough torque to start the rotation. Thus, a lamination design that produces high torque in the overlap region between the two polarities of currents would guarantee the motor a good starting capability.

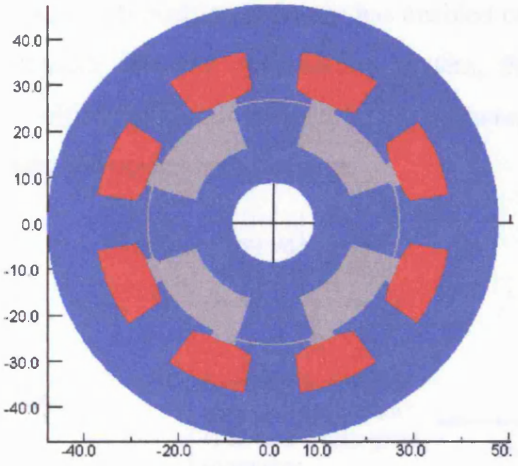


Fig. 7.1 The lamination design of the existing motor

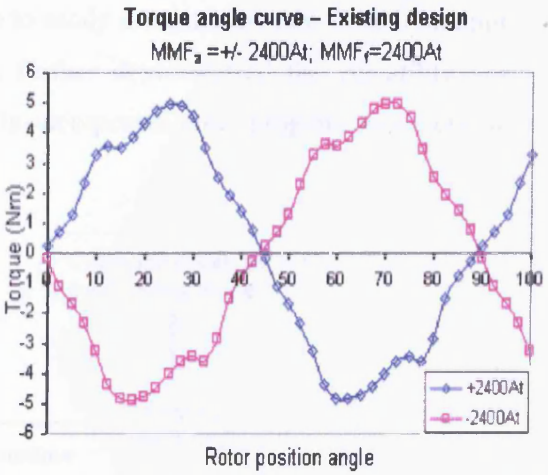


Fig. 7.2 The simulated torque angle curve of the existing design for a given winding mmfs

7.2 Development of a different fitness function program

For the design aims described above the genetic algorithm optimisation program would require a different fitness function as compared to the one used in the earlier chapter to appraise the "quality" of the new designs. For this design problem, the fitness function for each lamination design needs to obtain the instantaneous torque waveform (at the overlap regions between positive and negative armature currents), and use the information as an indication of each design's fitness. The instantaneous torque of the lamination designs at given field and armature current excitations were calculated using the Maxwell Stress method within the FEA software [20], and the calculated data is evaluated directly by the fitness function. A different fitness function program for evaluating the instantaneous torque has been developed and integrated with the genetic algorithms and the FEA software to facilitate the design optimisation. The functional outline of the design optimisation program for improving a FS motor starting torque is illustrated in fig. 7.3. Noticeably, the use of modular program structure in the GA driven design optimisation software has enabled one to easily integrate the new fitness function program into the optimisation system, this further demonstrated the versatility and flexibility of the developed (GAs) software to incorporate other program module(s) to form the optimisation system.

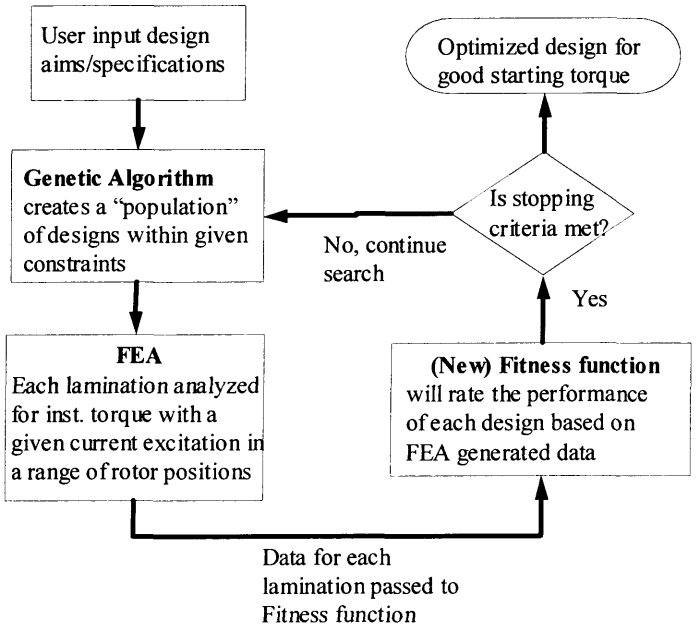


Fig. 7.3 The program outline of the design optimisation program for improving a FS motor starting torque

7.3 Defining the optimisation problem

The same initial design procedures (as described in chapter 6) were used in defining the design aims /specifications and in identifying the possible solutions for this particular optimisation problem.

7.3.1 Design aims and specifications

The design aims and specifications in this optimisation problem are:

1) To maximise the torque available in the overlap regions between positive and negative armature currents for a given field and armature mmfs where in the parameterised FE model total field mmf is set at 2400At and total armature mmf is set at either 2400At or -2400At. These values were used as they reflect the actual operating current level and the number of winding turns of the existing motor.

2) To achieve the above while complying with the following motor dimensions- number of stator poles: 8, number of rotor poles: 4, stator outer diameter: 95mm, airgap length: 0.6mm, rotor outer diameter 52.6mm, shaft diameter: 17.45mm, and motor stack length: 40mm.

7.3.2 Identification of the motor parameters to be varied

For the design aims described above the following possible design variations have been identified and they are shown in Table 7.1 the parameters variation are illustrated in figs. 7.4 and 7.5. There are 12 design variables that would be altered, six representing the rotor variations and the other six design variables are found on the stator.

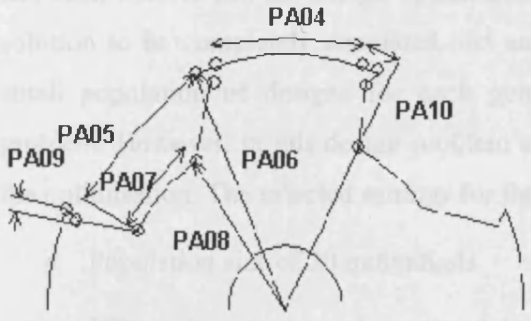


Fig. 7.4 Design variables of the rotor

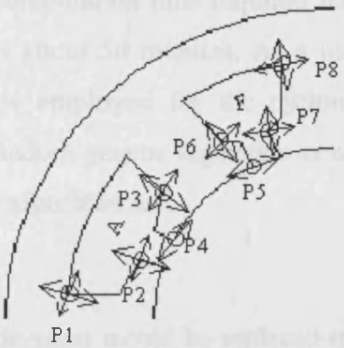


Fig. 7.5 The design variables of the stator

Design Parameter	Mapping range			
	Lower limit	Upper limit	Encoded bits	No. of possible values
Rotor arc, as a fraction of slot pitch of 90° (PA04)	0.3	0.45	4	16
Graded air-gap at leading edge of rotor pole (PA05)	0.3	0.4	3	8
R1 radius at leading edge of rotor pole (PA06)	0.85	0.96	4	16
Pole width at leading edge (PA07)	0.5	0.65	4	16
Pole width at trailing edge of R1 (PA09)	0.18	0.28	3	8
Pole width at trailing edge of R2 (PA10)	0.18	0.35	4	16
Armature slot opening (P4)	0.2	0.62	4	16
Field slot opening (P5)	0.1	0.45	4	16
Armature slot, width of tooth tip (P2)	0.2	0.62	4	16
Armature slot, thickness of tooth tip (P2)	5	15	3	8
Field slot, width of tooth tip (P7)	0.1	0.45	4	16
Field slot, thickness of tooth tip (P7)	5	15	3	8

Table 7.1 Design variables with their given dimensional limits

7.3.3 Genetic operators

In this optimisation problem, the same computer (with Pentium III 700Mhz processor) has been used to run the design optimisation. The computation time required for each solution to be completely simulated and analysed is about 50 minutes. As a result, a small population of designs for each generation is employed for the optimisation problem. However, in this design problem a more random genetic algorithm is used in the optimisation. The selected settings for the genetic algorithm are:

- Population size of 20 individuals.
- 30% replacement rate (meaning 6 least fit individual would be replaced in each generation).
- Random selection

- Uniform single point crossover
- Random mutation with the mutation frequency set at 0.002 (two mutations per thousand bits).
- Termination after 20 generation.

7.3.4 Defining the fitness function

The fitness function set for the initial design and optimisation process would be aiming at increasing the torque available in the overlap region between positive and negative armature currents. The fitness function program would interpret the static torque-angle curves generated by FE model and gives an indication of the fitness of each lamination design. The algebraic expression of the fitness function is given as:

$$F(x) = f_1(x) + f_2(x) - \sum_{i=1}^{n} h_i(x)$$

where $F(x)$ is the fitness value assigned to a lamination design, $f_1(x)$ is the motor torque at the 1st intersection between positive armature mmf and negative armature mmf torque curves, $f_2(x)$ is the motor torque at the 2nd intersection between positive armature mmf and negative armature mmf torque curves. $h_i(x)$ is a penalty function used for penalizing designs that have a large difference between f_1 and f_2. The fitness function is defined by the sum of two torque curves intersections between the positive armature mmf torque curve and the negative armature mmf torque curve. Further penalties were included in the fitness function when design violates certain criteria. Penalties or negative values are added to the assigned fitness value for each design, if the following are violated,

1) when a lamination design has a torque curves intersection of zero or less;
2) when the difference between the first and second torque curves intersections are more than a user-defined value (eg. 0.3Nm);
3) heavier penalty if both conditions are violated.

7.4 Results from the GA design optimisation for improving motor starting torque

The procedures of starting the optimisation program are similar to those described earlier in chapter 6. From specifying the population size, the genetic operators, the design variables with their dimensional limits (for details of the various specification refer to earlier sections). The initial population consists of 20 randomly generated lamination designs by the genetic algorithm, where the geometry of each design is defined by a set of design constants (non-variants pre-specified in the parameterised FEA model) and a set of design variables, and it is by changing the design variables which allows the different lamination shapes to be generated. Each set of design variables is passed to the Design Environment to create the finite element model and the design is analysed for its torque producing capability for a given winding mmfs. The instantaneous torque solutions of each design are then used in calculating the fitness value of each design, and based on the fitness information of each design, the genetic algorithm then performs its genetic operation to produced new and improved generation of designs. The GA design optimisation process is terminated after 14 generations 6 generations before target, as the possibility of finding the "global optimal" design cannot be guaranteed.

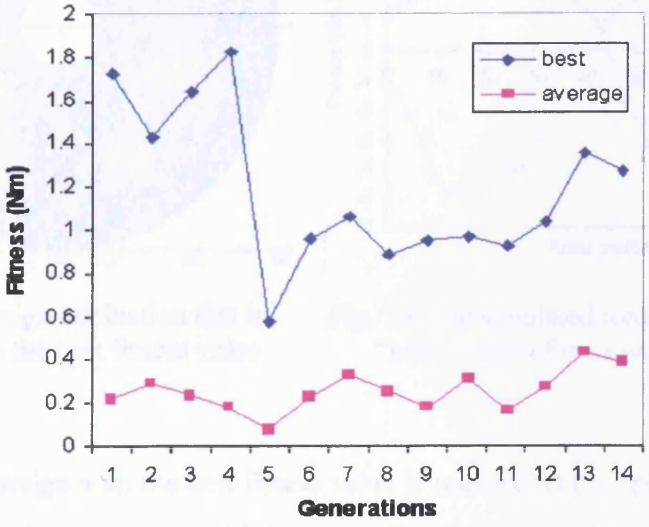


Fig. 7.6 Average fitness and best fitness trend during the optimisation process

Fig. 7.6 illustrates the search results of the design optimisation the blue trend line shows the fitness value of the best lamination design in each generation and the pink line represents the average fitness value of the lamination designs in each generation. It has

been observed that the blue trend line (representing the best fitness in each generation) exhibits a large fluctuation in *generation* 5. The huge dip in the fitness of the best individual in *generation* 5 is due to a combination of two factors; firstly the average fitness in the preceding population of designs, *generation* 4 is low and also in the same population existed a "super" fit individual design (with the highest fitness value amongst all designs). However, this would mean that the fitness for the most of the designs in that generation were lower than average fitness. Secondly, random selection is the chosen method for selecting pair of designs to exchange their genetic information, under the circumstances stated earlier it would mean that the "super" fit design has a very high probability of being paired with a highly unfit design thus resulting in the deterioration of the fitness. As the design optimisation progressed the fitness of the overall population starts to improve as more fit individuals were reproduced while the unfit ones were being replaced.

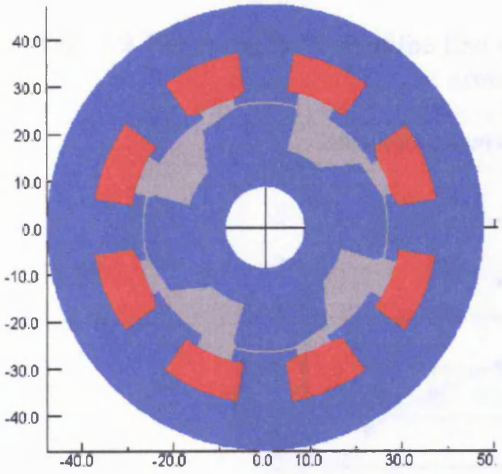


Fig.7.7 The design lamination that is predicted with the best fitness value

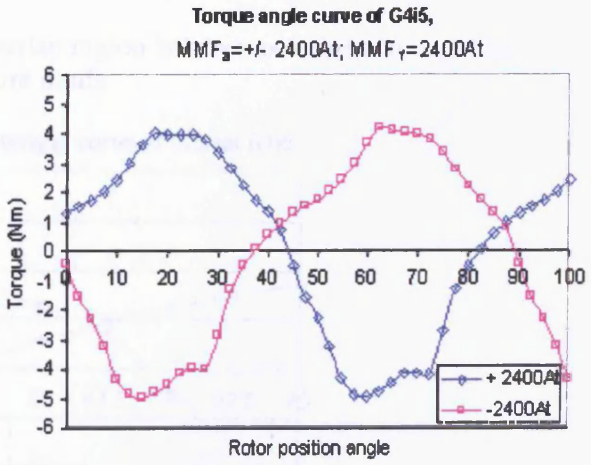


Fig. 7.8 The simulated torque angle curve of the "best" design for a given winding mmfs

The lamination design with the best fitness value is selected as the optimal design (fig. 7.7), and this design was generated in *generation* 4 of the optimisation. The static torque-rotor position waveforms for this particular design are shown in figs. 7.8, 7.9 and 7.10, where fig 7.8 illustrates the full scope of the torque- rotor position waveform and the figs 7.9 & 7.10 show the zoom in of the overlap regions between positive and negative armature mmfs. The fitness value of the design of 1.83 is calculated by

summing the values of both intersections between the two armature mmfs. The penalty function in this case is zero as none of the given constraints are violated.

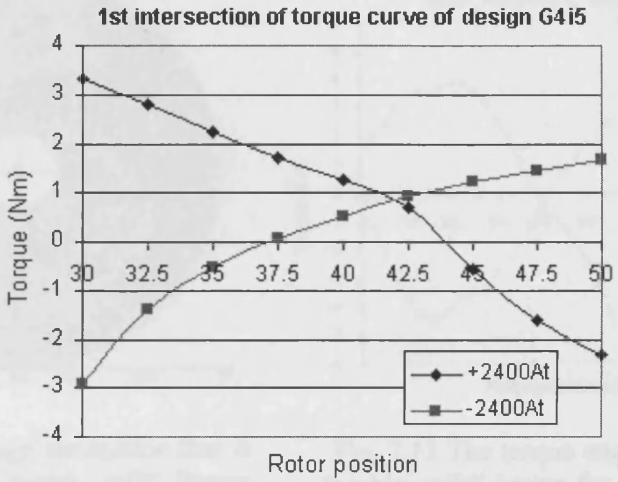


Fig. 7.9 The zoom in view of the first overlap region between positive and negative armature mmfs

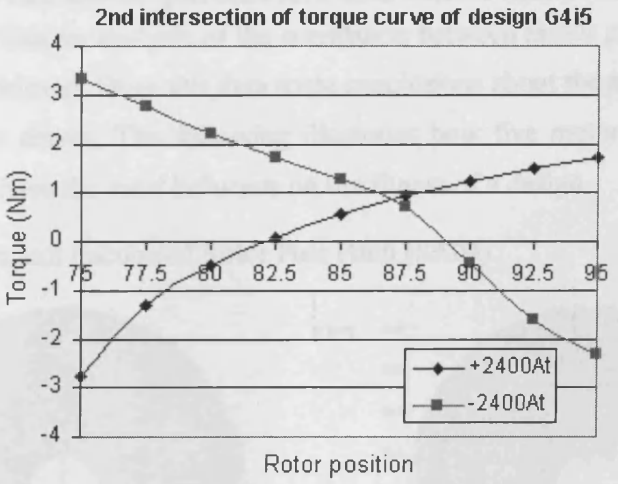


Fig. 7.10 The zoom in view of the second overlap region between positive and negative armature mmfs

Figure 7.11 illustrates an interesting lamination design in the population that has been evaluated as highly unfit by the fitness function. The design has a negative and positive intersection of the torque curves (fig. 7.12), the design is considered by the fitness function to be highly unfit because it has violated all the three constraints of the penalty function (refer to section 7.3.4). In general, the results from the design optimisation for improved starting torque had demonstrated a wide range of fitness for the different lamination designs generated. The outcome of the results is due to the fact that a more

random and less deterministic genetic algorithm has been employed, the wider scope of designs with varying fitness values generated by the genetic algorithm would allow one to better understand or perhaps be mapped out as functions on how the design variations can affect the fitness of a lamination design.

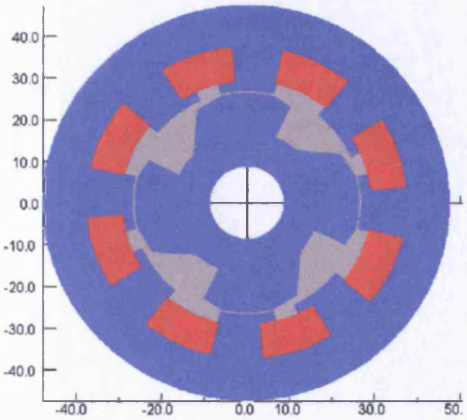


Fig.7.11 The design lamination that is predicted with a "highly unfit" fitness value

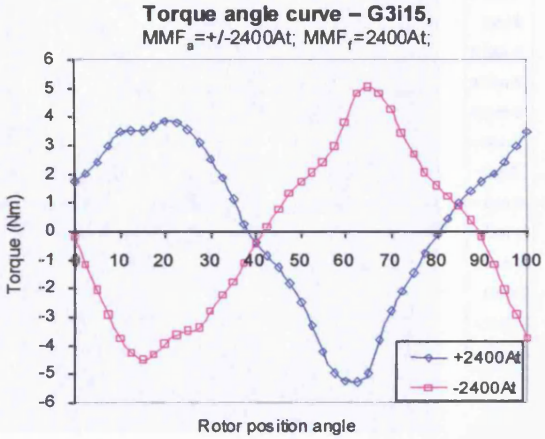
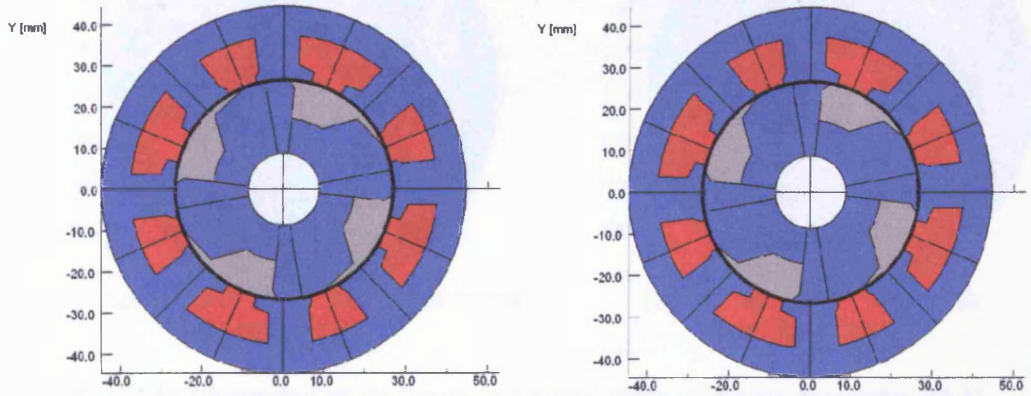


Fig. 7.12 The torque angle curve of the "highly unfit" design for a given winding mmfs

7.5 Analysis of the motor parameters and fitness function

This section describes an analysis of the correlation between motor parameters and the fitness function achieved. From this data some conclusions about the most critical motor parameters can be drawn. The following illustrates how five motor parameters have been identified to have the most influence on the fitness of a design.

- Rotor Pole Arc as a fraction of Rotor Pole Pitch (PA04)



low_limit =0.3 upp_limit =0.45

Fig. 7.13 Illustration of the upper and lower limits of design parameter PA04

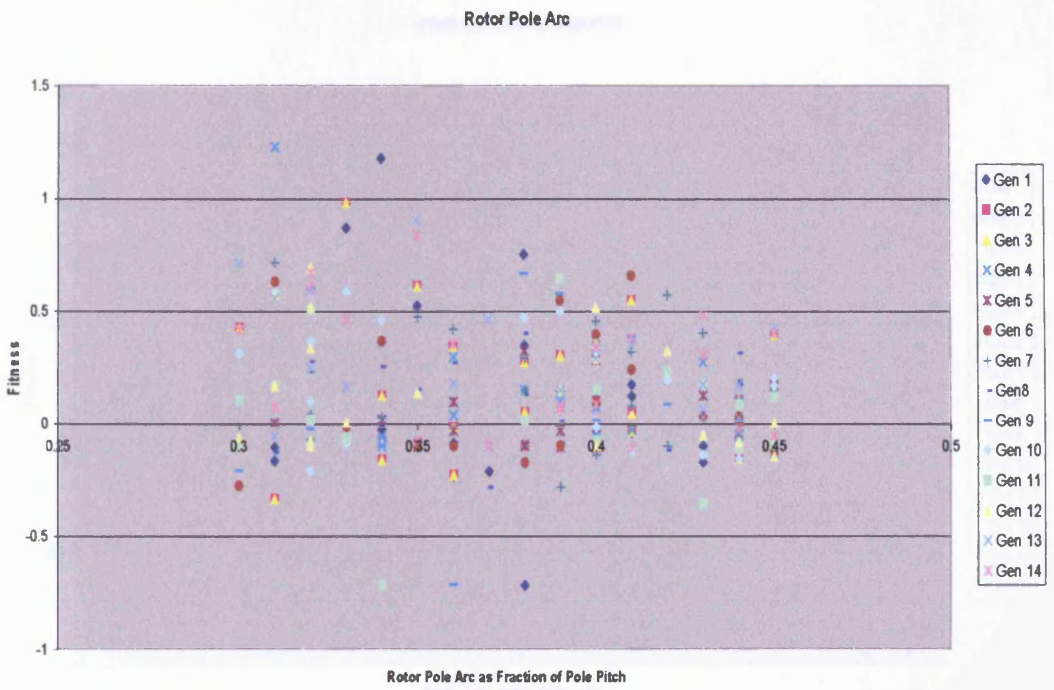


Fig. 7.14 The mapping of the critical motor parameter, PA04 against fitness

Observation 1: It is identified that smaller rotor pole arc that is proportional stator pole arc would led to higher fitness.

- Angle from Centre of Rotor Pole to tip of graded rotor on leading edge as a fraction of rotor pole pitch (PA05)

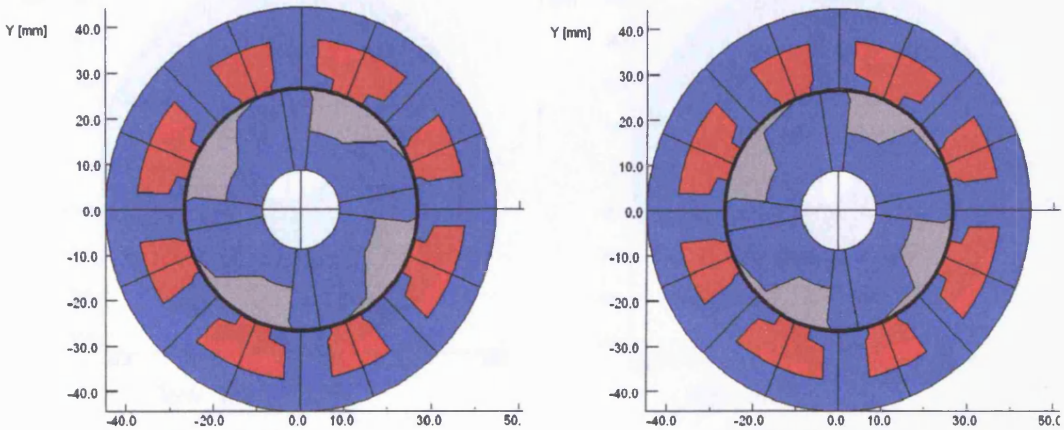


Fig. 7.15 Illustration of the upper and lower limits of design parameter PA05

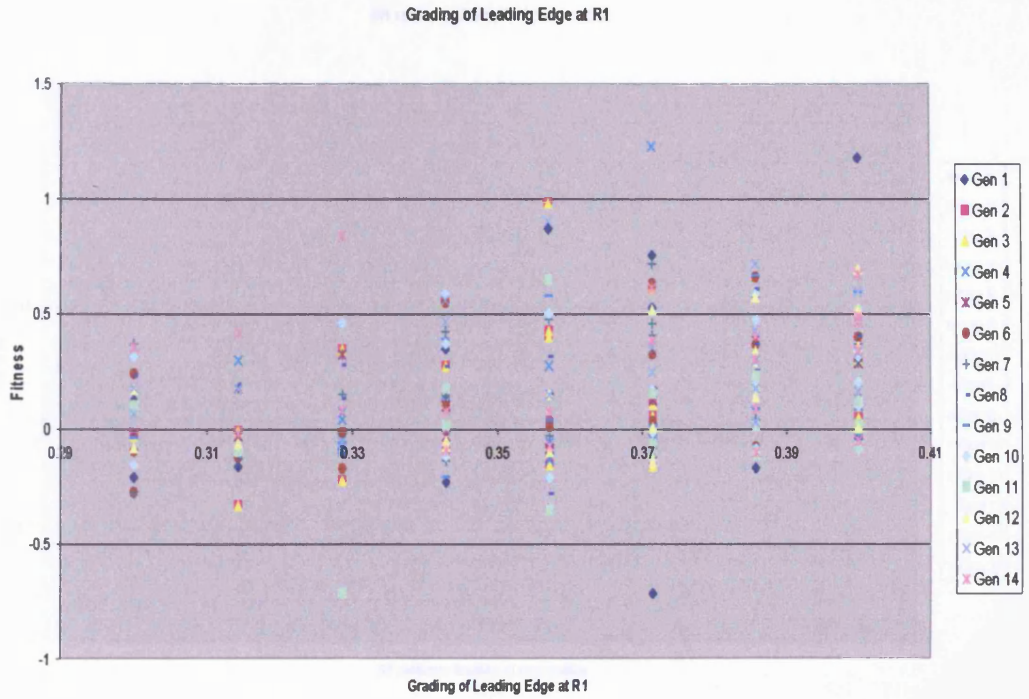


Fig. 7.16 The mapping of the critical motor parameter, PA05 against fitness

Observation 2: Designs with wider graded pole on leading edge generally led to higher fitness.

- Radius of leading edge of graded rotor as a fraction of rotor radius (R1), PA06

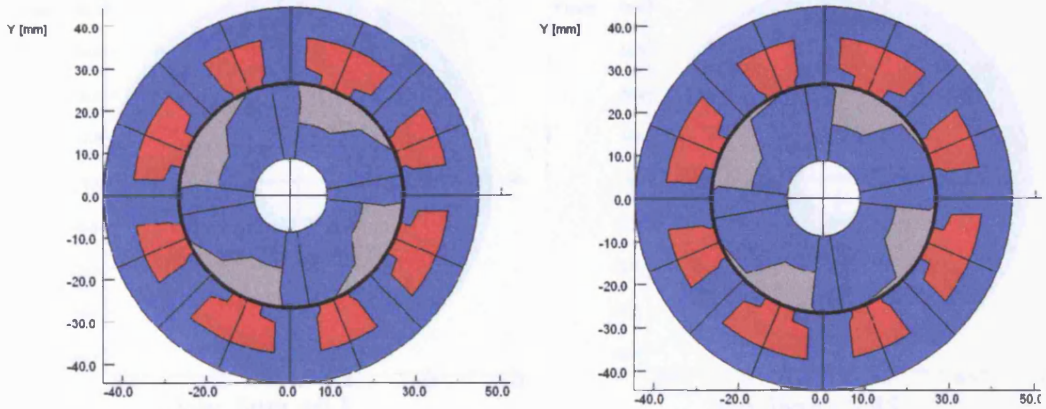


Fig. 7.17 Illustration of the upper and lower limits of design parameter PA06

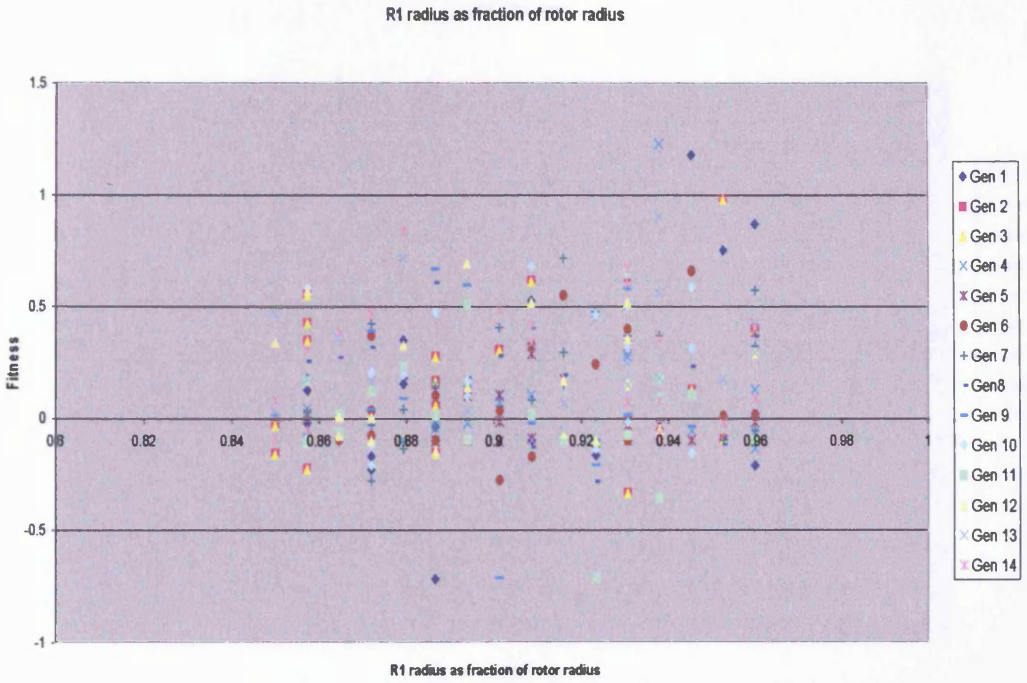


Fig. 7.18 The mapping of the critical motor parameter, PA06 against fitness

Observation 3: Designs with radial position of tip of graded leading edge was optimal at 0.94 of rotor radius.

- Slot1 (Field) opening as a fraction of stator pole pitch

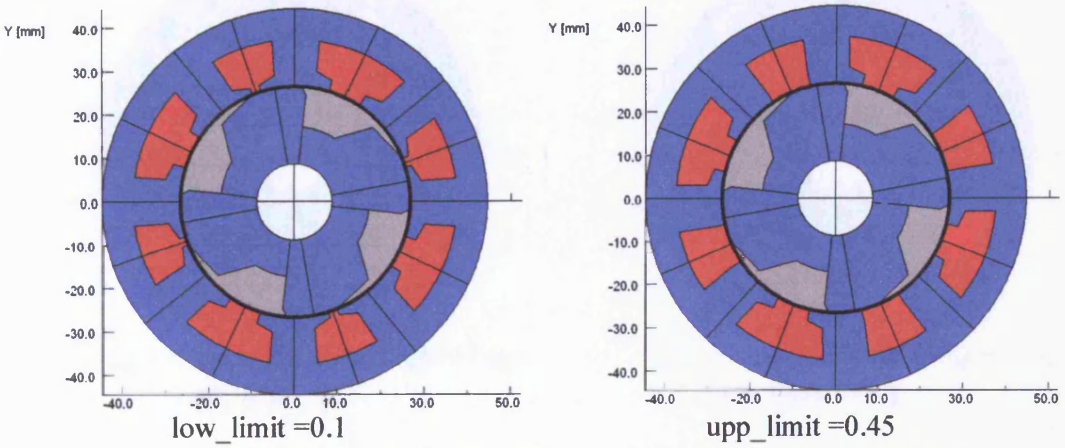


Fig. 7.19 Illustration of the upper and lower limits of design parameter P5

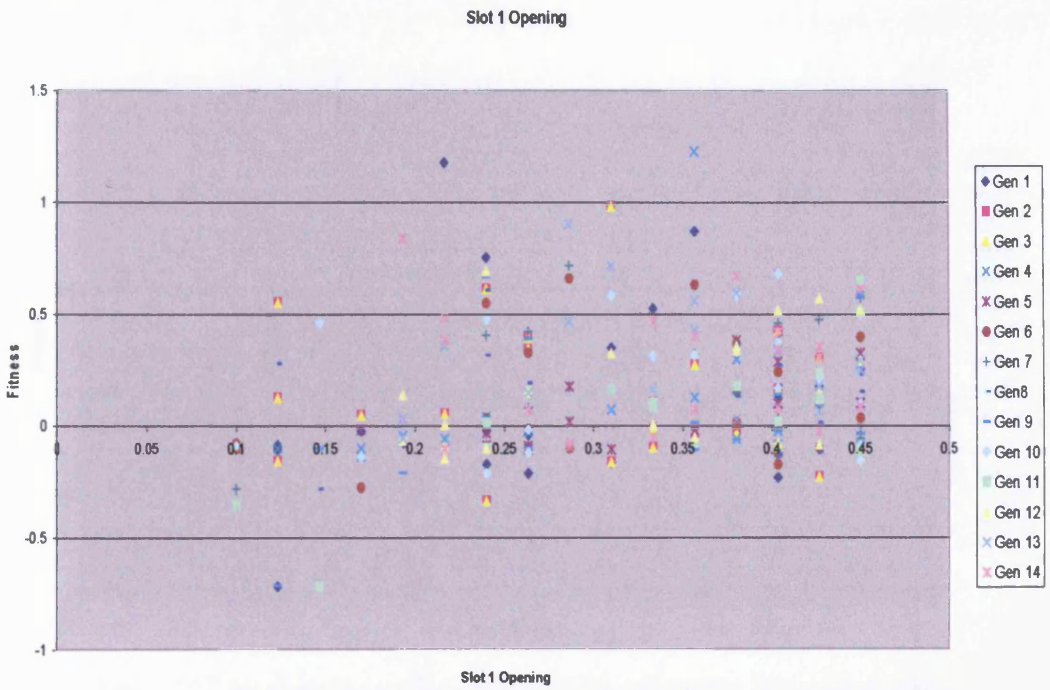


Fig. 7.20 The mapping of the critical motor parameter, P5 against fitness

Observation 4: Slot 1 (field) opening produced better designs in the range 0.2 to 0.36.

- Slot2 (Armature) opening as a fraction of stator pole pitch, P4

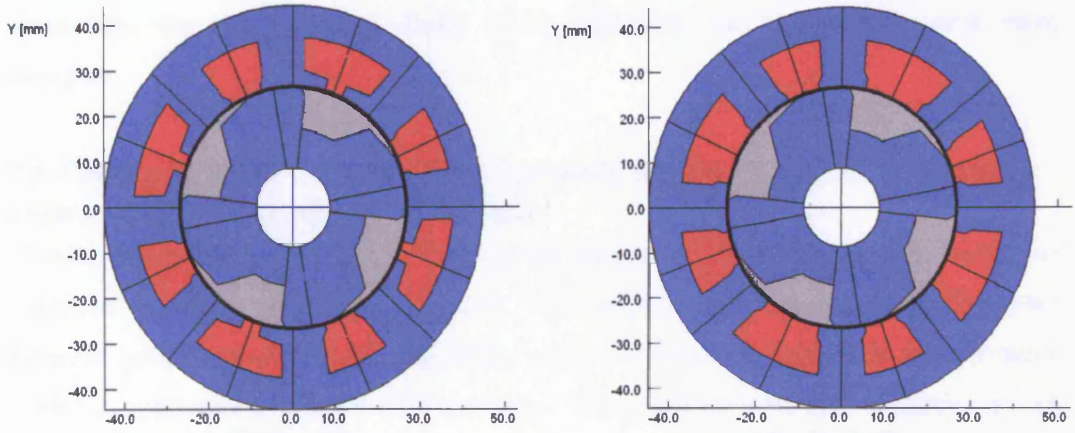


Fig. 7.21 Illustration of the upper and lower limits of design parameter P4

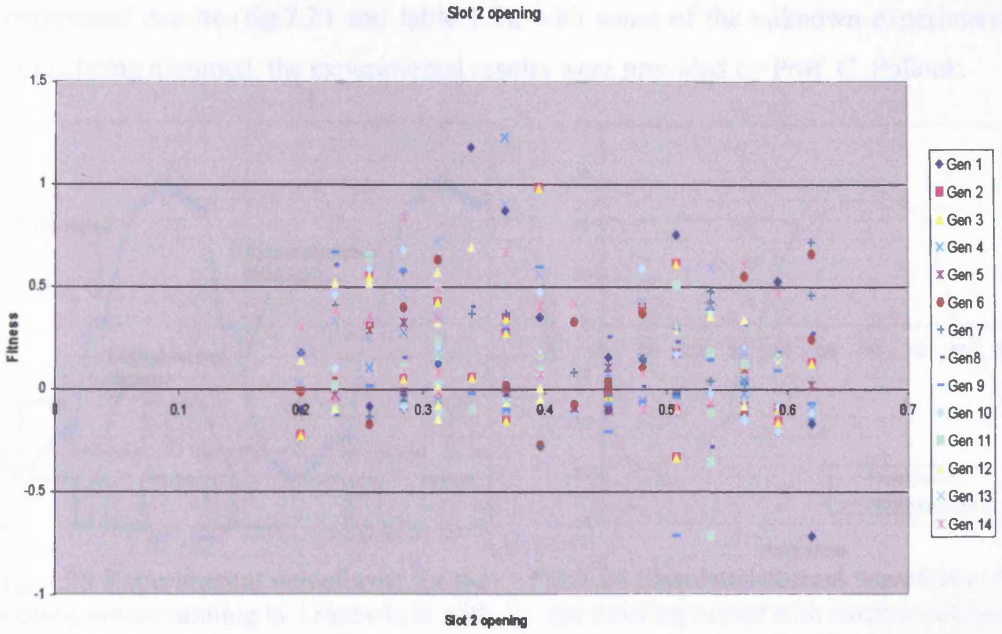


Fig. 7.22 The mapping of the critical motor parameter, P4 against fitness

Observation 5: Slot 2 (armature) opening produced better designs in the range 0.3 to 0.4.

This simple exercise of mapping out individual motor parameter (within a given range) against the design fitness function has allowed the identification of the critical design parameters that has strong influence in the fitness of the design to become more intuitive.

7.6 Comparison of the simulated dynamic performance of the existing motor with the GA optimised motor

In this section, the predicted dynamic performance of the optimised FS motor for improved starting performance is given. The prediction of the optimised FS motor dynamic performance is made separately using the developed dynamic drive system model in conjunction with the FEA software. The dynamic drive system model is first used to predict the dynamic performance of the existing design and followed by fine-tuning the model. The predicted results of the existing motor are illustrated in figs. 7.23-7.25 and in table 7.2. In the experimental plot (fig. 7.23) the blue trace is the armature current (ch2) measured at 10A/10mvdiv and the yellow trace is the opto signal where the starting edge represents the rotor to stator pole alignment position. The dynamic simulation model was fine-tuned to get result as close as possible to match the actual

experimental results (fig.7.23 and table 7.2), with some of the unknown experimental settings being assumed, the experimental results were provided by Prof. C. Pollock.

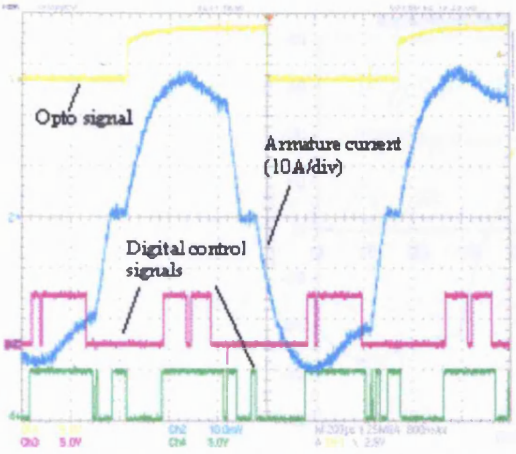


Fig. 7.23 Experimental waveforms for the existing motor running at 13krev/min with a external load of 2.35Nm

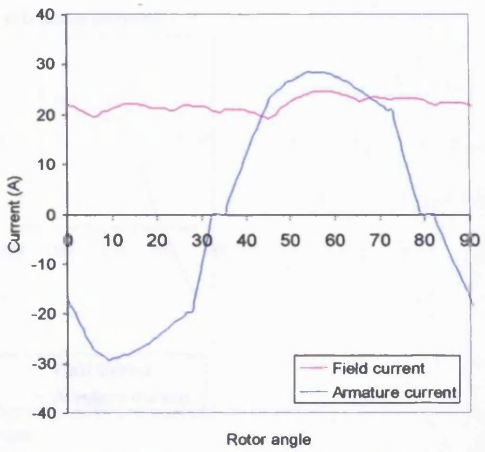


Fig.7.24 Simulated current waveforms for the existing motor with similar voltage and speed settings.

	Measured	Predicted
Voltage	240.49	240
Rotor speed	13333	13333
Av. Torque	2.35	2.21
Input power	4630.7	4155.7
Copper loss	-	886.9
Iron loss	-	187.5
Output power	3281	3081.3
Efficiency	70.85	74.1

Table 7.2 Comparison of measured and predicted results of the existing motor.

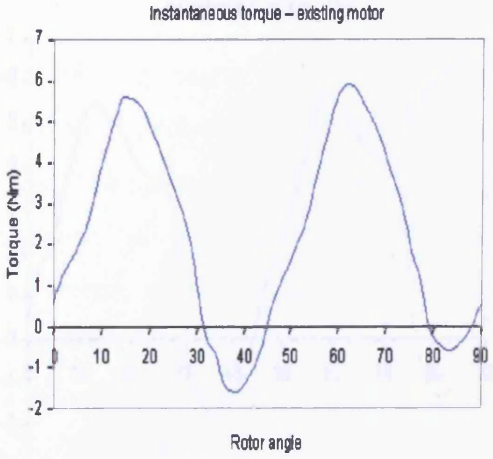


Fig. 7.25 Simulated instantaneous torque of existing motor using current waveforms shown in fig. 7.14, with an average torque 2.22Nm

The predicted dynamic performances of both the optimised and existing designs are compared (in table 7.3), with these results and the static torque data the two designs can be more realistically compared. The optimised design is predicted to have almost similar torque output compared to the existing motor but have the advantage of better starting capability. A new design based on the results of this simulation is now being

constructed. The improved starting torque has been achieved without any noticeable deterioration in the running performance.

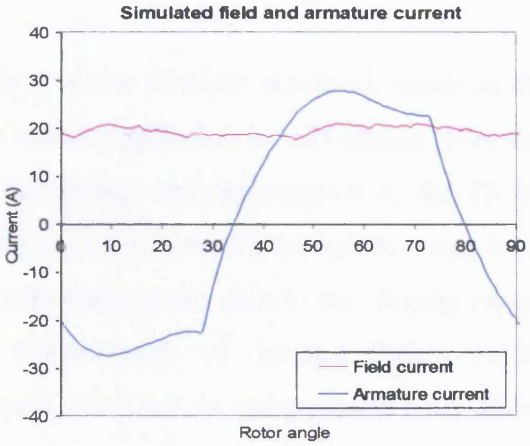


Fig.7.26 Simulated current waveforms for the optimised design operating at 240V and 13krev/min

	Optimised	Existing
Voltage	240	240
Rotor speed	13333	13333
Av. Torque	2.17	2.21
Input power	3987.2	4155.7
Copper loss	752.9	886.9
Iron loss	202.6	187.5
Output Power	3031.8	3081.3
Efficiency	76	74.1

Table 7.3 Comparison of predicted results of the optimized design and the existing design.

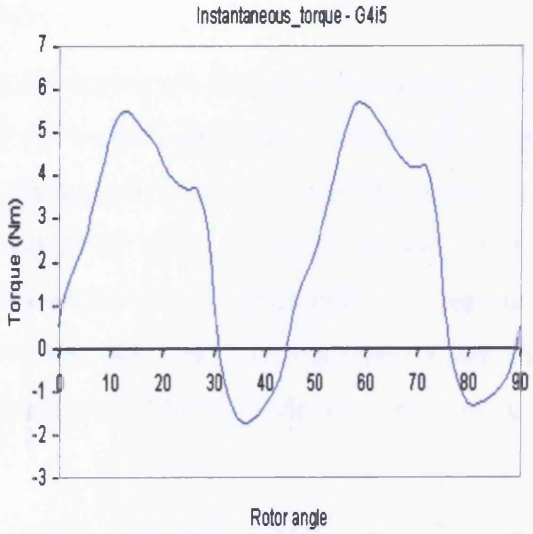


Fig. 7.27 Simulated instantaneous torque of optimised design using current waveforms shown in fig. 7.16, with an average torque 2.22Nm

8 Conclusions and further work

8.1 Conclusions

The flux switching motor is a new class of reluctance machine in which experiments of the early prototype have offered performance advantages over brushed motors in many applications. However, the design and optimisation of the FS motor is a complicated process, achieving the design objectives often requires many mechanical and electrical parameters to be tuned simultaneously due to the strong interdependencies between most design variables. Furthermore, the limited design knowledge for this newly developed technology made the task of optimisation even more difficult. From past experiences [2-7], it is understood that the design optimisation of several electrical machine types with multi-dimensional problem have be solved effectively with genetic algorithms. Thus, to facilitate the design optimisation of the FS motor a genetic algorithm based optimisation tool is developed.

The construction, operation and design of the FS motor are described in chapter 2, and design parameters that would possibly affect the overall performance of the motor are highlighted. The data of the highlighted design parameters of the FS motor are consolidated and developed into a parameterised finite element model, where geometrical dimensions of the FS motor are being defined and controlled with algebraic equations. The introduction of such methodology provides the opportunity for the optimisation software to vary the design parameters of the FS motor to create new and improved lamination design.

Chapter 3 describes the development of a dynamic simulation model of a FS motor dynamic drive system that is coupled with a FEA program. The system model uses minimal static FEA calculated results of winding flux linkages and further extrapolate the rest of the required winding flux linkages at other field mmf levels using simple algebraic equations. The developed method consists a linear model and a non-linear model in which the two models defined the flux-mmf characteristic of a given FS motor geometry. The complete winding flux linkages variation results are used in the system model to predict the motor armature back-emf, the variation in the armature back-emf relative to rotor position is then used to calculate the dynamic performance of the FS motor with simple time-stepping electric equivalent circuit equations and electromechanical torque equations. Also within the system model is a switch control

module that is linked with the time-stepping circuit equations and a new winding optimisation module. The winding optimisation module first chooses the best possible winding configuration for a given slot area by adjusting the winding turns and wire size to meet user specified requirements on the motor's intended speed and load. While the switch control module is employed after to find the most efficient firing angle to deliver the required motor performance. The developed system model in conjunction with a separately developed iron loss model provides rapid estimation of FS motor performance, thus facilitating the simulation and comparison of a wide range of the motor lamination shapes. The developed model can be used either as a standalone dynamic simulation model or be integrated into the optimisation software as a design tool.

Chapter 4 examines methods of quantifying both the electrical and mechanical losses that occur when operating a FS motor, with the focus mainly on the prediction of the iron loss. The natural phenomenon of the iron loss and other related issues were studied and different methods of quantifying the iron loss were analysed. A simple comparison between the two different iron loss calculation methods is made. Both methods (Steinmetz equation and loss separation method) use a direct approach to quantify iron loss based on experimental results (loss data curves) as a function of magnetization frequency and peak induction level, however it was the latter method that was found to be more reliable as it is found to be more accurate on a wider frequency range. The developed iron loss model (based on loss separation method) has been used to estimate iron losses in a prototype 8/4 FS motor under different operating conditions while accuracy of the estimations and experimental results have been compared. Accuracy of the model varied from the different test ranges, the possibilities of discrepancies in the loss estimations are discussed. The developed method for predicting iron loss is thought to be the best available solution at the time, as it enables a fast and direct approach of estimating the iron losses, which is necessary in a design optimisation cycle.

The design and development of the optimisation software was described in chapter 5. The design of the software uses an object-oriented analysis method, from identifying every required functions (objects) and defining their associations with other (objects) functions. The identified objects were assembled into smaller and more manageable projects (modular programs) based on their relationships with each other through modularisation. All objects functionalities in each program module were further defined

into detailed statements in terms of what are the tasks and responsibilities of the each object, and a program structure further mapped out as a flow chart for each module. Using the defined program structure, each program module was developed as a separate project, and upon completion of all projects the separately developed program modules were then integrated into the optimisation software.

Chapter 6 described the design optimisation of an 8/4 flux switching motor to improve motor efficiency with specified design constraints using the developed optimisation software. The procedures of defining the optimisation problem (the problem space, the objectives) were detailed and suggestions for the selection of the appropriate genetic operators to achieve a successful evolution were given. The optimisation software was firstly setup with user input of required design aims and specifications, then the program was executed to design and optimised the flux switching motor without any further requirements of human interaction, the entire design optimisation process was directed by a genetic algorithm. After several iteration of the design cycle, a new improved lamination design and optimal winding configuration for a FS motor emerged. The new design of flux switching motor created by the optimisation software demonstrated a better dynamic performance compared to an initial (seed) design. The new lamination design with its winding configuration had been constructed and comparisons between the software predicted and actual measured motor performance were made. The comparison results confirmed the limitations of the dynamic drive simulation model to accurately predict the actual operating performance, as the modulation of the field current due to strong mutual coupling of the armature excitation is not included in the model. Nonetheless, the results have demonstrated the potential of the software to design the FS motor and its winding configuration, while quite accurately predicting its performance. A further significant aspect of the software is that of the minimal need for interactive design input.

Chapter 7 described design optimisation of a flux switching motor of another application for improving the starting capability. A different fitness function program module has been developed and integrated with both the genetic algorithm optimisation tool and the FEA software. The interface of the new fitness function with the main optimisation software has been simplified as a result of the "plug and play" nature of the modular program structure. The new fitness function used static FEA calculated torque – angle curves of both the positive and negative armature mmfs for a given field mmf to

derive the fitness of each lamination design, the overlap regions between positive and negative mmfs torque curves indicated the fitness of the design where design with two high torque intersections between the curves was rewarded with a high fitness. In this optimisation problem a genetic algorithm with a more random nature has been used to improve an existing design. The solutions generated during the design cycle as a result have also demonstrated a wide (random) range of fitness for the different lamination designs created. The lamination design with the best fitness value has shown improved starting capability compared to existing design. A separate comparison of dynamic simulation of the existing design and the optimised design has been made, and the results of the comparison showed that the existing motor is producing slightly more torque than the optimised design but the optimised design has a slightly better efficiency. The newly developed fitness function enables the user to improve the starting capability of a flux switching motor. However, further investigation on the dynamic performance of the optimised design should be conducted to ensure that the dynamic performance of the optimised design does not deteriorate as a result of improving the starting capability of the motor.

The fully integrated genetic algorithm optimisation software has facilitated its user to design and optimise flux switching motor and the windings configuration with minimal input to the design process. In addition, the software could rapidly search through many potential solutions to find the best possible solution for a given set of constraints and specifications within a reasonable timescale. This rapid design process is made possible by a simplified dynamic simulation model, which can rapidly calculate the dynamic performance of a FS motor with reasonable accuracy. Within the dynamic simulation model consisted several losses determination methods which are extendable to application of other types of electrical machine. The developed method for design and optimisation of flux switching motor and drive is suitable for implementation by research groups or bodies interested in developing such technologies but have limited resources in term of sophisticated analytical software, high performance computers and laboratory equipments.

Furthermore, the developed optimisation software for flux switching motor and drive has significantly reduced the requirements of user interaction with the different design tools. At the various stages of the design cycle commonly associated with manual design process using computer aided design tools, the user input is limited to merely pre

processing setup of design aims and specifications. This would allow the engineer to have more time on conceptual machine design, while the genetic algorithm would search through the possible permutation of the lamination designs. The results gathered from the search could be further processed with statistical software or method to identify the related design parameters that governed the motor performance. The process of mapping out the search results would allow the engineer to study and better understand how design could be improved by varying the design parameters. The developed software provides the user with a different and possibly more efficient method to the design optimisation of a flux switching motor and drive. The developed software has been used to design FS motors without any human intervention once the program has been initialised. However the results show that at this stage it is best to apply the optimisation software only as an initial design tool to gain a thorough understanding of the design problem. While a more sophisticated FS motor model that supports full dynamic simulation should be used as a followed up to verify those designs that have been assigned with high fitness values.

The development and integration of the general-purpose genetic algorithm optimisation tool with the 2-D FEA software is one of the highlights of the developed software. The application of the developed GA optimisation software is not only limited to design optimisation of the FS motor and drive. It may be applied to the design and optimisation of other types of electromagnetic device, which may be modelled using 2-dimensional finite element analysis. The extended application of the optimisation software is made possible, as reusability and versatility of the modularised program had allowed other parameterised finite element model of any electromagnetic devices and the related performance evaluation (fitness function) program to be easily integrated with the developed software. The generic characteristic of the GA optimisation program would prove to be very useful when employed in design optimisation of electromagnetic devices.

8.2 Areas for further work

The drive system model developed for rapid estimation of the FS motor dynamic performance does not account for the windings mutual coupling effect and the armature reaction effect. The model offers a rapid solution for determining the fitness of a lamination design to facilitate the purpose of GA design optimisation process, it does not model the field current as accurately as required. When it comes to verifying the

actual dynamic performance of the designs that have been assigned with high fitness values, a more reliable FS motor model that supports full dynamic simulation should be used. A more sophisticated and accurate dynamic simulation model for the FS motor could be developed in either a Pspice circuit analysis or a Matlab simulink program, and the accuracy of the dynamic model would be largely dependent on the quality of the generated flux map [35], a complete flux map with every possible combinations of field and armature winding mmfs representing the electromagnetic characteristic of a given FS motor geometry usually takes several days to be completely generated, as a result is not suitable for the proposed design optimisation process and is recommended to be used only when an accurate dynamic simulation of the FS motor and drive performance is required.

The results obtained for design optimisation of the two flux switching motors presented in this thesis substantiate the usefulness of the genetic algorithms based optimisation method in FS motor design (and motor design in general). The algorithm is capable of finding improved lamination design with a considerable level of confidence as long as the numerical measures of motor and the fitness function is correctly formulated. However, the developed method has never been able to be tested for its true potential of finding the near global optimal solution (due to limitation of computer resources and restricted timescales), as it has been widely documented that the best performance of the genetic algorithms method is obtained by a large size population (>100) and a reasonable of selection pressure and small rate of mutation [2-3,8-10]. Hence, it would be useful in the future to run large population size genetic algorithm to verify the result. In the near future minor improvements to the GAs optimisation tool may include the use of real value representation of individuals, other selection methods like tournament selection etc. In the longer term and for significant improvement to the software one would consider hybridisation of the GAs with other more deterministic gradient method such as the sequential quadratic programming for quick and accurate convergence [36].

References

[1] C. Pollock and M. Wallace, 1999, "The Flux Switching Motor, A DC motor without magnets or brushes", IEEE, Industry Applications Conference 34th Annual meeting, October 1999, 1980-1984.

[2] N. Bianchi and S. Bolognani, 1997, "Brushless DC motor design: An optimisation procedure based on genetic algorithms", EMD97 IEE Conference Proceeding No.444, 16-20.

[3] J. Wieczorek, O. Gol and Z. Michalewicz, 1998, "An evolutionary algorithm for the optimal design of induction motors", IEEE Transactions on Magnetics, Nov 1998, Vol. 34, No. 6, 3883-3887.

[4] J. K. Lahteenmaki, 1997, "Optimisation of high speed motors using a genetic algorithm", EMD97 IEE Conference Publication No.444, 26-30.

[5] O. A Mohammed, 1997, "GA optimization in electric machines", IEEE, TA1-2.1 to TA1-2.5.

[6] S. Palko and T. Jokinen, 1997, "Optimisation of Squirrel Cage induction motors using finite element method and genetic algorithms", EMD97 IEE Conference Publication No.444, 21-25.

[7] M. Kim, C. Lee and H. Jung, 1998, "Multiobjective Optimal Design of Three phase Induction motor using improved Evolution Strategy", IEEE Trans. on Magentics, Vol. 34, No. 5, 2980-2983.

[8] D. E. Goldberg, 1989, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley Publishing Company, Reading MA.

[9] J. Holland, 1975, "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor.

[10] Z. Michalewicz, 1996, "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer, USA

[11] J. Schaffer et. al, 1989, "A study of control parameters affecting online performance of genetic algorithms", Proc. 3rd Int. Conf. on Genetic Algorithms, 51-60.

[12] T. Bäck, D. Fogel and Z Michalewicz, 2000, "Evolutionary Computation 1 Basic algorithms and operators", IOP Publishing Ltd., Bristol, UK.

[13] T. Bäck, D. Fogel and Z Michalewicz, 2000, "Evolutionary Computation 2 Advanced algorithms and operators", IOP Publishing Ltd., Bristol, UK.

[14] B.C. Mecrow, 1996, "New winding configurations for doubly salient reluctance machines", IEEE Trans. on Industry Applications, vol. 32, No. 6, 1348-1356.

[15] T.J.E. Miller, 1993, "Switched reluctance motors and their control", Magna Physics publishing. UK

[16] K.S Chai and C. Pollock, 2002, "Using genetic algorithms in design optimization of the flux switching motor", IEE Int. Conf. On Power Electronics, Machines and Drives, 540-545.

[17] K.S Chai and C. Pollock, 2003, "Evolutionary Computer Controlled Design of a Reluctance Motor Drive System", Annual Meeting IEEE-IAS, Salt Lake City, USA.

[18] C. Pollock, H. Pollock, R. Barron, R. Sutton, J. Coles, D. Moule, A. Court 2003, "Flux Switching Motors for Automotive Applications", Annual Meeting IEEE-IAS, Salt Lake City, USA .

[19] H. Pollock, C. Pollock, R.T. Walter and B.V. Gorti, 2003, "Low cost, high power density, flux switching machines and drives for power tools", Annual Meeting IEEE-IAS, Salt Lake City, USA.

[20] VF-09-99-A3, 27 September 1999, "OPERA-2d User Guide", Vector Fields Limited, Oxford, UK.

[21] Philip Beckley, "Electrical steels- A Handbook for Producer and Users", 1st edition, 2000, European Electrical Steels, Orb Works, Newport, South Wales.

[22] Ewen Ritchie, "Iron losses and properties of soft magnetic materials for electrical machines", Lecture note for Msc students, Institution of Energy Technology, Aalborg University, 5 Aug 1998, I14-98-S-0087.

[23] Giorgio Bertotti, "Magnetic Losses", Encyclopedia of Materials: Science and Technology, 2001, vol. 5, pp4798-4804.

[24] Giorgio Bertotti, "General properties of power losses in soft ferromagnetic materials", IEEE Trans. Mag., 1988, vol. 24, No.1, pp621-630.

[25] J. Reinert, A. Brockmeyer, and R.W. De Doncker, "Calculation of the losses in Ferro- and Ferrimagnetic materials based on the Modified Steinmetz Equation", in

Proceedings of 34th Annual Meeting of the IEEE Industry Applications Society, 1999, vol.3, pp. 2087–92.

[26] D.C. Jiles, D.L. Atherton, "Theory of ferromagnetic hysteresis", Journal of Magnetism and Magnetic Materials, 1986, vol.61, pp 48-60.

[27] A. Brockmeyer, L.Schulting, "Modelling of dynamic losses in magnetic material", EPE Proc. 1993, Brighton, UK, pp112-117.

[28] I. D. Mayergoyz, "Mathematical Models of Hysteresis", New York: Springer, 1999.

[29] C. P. Steinmetz, "On the law of hysteresis", *AIEE Transactions*, vol. 9, pp. 3–64, 1892, Reprinted under the title "A Steinmemtz contribution to the ac power revolution", introduction by J. E. Brittain, in Proceedings of the IEEE 72(2) 1984, pp. 196-221.

[30] F. Fiorillo, A. Novikov, "An improved approach to power losses in magnetic laminations under non-sinusoidal induction waveform", IEEE Trans. Magn., 1990, vol.26. No.5, pp2904-2910.

[31] R.M. Bozorth, 1993, "Ferromagnetism" IEEE Press, NewYork.

[32] K. Attallah, Z. Q. Zhu, D. Howe, "The Prediction of Iron Losses in Brushless Permanent Magnet DC Motors", Proc. ICEM, Manchester, 1992, pp 814-818.

[33] J. Hu, C. R. Sullivan, "Optimization of shapes for round wire high frequency gapped inductor windings", IEEE, IAS 33th Annual meeting, October 1998.

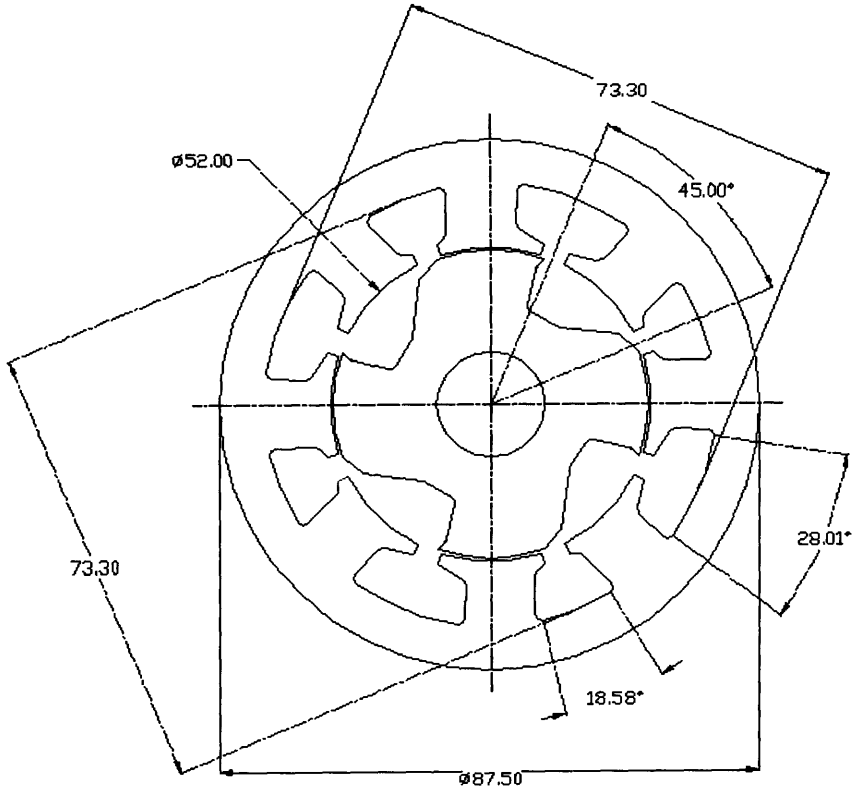
[34] S.D. Calverley, G. W. Jewell and R.J. Saunders, "Aerodynamic losses in switched reluctance machines", IEE Proc. Electr. Power Appl., Vol. 147, No.6, November 2000, pp443-448.

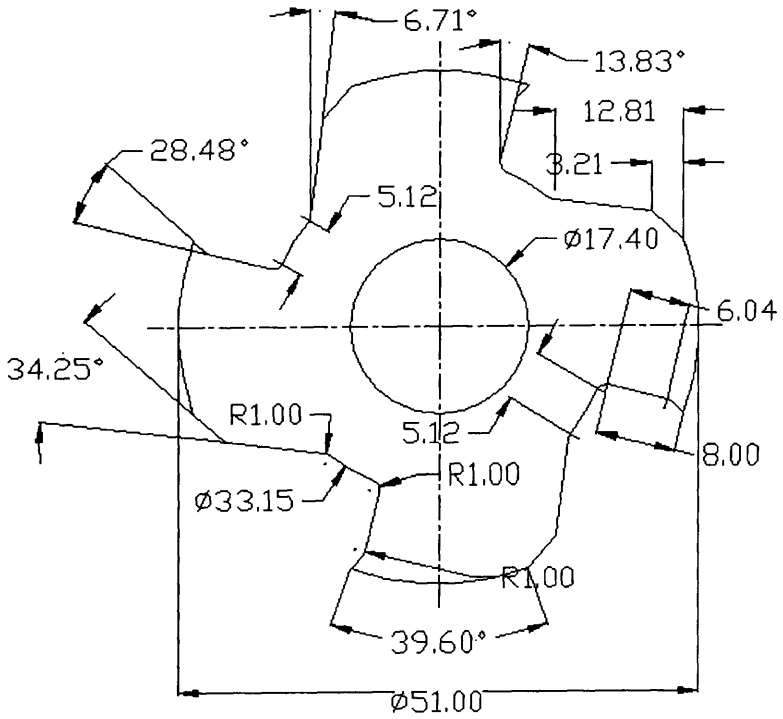
[35] John Reeve, "Computer aided design, simulation and optimisation of the flux switching machine", Ph.D thesis, University of Leicester, October 2002.

[36] H. Myung, J. Kim, and D.B. Fogel, 1995, "Preliminary investigations into a two stage method of evolutionary optimization on constrainted problems", in Proc. 4th Annual Conference Evolutionary Programming, Cambridge, MA:MIT Press, pp449-463.

Appendices

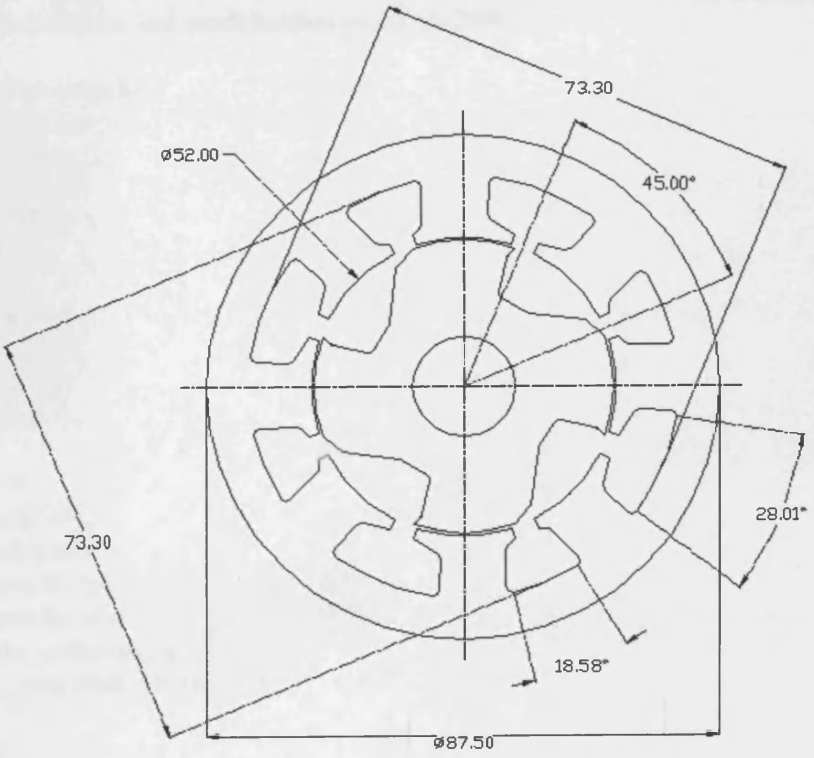
Appendix A- Lamination design of *G15i1* (in chapter 6)

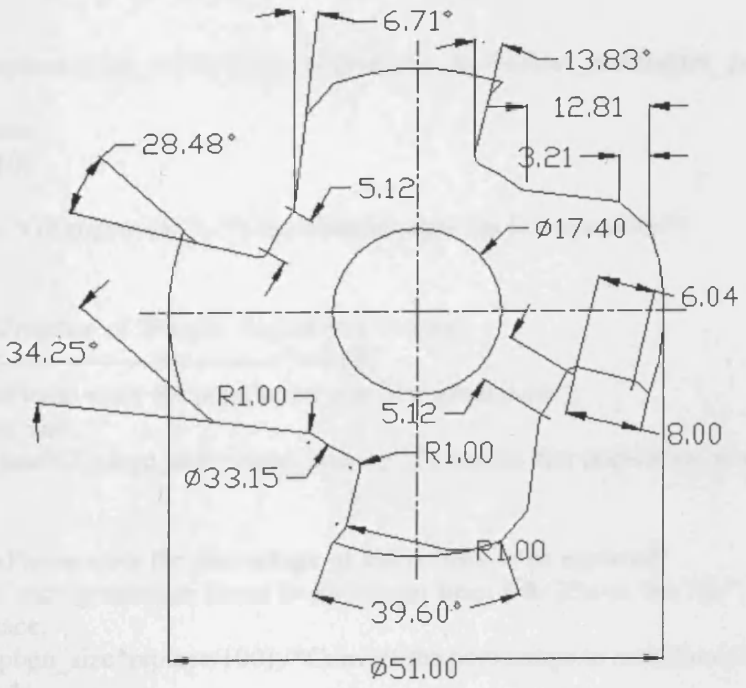




Appendices

Appendix A- Lamination design of *G15i1* (in chapter 6)





Appendix B - Implemented C code of Gaengine module

```
//Computer Optimization of electrical machine, program written by Kao Siang,Chai..
//GAengine module, last modification on 02/04/2002.

#include<iostream.h>
#include<fstream.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
#include<time.h>

char strs[200][110];
char file_ptr[12];
char* ge[]= {};
int shuffle[]= {};

int pos = 0;
char temp1[110];
char temp2[110];
int Compare1(const void*, const void*);
int Compare2(const void*, const void*);
void single_splice(int, int, int, int);
void dble_splice(int, int, int, int);

int main()
{
 int
popn_size,replace,cross_order,cross_method,mut_method,ter_method,ter_gen,n_conv,d
_fitness;
 float mut_rate;
 char input[10];

 ifstream fin("GAengine.ini"); /*Consolidating data for GA operation*/
  if(!fin)
   {
   cout<<"\tProgram of Genetic Algorithms"<<endl;
   cout<<"\t----------------------------"<<endl;
   cout<<"\nPlease enter the population size of a generation ";
   cin>>popn_size;
   if((popn_size%2))popn_size=popn_size-1;/*To ensure that population is an even
integer.*/

   cout<<"\nPlease enter the percentage of individuals to be replaced"
      <<"\nof each generation (must be an integer from 0 to 20,exc. the %) ";
   cin>>replace;
   replace=(popn_size*replace/100);/*Convert the percentage to actual no.of
replacement.*/
   if ((replace%2))replace=replace+1;/*To ensure that replacement is an even integer.*/
```

```
   cout<<"\nChoose the order of crossover operation: 1)Rank order, 2)Random order ,
3)SUS";
   cin>>cross_order;
   cout<<"\nChoose the method of crossover operation: 1)single point,"
      <<"\n2)two points ";
   cin>>cross_method;
   cout<<"\nEnter the rate of mutation(eg. 1 bit of every thousand bits = 0.001) ";
   cin>>mut_rate;
   cout<<"\nChoose a method of mutation: 1)Flip-bit, 2)Random(flip or remain) ";
   cin>>mut_method;
   cout<<"\nChoose a method to rank fitness: 1)Maximising , 2)Minimising ";
   cin>>d_fitness;
   cout<<"\nChoose a method of termination: 1)maximum generation, 2)convergence ";
   cin>>ter_method;
   if(ter_method == 1)
    {
    cout<<"\nAfter how many generation would you like to terminate the process? ";
    cin>>ter_gen;
    }
   if(ter_method == 2)
    {
    cout<<"\nWhat is the tolerance for convergence(eg. Stop!! if fitness doesn't
improve"
       <<"\nfor 5 consecutive generations : enter 5)? ";
    cin>>n_conv;
    cout<<"\nWhat is the max. generation to terminate the process,"
       <<"\nif it doesn't converge? ";
    cin>>ter_gen;
    }

 //Creating GAengine.ini file.
  ofstream fout("GAengine.ini");
   fout<<popn_size<<endl;
   fout<<replace<<endl;
   fout<<cross_order<<endl;
   fout<<cross_method<<endl;
   fout<<mut_rate<<endl;
   fout<<mut_method<<endl;
   fout<<d_fitness<<endl;
   fout<<ter_method<<endl;
   if(ter_method == 1)
   {
    fout<<ter_gen<<endl;
   }
   if(ter_method == 2)
   {
    fout<<n_conv<<endl;
    fout<<ter_gen<<endl;
   }
```

```
 fout.close();
 }
else
 {
 //Initialisation...
 fin.getline(input,sizeof(input));
 popn_size=atoi(input);
 fin.getline(input,sizeof(input));
 replace=atoi(input);
 fin.getline(input,sizeof(input));
 cross_order=atoi(input);
 fin.getline(input,sizeof(input));
 cross_method=atoi(input);
 fin.getline(input,sizeof(input));
 mut_rate=atof(input);
 fin.getline(input,sizeof(input));
 mut_method=atoi(input);
 fin.getline(input,sizeof(input));
 d_fitness=atoi(input);
 fin.getline(input,sizeof(input));
 ter_method=atoi(input);
 if(ter_method == 1)
  {
  fin.getline(input,sizeof(input));
  ter_gen=atoi(input);
  }
  if(ter_method == 2)
  {
  fin.getline(input,sizeof(input));
  n_conv=atoi(input);
  fin.getline(input,sizeof(input));
  ter_gen=atoi(input);
  }
 fin.close();

//Check if the last generation is reached.
 strcpy(file_ptr,"GEN_");
 strcat(file_ptr,ge[ter_gen-1]);
 strcat(file_ptr,".chr");

 ifstream end(file_ptr,ios::nocreate);
  if(end)
   {
   cout<<"The last generation has been reached.";
   ofstream gameover1("Finished");
   gameover1<<"1"<<endl;
   gameover1.close();
   return 1;
   }
 end.close();
```

```
//detect for the latest generation of .fit file.
  int aa=0;
  check:
     strcpy(file_ptr,"GEN_");
     strcat(file_ptr,ge[aa]);
     strcat(file_ptr,".fit");
 ifstream detect(file_ptr,ios::nocreate);
  if(!detect)
   {
   aa--;
   goto check1;
   }
  else
   {
   aa++;
   goto check;
   }
  check1:
      detect.close();

//Open the most current .fit file..
  strcpy(file_ptr,"GEN_");
  strcat(file_ptr,ge[aa]);
  strcat(file_ptr,".fit");

  ifstream in(file_ptr);
   if(!in)
    {
    cout<<"Can't open "<<file_ptr<<"."<<endl;
    }
   for(int bb=0; bb<popn_size; bb++)
     {
     in.getline(strs[bb],sizeof(strs));
     }
  in.close();
 //To test the length of each individual chromosome.
  do
  {
   pos++;
  }while(strs[0][pos] !=',');
    if(d_fitness==1)
     {
     qsort(strs, popn_size, 110, Compare1); /*Sorting function*/
     }
    if(d_fitness==2)
     {
     qsort(strs, popn_size, 110, Compare2); /*Sorting function*/
     }
 /* Convergence function */
  if(ter_method == 2)/*If user choose to have convergence termination.*/
```

```
    {
    char con_line[10]={0};
    float current=0;
    float fittest=0;
    int e_time=0;

    strncpy(con_line,strs[0]+pos+1,9);
    current=atof(con_line);
    ifstream contrast("converge.txt");/*Read previous fittest value, if exist*/
    if(!contrast)
     {
      goto destination;
     }
    else
     {
      contrast.getline(con_line,sizeof(con_line));
      e_time=atoi(con_line);
      contrast.getline(con_line,sizeof(con_line));
      fittest=atof(con_line);
     }
    destination:
    ofstream converge("converge.txt");/*Store present fittest value for convergence
test.*/
     if(current > fittest)
      {
       converge<<"0"<<endl;
       converge<<current<<endl;
      }
     if(current <= fittest)
      {
       converge<<e_time+1<<endl;
       converge<<fittest<<endl;
      }
    converge.close();
    contrast.close();
    if(current > fittest)
     {
     e_time=0;
     }
    if(current <= fittest)
     {
     e_time=e_time+1;
     }
    if(e_time >= n_conv ) /* Will converge if fitness doesn't increase after # generation
*/
     {
     cout<<"Convergence in the fitness value... stagnant!!!";
     ofstream gameover("Finished");
     gameover<<"1"<<endl;
     gameover.close();
```

```
      return 1;
     }/* Convergence function ends*/
    }
  //Remove fitness value & ',' from chromosome.
   for (int cc = 0; cc < popn_size; cc++)
   {
    strcpy(temp1,strs[cc]);
    for(int dd=0; dd<110; dd++)
     {
      strs[cc][dd]=0;
     }
    strncpy(strs[cc],temp1,pos);
   }

   srand((unsigned)time(NULL)); /*Seed for the random numbers generator.*/

   if(cross_order==1)/*Cross-over in rank order,fittest pair..next fittest pair...*/
    {
    int c_pt,stop;
    int ee=0;
    for(int a=0; a<110; a++)
     {
      temp1[a]=0;
      temp2[a]=0;
     }
    c_pt=1+(int)((pos-1.0)*rand()/(RAND_MAX+1.0));/*Generate a random crossover
point.*/
    stop=popn_size-replace;/*No. of individuals to be crossover*/

     if(cross_method==1)
      {
       single_splice(c_pt, stop, pos, ee);/*single splice.*/
      }
     if(cross_method==2)
      {
       int sc_pt;
       norpt:
           sc_pt=1+(int)((pos-1.0)*rand()/(RAND_MAX+1.0));/*Generate a 2nd
random crossover point.*/
        if((sc_pt==c_pt)||(sc_pt<c_pt))
         {
          goto norpt;
         }
       dble_splice(c_pt, sc_pt, stop, ee);/*double splice*/
      }

   }/*Ranked cross-over ends.*/

  if(cross_order==2)/*Cross-over in random order. */
   {
```

```
    int c_pt1,stop1,swap1,swap2,t_sw1,t_sw2;
    int ff=0;
    for(int d=0; d<110; d++)
     {
      temp1[d]=0;
      temp2[d]=0;
     }
    c_pt1=1+(int)((pos-1.0)*rand()/(RAND_MAX+1.0));/*Generate a random
crossover point.*/
    stop1=popn_size-replace;/*No. of individuals to be crossover*/

    for(int sw=0; sw<800; sw++)/*shuffle 400 times*/
     {
      redo:
        swap1=(int)((popn_size-(replace+1.0))*rand()/(RAND_MAX+1.0));
        swap2=(int)((popn_size-(replace+1.0))*rand()/(RAND_MAX+1.0));
      if(swap1==swap2)goto redo;
        t_sw1=shuffle[swap1];/*Ready for the swap sequence.*/
        t_sw2=shuffle[swap2];
        shuffle[swap1]=t_sw2;
        shuffle[swap2]=t_sw1;/*Swapping had taken place.*/
     }/*Shuffling ended*/

     if(cross_method == 1)
      {
      single_splice(c_pt1, stop1, pos, ff);/*single splice*/
      }
     if(cross_method == 2)
      {
      int sc_pt1;
       norpt1:
           sc_pt1=1+(int)((pos-1.0)*rand()/(RAND_MAX+1.0));/*Generate a 2nd
random crossover point.*/
        if((sc_pt1==c_pt1)||(sc_pt1<c_pt1))
         {
          goto norpt1;
         }
       dble_splice(c_pt1, sc_pt1, stop1, ff);/*double splice*/
      }
   }/*Random cross-over ends*/
   /*Mutation*/
   int total_bits,bit_change;
    total_bits=popn_size*pos;
    bit_change=mut_rate*(float)total_bits;
    for(int gg=0; gg<bit_change; gg++)
      {
      int mut_bit,row,col;
      mut_bit =(int)((total_bits-1.0)*rand()/(RAND_MAX+1.0));/*Generate a no.
within range.*/
      row=mut_bit/pos;
```

```
      col=mut_bit%pos;
      if(mut_method==1)/*flip bit begins*/
       {
        if(strs[row][col]=='1')
         {
          strs[row][col]='0';
         }
        if(strs[row][col]=='0')
         {
          strs[row][col]='1';
         }
       }/*flip bit ends*/
      if(mut_method==2)/*Random method, flip or unchanged*/
       {
        int decide;
        decide=(int)(1.0*rand()/(RAND_MAX+1.0));
        if(decide==1)
          {
           strs[row][col]='1';
          }
        if(decide==0)
          {
           strs[row][col]='0';
          }
       }/*Random method ends.*/

     }/*Mutation ends*/
 //Replacement of weak individuals with new individuals.
  int get_rid;
  get_rid=popn_size-replace;
 for(int hh=get_rid; hh<popn_size; hh++)
   {
    for(int f=0; f<110; f++)//Clear memory in temp1.
      {
       strs[hh][f]=0;
      }
    for(int ii=0; ii<pos; ii++)
      {
       char rp;
       if((rand()%2)) rp='1';
       else rp='0';
       strs[hh][ii]=rp;
      }

   }

//Create .chr file..
strcpy(file_ptr,"GEN_");
strcat(file_ptr,ge[aa+1]);
strcat(file_ptr,".chr");
```

```
   ofstream print(file_ptr);
    for(int kk=0; kk<popn_size; kk++)
      {
       print<<strs[kk]<<endl;
      }
    print.close();
   cout<<file_ptr<<" is created.";
  }
 ofstream gameover2("Finished"); /*Create indicator file for controller. */
 gameover2<<"0"<<endl;
 gameover2.close();

 return 0;
}

//function to rank the chromosome...greater is better.
int Compare1 (const void* a_PTR, const void* b_PTR)
{
 int nums;
 float num1,num2;
 char temp3[10],temp4[10];
 int pos1=pos+1;
 strcpy(temp1,((char*)(a_PTR)));
 strncpy(temp3,temp1+pos1,9);
 strcpy(temp1,((char*)(b_PTR)));
 strncpy(temp4,temp1+pos1,9);

 num1=atof(temp3);
 num2=atof(temp4);

 if(num2 < num1) nums=-1;
 if(num2 == num1) nums=0;
 if(num2 > num1) nums=1;

 return nums;
}

//function to rank the chromosome...lesser is better.
int Compare2 (const void* a_PTR, const void* b_PTR)
{
 int nums;
 float num1,num2;
 char temp3[10],temp4[10];
 int pos1=pos+1;
 strcpy(temp1,((char*)(a_PTR)));
 strncpy(temp3,temp1+pos1,9);
 strcpy(temp1,((char*)(b_PTR)));
 strncpy(temp4,temp1+pos1,9);
```

```
 num1=atof(temp3);
 num2=atof(temp4);

 if(num2 > num1) nums=-1;
 if(num2 == num1) nums=0;
 if(num2 < num1) nums=1;

 return nums;
}

void single_splice(int cut_pt , int stp, int pos, int count)
{
 do
  {
   strncpy(temp1, strs[count]+cut_pt, pos - cut_pt);/*store exchange part of Parent1*/
   strncpy(temp2, strs[count+1]+cut_pt, pos - cut_pt);/*store exchange part of Parent2*/
   for(int b=cut_pt; b<110; b++)
    {
     strs[count][b]=0;/*Clear bits of Parent1 to be exchanged.*/
     strs[count+1][b]=0;/*Clear bits of Parent2 to be exchanged .*/
    }
   strcat(strs[count],temp2);/*Child1 formed*/
   strcat(strs[count+1],temp1);/*Child2 formed*/
   count++;
   count++;
  }while(count != stp);

}
void dble_splice(int cut_pt, int cut_pt1, int stp, int count)
{
 do
  {
   int temp_count=0;
   strncpy(temp1, strs[count]+cut_pt, cut_pt1 - cut_pt);/*store exchange part of
Parent1*/
   strncpy(temp2, strs[count+1]+cut_pt, cut_pt1 - cut_pt);/*store exchange part of
Parent2*/
   for(int b=cut_pt; b<cut_pt1; b++)
    {
     strs[count][b]=temp2[temp_count];
     strs[count+1][b]=temp1[temp_count];
     temp_count++;
    }
   count++;
   count++;
  }while(count != stp);

}
```

Appendix C - Implemented C code of decode module

```
//Computer Optimization of electrical machine, program written by Kao Siang,Chai.
//Decode module, last modification on 02/04/2002.

#include<iostream>
#include<fstream>
#include<string>
#include<stdlib>
#include<math>
#include<ctype>
#include<time>

char input[31];
char input1[31];
char init_chromosome[102];
char file_ptr[31];
char ch_file_ptr[31];
char* gno[]= { };

int convert_b_d(char* ,int);
float sum(float, float, int, int);

int main()
{
 int chro_line =0;
 int pos=0;
 int c_len=0;
 int var,val,no,g_size,sh;
 int extra[50];
 int bits[50];
 float upp_limit[50];
 float low_limit[50];
 char runp[50][31];
 char runp1[150][31];
 char pa[50][5];
 char pa1[150][5];
 char description[50][81];
 float total[50];
 char temp_bits[101]={"0"};
 char single[100]={0};

//initializing of program..checked if .ini file exist, if not...
 ifstream in("decode.ini");

 if(!in)
 {
   cout<<"Program for decoding the chromosome\n";
  do{
```

```
    cout<<"Please enter the amount of variables you want:";
    cin.getline(input, sizeof(input));
    var = atoi(input);
    }while((var<1)||(var>30));
  int vd=0;
  int vd1=0;
//Requesting for the design parameters.
   for(int i=0; i<var; i++)
    {
     do{
       cout<<"\nHow many bits are there in variable "<< i+1<<" : ";
       cin.getline(input, sizeof(input));
       bits[i] = atoi(input);
      }while((bits[i]<1)||(bits[i]>20));

       cout<<"\nWhat is the upper limit of the variable :";
       cin.getline(input, sizeof(input));
       upp_limit[i] = atof(input);


       cout<<"\nWhat is the lower limit of the variable :";
       cin.getline(input, sizeof(input));
       low_limit[i] = atof(input);

      cout<<"\nPlease name your variable (max 30 letters) :";
      cin.getline(runp[i], sizeof(runp));
      cout<<"\nNow enter a number for the PA :";
      cin.getline(pa[i], sizeof(pa));
      cout<<"\nGive a brief description for your variable\n";
      cin.getline(description[i], sizeof(description));
      cout<<"\nDo you want any extra part(s) to be included in variable "<<i+1
        <<"\n(max. 3 extra parts or enter 0 if you don't need any.)"<<endl;
      cin.getline(input, sizeof(input));
      extra[i] = atoi(input);
      vd1 +=extra[i];
      sh=0;
       for(int ex=vd; ex<vd1; ex++)
         {
          cout<<"\nPlease name your extra part no."<<sh+1<<" for variable"<<i+1
            <<"\n(max 30 letters) :";
          cin.getline(runp1[ex], sizeof(runp1));
          cout<<"\nNow enter a number for the PA :";
          cin.getline(pa1[ex], sizeof(pa1));
          vd++;
          sh++;
         }
    }

//Create .ini file, if it does not exist.
```

```
    ofstream fout("decode.ini");
     int zt=0;
     int zt1=0;
     if(!fout)
      {
       cout<<"Can't created .ini file.\n";
       return 1;
      }

      fout<<var<<endl;

      for(int z=0; z<var; z++)
      {
       fout<<"bits ="<<bits[z]<<endl;
       fout<<"upp_limit ="<<upp_limit[z]<<endl;
       fout<<"low_limit ="<<low_limit[z]<<endl;
       fout<<"runp ="<<runp[z]<<endl;
       fout<<"pa ="<<pa[z]<<endl;
       fout<<"description ="<<description[z]<<endl;
       fout<<"extra ="<<extra[z]<<endl;
       zt1 +=extra[z];
       for(int et=zt; et<zt1; et++)
         {
          fout<<"additional ="<<runp1[et]<<endl;
          fout<<"more ="<<pa1[et]<<endl;
          zt++;
         }
      }
     fout.close();
  }
//if .ini file already exist.. open it and proceed setup with original setting.
  else
  {
   in.getline(input,sizeof(input));
   var = atoi(input);
   int y=0;
   int yy=0;
   do
    {
     in.getline(input, sizeof(input));
     if(strstr(input,"bits"))
      {
       strcpy(input1, input+6);
       bits[y]=atoi(input1);
      }
     if(strstr(input,"upp_limit"))
      {
       strcpy(input1, input+11);
       upp_limit[y]=atof(input1);
      }
```

```
   if(strstr(input,"low_limit"))
    {
     strcpy(input1, input+11);
     low_limit[y]=atof(input1);
    }
   if(strstr(input,"runp"))
    {
     strcpy(input1, input+6);
     strcpy(runp[y], input1);
    }
   if(strstr(input,"pa"))
    {
     strcpy(input1, input+4);
     strcpy(pa[y], input1);
    }
   if(strstr(input,"description"))
    {
     strcpy(input1, input+13);
     strcpy(description[y], input1);
    }
   if(strstr(input,"extra"))
    {
     strcpy(input1, input+7);
     extra[y]=atoi(input1);
     y++;
    }
   if(strstr(input,"additional"))
    {
     strcpy(input1, input+12);
     strcpy(runp1[yy], input1);
    }
   if(strstr(input,"more"))
    {
     strcpy(input1, input+6);
     strcpy(pa1[yy], input1);
     yy++;
    }
  }while(!in.eof());
}
in.close();/*Initialisation completed.*/

/*Create GEN_001.chr if it doesn't exist.*/
ofstream start("GEN_001.chr", ios::noreplace);
if(!start)
 {
  goto quick;/*Jump to 'quick' if GEN_001.chr already exist.*/
 }
else/*Start creation of GEN_001.chr file.*/
 {
  ifstream take("GAengine.ini");
```

```
   take.getline(input,sizeof(input));//Read in info. on size of a generation.
   g_size=atoi(input);
   take.close();
   for(int nn=0; nn<var; nn++)
    {
     c_len+=bits[nn];//determine the length of an individual.
    }
    srand((unsigned)time(NULL));//Seed for random generator.
    for(int bb=0; bb<g_size; bb++)
     {
      for(int cc=0; cc<c_len; cc++)
        {
         char p;
         if((rand()%2)) p='1';
         else p='0';
         single[cc]=p;
        }
       start<<single<<endl;
       for(int ee=0; ee<100; ee++)
         {
          single[ee]=0;
         }
     }
  }
  start.close();

//Detecting the latest chromosome file available...

quick:
    no = 0;
    create:
         strcpy(ch_file_ptr,"GEN");
         strcat(ch_file_ptr,"_");
         strcat(ch_file_ptr,gno[no]);
         strcat(ch_file_ptr,".chr");

  ifstream test(ch_file_ptr, ios::nocreate);
  if(!test)
  {
   no--;
   goto create1;
  }
  else
  {
   no++;
   goto create;
  }
//Latest chromosome file found...
  create1:
        strcpy(ch_file_ptr,"GEN");
```

```
        strcat(ch_file_ptr,"_");
        strcat(ch_file_ptr,gno[no]);
        strcat(ch_file_ptr,".chr");
  test.close();

//Test for the no. of individuals in the current generation.
  ifstream testl(ch_file_ptr);

  if(!testl)
  {
   cout<<"Can't open output file.\n";
   return 1;
  }
  do
  {
   testl.getline(init_chromosome,sizeof(init_chromosome));
   chro_line++;
  }while(!testl.eof());
  testl.close();

//latest generation of chromosome file opened.
  ifstream fin(ch_file_ptr);

  if(!fin)
  {
  cout<<"Can't open output file.\n";
  return 1;
  }
//Process 'raw' data of chromosome file.
 for(int ra=0; ra<chro_line-1; ra++)
  {
   fin.getline(init_chromosome,sizeof(init_chromosome));
   pos = 0;

    for(int f=0; f<var; f++)
    {
     strncpy(temp_bits, init_chromosome+pos, bits[f]);
     pos +=bits[f];
     for(int w=bits[f]; w<101; w++)
      {
       temp_bits[w] = 0;
      }
//Conversion of 'raw' data into 'meaningful' data to simulator.
     val = convert_b_d(temp_bits, bits[f]);
     total[f] = sum(upp_limit[f], low_limit[f], bits[f], val);
    }

//Creating .comi file.
    strcpy(file_ptr,"G");
    strcat(file_ptr,gno[no]);
```

```
    strcat(file_ptr,"I");
    strcat(file_ptr,gno[ra]);
    strcat(file_ptr,".ind");

    ofstream fout(file_ptr);
    int st=0;
    int st1=0;
    for(int j=0; j<var; j++)
     {
      fout<<"/"<<description[j]<<endl;
      fout<<"RUNP "<<runp[j]<<endl;
      fout<<"RUNP PA"<<pa[j]<<"="<<total[j]<<endl;
      fout<<"|"<<endl;
      fout<<"\n";
      st1 +=extra[j];
      for(int er=st; er<st1; er++)
       {
        fout<<"RUNP "<<runp1[er]<<endl;
        fout<<"RUNP PA"<<pa1[er]<<"="<<total[j]<<endl;
        fout<<"|"<<endl;
        fout<<"\n";
        st++;
       }
     }
    fout.close();
  }
  fin.close();
  cout<<chro_line-1<<" files created for generation "<<no+1<<endl;

 return 0;

}

//function for conversion of 'raw' data to 'meaningful' data.
float sum(float a, float b, int c, int d)
{
 return (((a-b)/(float)(pow(2,c)-1))*(float)d)+b;
}
int convert_b_d(char* temp_bits, int bits)
{  int temp_dec=0;
    int nbits = bits-1;
  for(int i=0; i<bits; i++)
    {

     temp_dec += ((temp_bits[i]-48)*(pow(2, nbits))) ;
     nbits--;
    }
  return temp_dec;
}
```

Appendix D - Implemented code (in Visual Basic) – instantaneous current module

```
'Computer Optimization of electrical machine, program written by Kao Siang,Chai..
'Current module, last modification on 26/07/2002.

Option Explicit
Dim xl As New Excel.Application
Dim wb As Excel.Workbook
Dim ws As Excel.Worksheet
Dim fld_curr(1 To 20) As Single
Dim arm_curr(1 To 20) As Single
Dim rot_angle(1 To 20) As Single
Dim fld_turns As Integer
Dim arm_turns As Integer
Dim fld_dia As String
Dim arm_dia As String
Dim gArray(1 To 250) As String  'Array for storing .csr data
Dim Gen$                        'String indicating generation
Dim gen_count As Integer        'Generation number
Dim Ind$                        'String indicating individual design
Dim ind_count As Integer        'States the number of designs in current generation
Dim error As String             'error flag


Private Sub Form_Initialize()

Generation_search   'Call generation_search function
Nos_Individual      'Call nos_individual function

Open_file 'Call Open_file function
Set wb = xl.Workbooks.Open(App.Path & "\fitness.xls")
Set ws = wb.Sheets("IM-test1")
xl.Visible = True 'Make fitness.xls visible to user

Dim cCol 'Column Counter 1 = A, 2 = B.. so on
Dim cRow 'Row Counter
Dim cCount 'For retrieving Array content
cCol = 2 'Start at Column B
cCount = 3
'Write data to cells
Do
For cRow = 1 To 11
ws.Cells(cRow, cCol).Value = gArray(cCount)
cCount = cCount + 1
Next
cCol = cCol + 1
Loop While cCol < 13 'End at column L
```

```
error = "0"
If ws.Range("B4").Value = 0 Then
Dim fname As String
Dim fname1 As String
Dim fname2 As String
error = "1"

fname = App.Path    'Current directory
   If Right$(fname, 1) <> "\" Then fname = fname & "\"

      fname1 = fname & "error.csr"
      fname2 = fname & "LG" & Gen$ & "i" & Ind$ & ".csr"

      FileCopy fname1, fname2
   GoTo Nottrap:
End If

ws.Calculate

Dim trial As Integer
Dim sg_value As Integer
Dim arm_value As Integer
trial = 0

If ws.Range("B88").Value / ws.Range("C88").Value > 2.5 Then
Inc_fld_dia 'Increase field wire size function
Inc_fld_dia
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71") = sg_value
ws.Calculate
End If

ws.Range("B71") = ws.Range("B88").Value 'Adjustment before commencing
optimizing the wire configuration
ws.Range("C71") = ws.Range("C88").Value

Do
If ws.Range("I72").Value - ws.Range("I94").Value < -100 Then
Inc_fld_dia 'Increase field wire size function
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71") = sg_value
ws.Calculate
Delay 1
If ws.Range("I72").Value - ws.Range("I94").Value < -50 Then
Inc_arm_dia 'Increase armature wire size function
ws.Calculate
sg_value = ws.Range("C88").Value
ws.Range("C71") = sg_value
```

```
ws.Calculate
End If
End If

If ws.Range("I72").Value - ws.Range("I94").Value > 80 Then
dec_arm_dia 'decrease armature wire size function
ws.Calculate
sg_value = ws.Range("C88").Value
ws.Range("C71") = sg_value
ws.Calculate
Delay 1
If ws.Range("I72").Value - ws.Range("I94").Value > 50 Then
dec_fld_dia 'decrease field wire size function
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71") = sg_value
ws.Calculate
End If
End If

If ws.Range("B88").Value / ws.Range("C88").Value <= 2.5 Then
If ws.Range("I72").Value - ws.Range("I94").Value < -65 And ws.Range("I72").Value -
ws.Range("I94").Value >= -100 Then
Inc_arm_dia 'Increase arm wire size function
ws.Calculate
sg_value = ws.Range("C88").Value
ws.Range("C71") = sg_value
ws.Calculate
End If
End If

If ws.Range("I72").Value - ws.Range("I94").Value < -30 And ws.Range("I72").Value -
ws.Range("I94").Value >= -65 Then
Inc_fld_dia 'Increase field wire size function
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71") = sg_value
ws.Calculate
End If

If ws.Range("I72").Value - ws.Range("I94").Value < 0 And ws.Range("I72").Value -
ws.Range("I94").Value >= -30 Then
arm_value = ws.Range("C88").Value - 2
ws.Range("C71") = arm_value
ws.Calculate
End If

If ws.Range("B88").Value / ws.Range("C88").Value > 2.5 Then
Inc_fld_dia 'Increase field wire size function
ws.Calculate
```

```
sg_value = ws.Range("B88").Value
ws.Range("B71") = sg_value
ws.Calculate
End If

trial = trial + 1
Loop While trial < 4 'Try different combination for n times

If ws.Range("B17").Value * ws.Range("I93").Value <= 200 Then 'Improving losses
(only when freq of armature back iron flux density is less than 200hz)
Dim Eff_1 As Single
Dim Eff_2 As Single

sg_value = ws.Range("C88").Value
ws.Range("C71") = sg_value
Eff_2 = ws.Range("I82").Value

Do
Inc_fld_dia
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71").Value = sg_value
ws.Calculate
Eff_1 = Eff_2
Eff_2 = ws.Range("I82").Value
Loop While ws.Range("I72").Value - ws.Range("I94").Value >= 0 And Eff_2 > Eff_1

If Eff_2 < Eff_1 Or ws.Range("I72").Value - ws.Range("I94").Value < 0 Then
dec_fld_dia
ws.Calculate
sg_value = ws.Range("B88").Value
ws.Range("B71").Value = sg_value
ws.Calculate
End If

End If 'End of improving losses condition

Dim Col
Dim a

fld_turns = ws.Range("B71")
arm_turns = ws.Range("C71")
fld_dia = ws.Range("B79")
arm_dia = ws.Range("C79")

Col = 2
For a = 1 To 20
fld_curr(a) = ws.Cells(37, Col).Value
arm_curr(a) = ws.Cells(62, Col).Value
rot_angle(a) = ws.Cells(54, Col).Value
```

```
Col = Col + 5
Next

write_current 'write .bmd file
Nottrap:
Avoid_error

wb.Close SaveChanges:=False
xl.Quit
Set xl = Nothing

End
End Sub

Public Sub Open_file()
Dim fnum As Integer    'for assigning file number
Dim one_line As String
Dim counter As Integer 'Counter for array
Dim fname As String

   fname = App.Path    'Current directory
   If Right$(fname, 1) <> "\" Then fname = fname & "\"

   If ind_count < 10 Then
   Ind$ = "00" + CStr(ind_count)   'Don't use Str(), as it'll create a blank space
   ElseIf ind_count > 9 Then
   Ind$ = "0" + CStr(ind_count)
   ElseIf ind_count > 99 Then
   Ind$ = CStr(ind_count)
   End If

   fname = fname & "G" & Gen$ & "i" & Ind$ & ".csr" 'Formed the file name

   fnum = FreeFile 'Assign a file number
   Open fname For Input As fnum 'Open .csr file
   counter = 1
   Do While Not EOF(fnum)
      ' Read a line.
      Line Input #fnum, one_line

      If Left$(one_line, 1) <> " " Then
      gArray(counter) = Mid$(one_line, 41)
      counter = counter + 1
      End If

      If Left$(one_line, 7) = "Area of" Then
      counter = counter - 1
      End If

   Loop
```

```
 Close fnum 'return assigned number after used
End Sub

Public Sub Generation_search()
Dim a$

gen_count = 1

Check:
If gen_count < 10 Then
Gen$ = "00" + CStr(gen_count)   'Don't use Str(), as it'll create a blank space
ElseIf gen_count > 9 Then
Gen$ = "0" + CStr(gen_count)
ElseIf gen_count > 99 Then
Gen$ = CStr(gen_count)
End If

a$ = "G" & Gen & "i001.csr"

On Error GoTo FileDoesntExist:
If FileLen(a$) > 0 Then
gen_count = gen_count + 1
GoTo Check:
End If
FileDoesntExist:
gen_count = gen_count - 1
If gen_count < 10 Then
Gen$ = "00" + CStr(gen_count)   'Don't use Str(), as it'll create a blank space
ElseIf gen_count > 9 Then
Gen$ = "0" + CStr(gen_count)
ElseIf gen_count > 99 Then
Gen$ = CStr(gen_count)
End If
Exit Sub

End Sub

Public Sub Nos_Individual()
Dim a$

ind_count = 1

Check:
If ind_count < 10 Then
Ind$ = "00" + CStr(ind_count)   'Don't use Str(), as it'll create a blank space
ElseIf ind_count > 9 Then
Ind$ = "0" + CStr(ind_count)
ElseIf ind_count > 99 Then
Ind$ = CStr(ind_count)
End If
```

```
a$ = "G" & Gen$ & "i" & Ind$ & ".csr"

On Error GoTo FileDoesntExist:
If FileLen(a$) > 0 Then
ind_count = ind_count + 1
GoTo Check:
End If
FileDoesntExist:
ind_count = ind_count - 1
Exit Sub
End Sub

Public Sub write_current()
Dim fnum As Integer
Dim fname As String

fname = App.Path    'Current directory
If Right$(fname, 1) <> "\" Then fname = fname & "\"
fname = fname & "G" & Gen$ & "I" & Ind$ & ".bmd"

fnum = FreeFile
Open fname For Output As fnum
Dim j
Print #fnum, "/field wire size:" & fld_dia
Print #fnum, "/armature wire size:" & arm_dia
Print #fnum, "/field turns:" & fld_turns
Print #fnum, "/armature turns:" & arm_turns
For j = 1 To 20
Print #fnum, "/coil config"
Print #fnum, "RUNP coil_fp"
Print #fnum, "RUNP PA01=" & fld_curr(j) & " PA02=" & fld_turns & " PA03=" &
arm_turns & " PA04=" & arm_curr(j)
Print #fnum, "|"
Print #fnum, " "
Print #fnum, "/rotor angle"
Print #fnum, "RUNP rotation"
Print #fnum, "RUNP PA01=" & rot_angle(j) & " FILE='IRON" & j & "'" & " +STOR"
Print #fnum, "|"
Print #fnum, " "
Next

Close fnum
End Sub

Public Sub Avoid_error()
Dim fnum As Integer
Dim fname As String

fname = App.Path    'Current directory
```

```
If Right$(fname, 1) <> "\" Then fname = fname & "\"
fname = fname & "YN_error"

fnum = FreeFile
Open fname For Output As fnum
Print #fnum, error
Close fnum

End Sub

Public Sub Inc_arm_dia()
Dim wire_dia
Dim wire_type As Integer

wire_dia = ws.Range("C79")

If wire_dia = 0.2 Then 'Determining wire size
wire_type = 91
End If
.
.
If wire_dia = 2 Then
wire_type = 104
End If

wire_type = wire_type + 1 'Increasing wire size
ws.Range("C79") = ws.Cells(wire_type, 1).Value

End Sub

Public Sub Inc_fld_dia()
Dim wire_dia
Dim wire_type As Integer

wire_dia = ws.Range("B79")
If wire_dia = 0.2 Then 'Determining wire size
wire_type = 91
End If
.
.
If wire_dia = 2 Then
wire_type = 104
End If

wire_type = wire_type + 1 'Increasing wire size
ws.Range("B79") = ws.Cells(wire_type, 1).Value
End Sub
Public Sub dec_arm_dia()
Dim wire_dia
Dim wire_type As Integer
```

```
wire_dia = ws.Range("C79")
If wire_dia = 0.2 Then 'Determining wire size
wire_type = 91
End If
.
.
If wire_dia = 2 Then
wire_type = 104
End If

wire_type = wire_type - 1 'Increasing wire size
ws.Range("C79") = ws.Cells(wire_type, 1).Value

End Sub
Public Sub dec_fld_dia()
Dim wire_dia
Dim wire_type As Integer

wire_dia = ws.Range("B79")
If wire_dia = 0.2 Then 'Determining wire size
wire_type = 91
End If
.
.
If wire_dia = 2 Then
wire_type = 104
End If

wire_type = wire_type - 1 'Increasing wire size
ws.Range("B79") = ws.Cells(wire_type, 1).Value

End Sub
Public Sub Delay(HowLong As Date)
Dim TempTime As Date
TempTime = DateAdd("s", HowLong, Now)
While TempTime > Now
DoEvents
Wend
End Sub
```

Appendix E - Implemented code (in Visual Basic) for fitness module

```
'Computer Optimization of electrical machine, program written by Kao Siang,Chai..
'Fitness module, last modification on 31/07/2002.

Option Explicit
Dim xl As New Excel.Application
Dim wb As Excel.Workbook
Dim ws As Excel.Worksheet
Dim gArray(1 To 1663) As String  'Array for storing .csr data
Dim Gen$                  'String indicating generation
Dim gen_count As Integer      'Generation number
Dim Ind$                  'String indicating individual design
Dim ind_count As Integer      'States the number of designs in current generation
Dim fitness() As Single
Dim i As Integer

Private Sub Form_Initialize()

Generation_search  'Call generation_search function
Nos_Individual     'Call nos_individual function

ReDim fitness(1 To ind_count) As Single 'Will clear memory if declared within for loop

For i = 1 To ind_count

Open_flux_linkage 'Call Open_flux_linkage function
Set wb = xl.Workbooks.Open(App.Path & "\fitness2.xls")
Set ws = wb.Sheets("IM-test1")
xl.Visible = True 'Make fitness.xls visible to user

Dim fRow
Dim fCol
Dim fCount
fCount = 3
fCol = 2
Do
For fRow = 1 To 7
ws.Cells(fRow, fCol).Value = gArray(fCount)
fCount = fCount + 1
Next
fCol = fCol + 1
Loop While fCol < 13

Lamination_dimension 'Get info on area of polygons
Dim cRow 'Row Counter
Dim cCount 'For retrieving Array content
cCount = 10
```

```
'Write data to cells
For cRow = 142 To 184
ws.Cells(cRow, 2).Value = gArray(cCount)
cCount = cCount + 1
Next

Open_iron_losses 'Call Open_iron_losses function

Dim Col 'Column Counter 1 = A, 2 = B.. so on
Dim Row 'Row Counter
Dim nCount 'For retrieving Array content
Col = 2 'Start at Column B
nCount = 3
'Write data to cells
Do
For Row = 186 To 268
ws.Cells(Row, Col).Value = gArray(nCount)
nCount = nCount + 1
Next
Col = Col + 1
Loop While Col < 22 'End at column U

If ws.Range("B4").Value = 0 Then
fitness(i) = 1
GoTo Nottrap:
End If

Wire_config 'Retrieve wire info
Dim wRow
Dim wCount
wCount = 1

For wRow = 165 To 168
ws.Cells(wRow, 10).Value = gArray(wCount)
wCount = wCount + 1
Next

ws.Calculate 'Calculate fitness
fitness(i) = ws.Range("I108")

Nottrap:
wb.Close SaveChanges:=False
xl.Quit
Set xl = Nothing
Next

write_fitness 'write .fit file
End
End Sub
```

```
Public Sub Open_flux_linkage()
Dim fnum As Integer    'for assigning file number
Dim one_line As String
Dim counter As Integer 'Counter for array
Dim fname As String

   fname = App.Path    'Current directory
   If Right$(fname, 1) <> "\" Then fname = fname & "\"

   If i < 10 Then
   Ind$ = "00" + CStr(i)   'Don't use Str(), as it'll create a blank space
   ElseIf i > 9 Then
   Ind$ = "0" + CStr(i)
   ElseIf i > 99 Then
   Ind$ = CStr(i)
   End If

   fname = fname & "G" & Gen$ & "i" & Ind$ & ".csr" 'Formed the file name

   fnum = FreeFile 'Assign a file number
   Open fname For Input As fnum 'Open .csr file
   counter = 1
   Do While Not EOF(fnum)
      ' Read a line.
      Line Input #fnum, one_line

      If Left$(one_line, 1) <> " " Then
      gArray(counter) = Mid$(one_line, 41)
      counter = counter + 1

      .
      .

      End If

   Loop

 Close fnum 'return assigned number after used
End Sub

Public Sub Generation_search()
Dim a$

gen_count = 1

Check:
If gen_count < 10 Then
Gen$ = "00" + CStr(gen_count)   'Don't use Str(), as it'll create a blank space
ElseIf gen_count > 9 Then
Gen$ = "0" + CStr(gen_count)
```

```
ElseIf gen_count > 99 Then
Gen$ = CStr(gen_count)
End If

a$ = "G" & Gen & "i001.csr"

On Error GoTo FileDoesntExist:
If FileLen(a$) > 0 Then
gen_count = gen_count + 1
GoTo Check:
End If
FileDoesntExist:
gen_count = gen_count - 1
If gen_count < 10 Then
Gen$ = "00" + CStr(gen_count)   'Don't use Str(), as it'll create a blank space
ElseIf gen_count > 9 Then
Gen$ = "0" + CStr(gen_count)
ElseIf gen_count > 99 Then
Gen$ = CStr(gen_count)
End If
Exit Sub

End Sub

Public Sub Nos_Individual()
Dim a$

ind_count = 1

Check:
If ind_count < 10 Then
Ind$ = "00" + CStr(ind_count)   'Don't use Str(), as it'll create a blank space
ElseIf ind_count > 9 Then
Ind$ = "0" + CStr(ind_count)
ElseIf ind_count > 99 Then
Ind$ = CStr(ind_count)
End If

a$ = "G" & Gen$ & "i" & Ind$ & ".csr"

On Error GoTo FileDoesntExist:
If FileLen(a$) > 0 Then
ind_count = ind_count + 1
GoTo Check:
End If
FileDoesntExist:
ind_count = ind_count - 1
Exit Sub

End Sub
```

```
Public Sub write_fitness()
Dim chromosome() As String
Dim ch_count As Integer
Dim fnum As Integer
Dim line As String
Dim fname1 As String
Dim fname2 As String

fname1 = App.Path    'Current directory
If Right$(fname1, 1) <> "\" Then fname1 = fname1 & "\"
fname1 = fname1 & "GEN" & "_" & Gen$ & ".chr"

fnum = FreeFile

Open fname1 For Input As fnum
ReDim chromosome(1 To ind_count + 1) As String 'last string is eof
ch_count = 1

Do While Not EOF(fnum)
Line Input #fnum, line
chromosome(ch_count) = line
If ch_count <= ind_count Then
ch_count = ch_count + 1
End If
Loop
Close fnum

fname2 = App.Path    'Current directory
If Right$(fname2, 1) <> "\" Then fname2 = fname2 & "\"
fname2 = fname2 & "GEN" & "_" & Gen$ & ".fit"

fnum = FreeFile
Open fname2 For Output As fnum
Dim j
For j = 1 To ind_count
Print #fnum, chromosome(j) & "," & fitness(j)
Next
Close fnum
End Sub

Public Sub Open_iron_losses()
Dim fnum As Integer    'for assigning file number
Dim one_line As String
Dim counter As Integer 'Counter for array
Dim fname As String

   fname = App.Path    'Current directory
   If Right$(fname, 1) <> "\" Then fname = fname & "\"
```

```
  If i < 10 Then
  Ind$ = "00" + CStr(i)   'Don't use Str(), as it'll create a blank space
  ElseIf i > 9 Then
  Ind$ = "0" + CStr(i)
  ElseIf i > 99 Then
  Ind$ = CStr(i)
  End If

  fname = fname & "LG" & Gen$ & "i" & Ind$ & ".csr" 'Formed the file name

  fnum = FreeFile 'Assign a file number
  Open fname For Input As fnum 'Open .csr file
  counter = 1
  Do While Not EOF(fnum)
     ' Read a line.
     Line Input #fnum, one_line

     If Left$(one_line, 1) <> " " Then
     gArray(counter) = Mid$(one_line, 41)
     counter = counter + 1
     End If

  Loop
 Close fnum 'return assigned number after used
End Sub

Public Sub Wire_config()
Dim fnum As Integer    'for assigning file number
Dim one_line As String
Dim counter As Integer 'Counter for array
Dim fname As String
Dim pos As Integer

  fname = App.Path    'Current directory
  If Right$(fname, 1) <> "\" Then fname = fname & "\"

  If i < 10 Then
  Ind$ = "00" + CStr(i)   'Don't use Str(), as it'll create a blank space
  ElseIf i > 9 Then
  Ind$ = "0" + CStr(i)
  ElseIf i > 99 Then
  Ind$ = CStr(i)
  End If

  fname = fname & "G" & Gen$ & "i" & Ind$ & ".bmd" 'Formed the file name

  fnum = FreeFile 'Assign a file number
  Open fname For Input As fnum 'Open .csr file
  counter = 1
  Do
```

```
    ' Read a line.
    Line Input #fnum, one_line
    pos = InStr(one_line, ":")
    gArray(counter) = Mid$(one_line, pos + 1)
    counter = counter + 1

  Loop While counter < 5

 Close fnum 'return assigned number after used
End Sub

Public Sub Lamination_dimension()
Dim fnum As Integer   'for assigning file number
Dim one_line As String
Dim counter As Integer 'Counter for array
Dim fname As String

  fname = App.Path   'Current directory
  If Right$(fname, 1) <> "\" Then fname = fname & "\"

  If i < 10 Then
  .
  .
  End If

  fname = fname & "G" & Gen$ & "i" & Ind$ & ".csr" 'Formed the file name

  fnum = FreeFile 'Assign a file number
  Open fname For Input As fnum 'Open .csr file
  counter = 1
  Do
    ' Read a line.
    Line Input #fnum, one_line

    If Left$(one_line, 1) <> " " Then
    gArray(counter) = Mid$(one_line, 41)
    counter = counter + 1
    End If
  Loop While counter < 54

 Close fnum 'return assigned number after used
End Sub
```

Appendix F - Implemented code for control module

```
/* ***************************************************
/* * SCRIPT TO CONTROL EXECUTION OF GENETIC ALGORITHM *
/* **** MODULES AND SOLVE FINITE ELEMENT MODEL FOR ****
/* ****************** EACH INDIVIDUAL *****************
/* ***************************************************
/* ********************* PC-OPERA *********************
/* ******************* VERSION 3 **********************
/* ***************************************************
/* *************** CREATED BY K.S. CHAI ***************
/* ******************** July 2002 *********************
/* ***************************************************
/* ***************** WINDOWS NT4 VERSION ***************


/ REQUIRES ...
/ Current.exe
/ fitness.exe
/ GAengine.exe
/ decode.exe
/ ppstrt95.bat
/ ... IN WORKING DIRECTORY TO FUNCTION

/ NEED TO UPDATE GAENGINE.INI MANUALLY THEN RE-RUN

/ INITIALISE ALL VARIABLES
$CONST #I 0
$CONST #J 0
$CONST #K 0
$CONST #R 0
$CONST #S 0
$CONST #T 0
$CONST #GN 0
$CONST #GP1 0
/ INITIALISING #EXIS,#FIN ETC TO TRY TO AVOID ERROR MESSAGES
$CONST #EXIS 2
$CONST #FIN 1
$CONST #DON 1
$CONST #RUN 1
$CONST #ERR 1

/ INITIALISING FILE finished TO VALUE 0
/ NO EFFECT IF NEW RUN, FORCES RESTART BUT WILL STOP
/ AGAIN IF GAENGINE.INI NOT UPDATED

$OS del finished
$OPEN 6 FILE=finished AUTH=WRITE
$WRITE 6 0
$CLOSE 6

/ LOOP FOR EACH GENERATION (000 TO 199)

/ $DO #I 0 1 1
$DO #J 0 9 1
$CONST #I 0
$DO #K 0 9 1
```

```
/ SETTING STRING GEN TO MATCH GENERATION

$STRING G1 %INT(#I)%INT(#J)
$STRING GEN &G1&%INT(#K)

/ (CONSTANT TO BE USED LATER TO DECIDE IF FIRST RUN
/ FOR INITIALISATION)
$CONST #G2 %INT(#I)%INT(#J)
$CONST #GN %INT(#G2)%INT(#K)

/ (DOES NOT RUN INDIVIDUAL LOOP FOR GENERATION 0)
/B1 loop STARTS
$IF %INT(#GN) GE 1

/ DOES NOT RUN LOOP IF finished CONTAINS 1
/ IE #FIN=1

$CONST #FIN 1
$OPEN 6 FILE=finished AUTH=READ
$READ 6 #FIN
$CLOSE 6

/B2 loop STARTS
$IF %INT(#FIN) EQ 0

/ WILL NOT RUN INDIVIDUAL LOOP IF FIRST SOLUTION FILE EXISTS
/ IE ALLOWS GA TO BE RESTARTED AFTER CLEAN STOP BY
/ GAENGINE AND UPDATING GAENGINE.INI FOR MORE GENERATIONS

$STRING GFIL G&GEN&I001.csr

$STRING CA 'cmd /c "if exist'
$STRING CB 'echo 1 >> ran"'
$STRING CC 'cmd /c "if not exist'
$STRING CD 'echo 0 >> ran"'

$CONST #RUN 1

$OS 'del ran'
$STRING XA '&CA& &GFIL&'
$STRING XB '&XA& &CB&'
$OS '&XB&'
$STRING XC '&CC& &GFIL&'
$STRING XD '&XC& &CD&'
$OS '&XD&'

$OPEN 5 ran AUTH=READ
$READ 5 #RUN
$CLOSE 5
$OS 'del ran'
/ NOW #RUN =1 IF GENERATION PREVIOUSLY RUN (FIRST CSR EXISTS) =0 IF
NOT

/B3 loop STARTS
$IF %INT(#RUN) EQ 0

/ LOOP FOR EACH INDIVIDUAL (000 TO 199)

/ $DO #R 0 1 1
$DO #S 0 9 1
$CONST #R 0
```

```
$DO #T 0 9 1

/ SETTING NUMBER FOR INDIVIDUAL

$STRING I1 %INT(#R)%INT(#S)
$STRING IND &I1&%INT(#T)

/ TO STOP OPERA RE-RUNNING SOLVER
/ IF CORRESPONDING CSR FILE ALREADY EXISTS

$STRING FCSR 'G&GEN&I&IND&.csr'
$STRING CA 'cmd /c "if exist'
$STRING CB 'echo 1 >> done"'
$STRING CC 'cmd /c "if not exist'
$STRING CD 'echo 0 >> done"'

$CONST #DON 1

$OS 'del done'
$STRING XA '&CA& &FCSR&'
$STRING XB '&XA& &CB&'
$OS '&XB&'
$STRING XC '&CC& &FCSR&'
$STRING XD '&XC& &CD&'
$OS '&XD&'

$OPEN 5 done AUTH=READ
$READ 5 #DON
$CLOSE 5
/ NOW #DON =1 IF PREVIOUSLY SOLVED (CSR EXISTS) =0 IF NOT

/B4 loop STARTS
$IF %INT(#DON) EQ 0

/ TESTING FOR EXISTENCE OF INDIVIDUALS BEFORE CALLING
/ FILE NAME FORMAT GxxxIyyy.ind

$STRING FILE 'G&GEN&I&IND&.ind'
$STRING CA 'cmd /c "if exist'
$STRING CB 'echo 1 >> exist"'
$STRING CC 'cmd /c "if not exist'
$STRING CD 'echo 0 >> exist"'

$CONST #EXIS 2

$OS 'del exist'
$STRING XA '&CA& &FILE&'
$STRING XB '&XA& &CB&'
$OS '&XB&'
$STRING XC '&CC& &FILE&'
$STRING XD '&XC& &CD&'
$OS '&XD&'

$OPEN 5 exist AUTH=READ
$READ 5 #EXIS
$CLOSE 5

/ NOW #EXIS =1 IF FILE EXISTS, =0 IF NOT EXIST


/ SET CONTROL NAME TO GIVE MATCHING CSR
```

```
/ B5 loop STARTS
$IF %INT(#EXIS) EQ 1

LOAD FILE='VF8_7_Bnsq.dem', APPE=NO
ZOOM
B 0 0

CONTROL OPTI=INIT
CONTROL FILE='G&GEN&I&IND&' OPTI=SET

$COMI FILE='G&GEN&i&IND&.ind' MODE=CONT

/ COMMANDS TO ROTATE AND STORE FOR 0-15DEG
/ IN 1.5DEG STEPS

RUNP rotation
RUNP PA01=* FILE='G&GEN&I&IND&' MIN=0 MAX=15 INCR=1.5 +STOR

/ COMMANDS TO CALL SOLVER AND PP FROM WITHIN DE
/ PREPARE CONTROL SET THEN CALL COMI FILE SPEARATELY
/ TO AVOID DE RUNNING AHEAD OF SOLUTIONS

/ PREPARING COMI FILE
CONTROL OPTI=PREP
$STRING CFIL 'G&GEN&I&IND&.comi'

/ CREATING OPERA2.COMI TO RUN THIS WHEN PP INVOKED
$OS 'del opera2.comi'
$STRING CMD1 'cmd /c "echo $COMI FILE=&CFIL& MODE=CONT >>
opera2.comi"'
$STRING CMD2 'cmd /c "echo END >> opera2.comi"'
$STRING CMD3 'cmd /c "echo YES >> opera2.comi"'
$OS '&CMD1&'
$OS '&CMD2&'
$OS '&CMD3&'

/ INVOKING PP SO WILL WAIT FOR IT TO FINISH BEFORE CONTINUING

$OS 'ppstart.bat'

$OS 'del opera2.comi'

/ REMOVING LARGE FILES WHICH ARE NO LONGER NECESSARY
$OS 'del *.mesh'
$OS 'del *.st'
$OS 'del *.op2'
$OS 'del *.res'
$OS 'del *.old'
$OS 'del *.var'
$OS 'del &CFIL&'
$OS 'del Opera2d_PP_?.lp'
$OS 'del Opera2d_PP_?.log'

/ NEW ADDITION OF IRONLOSS CALCULATION
$PAUSE 2
$OS 'current.exe'
$PAUSE 2

/B6 loop STARTS
$CONST #ERR 1
$OPEN 5 YN_error AUTH=READ
```

```
$READ 5 #ERR
$CLOSE 5

$IF %INT(#ERR) EQ 0
LOAD FILE='VF8_7_Bnsq2.dem', APPE=NO
ZOOM
B 0 0

CONTROL OPTI=INIT
CONTROL FILE='LG&GEN&I&IND&' OPTI=SET

$COMI FILE='G&GEN&I&IND&.ind' MODE=CONT
$COMI FILE='G&GEN&I&IND&.bmd' MODE=CONT
CONTROL OPTI=PREP
$STRING DFIL 'LG&GEN&I&IND&.comi'

/ CREATING OPERA2.COMI TO RUN THIS WHEN PP INVOKED
$OS 'del opera2.comi'
$STRING CMD4 'cmd /c "echo $COMI FILE=&DFIL& MODE=CONT >>
opera2.comi"'
$STRING CMD5 'cmd /c "echo END >> opera2.comi"'
$STRING CMD6 'cmd /c "echo YES >> opera2.comi"'
$OS '&CMD4&'
$OS '&CMD5&'
$OS '&CMD6&'

/ INVOKING PP SO WILL WAIT FOR IT TO FINISH BEFORE CONTINUING

$OS 'ppstart.bat'
$OS 'del opera2.comi'

/ REMOVING LARGE FILES WHICH ARE NO LONGER NECESSARY
$OS 'del *.mesh'
$OS 'del *.st'
$OS 'del *.op2'
$OS 'del *.res'
$OS 'del *.old'
$OS 'del *.var'
$OS 'del &DFIL&'
$OS 'del Opera2d_PP_?.lp'
$OS 'del Opera2d_PP_?.log'
$END IF
/ B6 IF loop ENDS

$END IF
/ B5 IF loop ENDS

$END IF
/ B4 IF loop ENDS

$OS 'del done'
$OS 'del exist'
$OS 'del YN_error'

/ $END DO
$END DO
$END DO

$END IF
/ B3 IF loop ENDS
```

```
$END IF
/ B2 IF loop ENDS

$END IF
/ B1 IF loop ENDS

/ CALL GA MODULES IN SEQUENCE MISSING OUT FITNESS
/ IF IS INITIALISING (IE GEN=000)
/ CHECKING FIRST IF finished IS SET TO 1

$CONST #FIN 1
$OPEN 6 FILE=finished AUTH=READ
$READ 6 #FIN
$CLOSE 6

/ C1 loop STARTS
$IF %INT(#FIN) EQ 0

/ WILL NOT RUN PROGRAMMES IF (#GN+1).CHR FILE EXISTS
/ IE ALLOWS GA TO BE RESTARTED AFTER CLEAN STOP BY
/ GAENGINE AND UPDATING GAENGINE.INI FOR MORE GENERATIONS

$CONST #GP1 #GN+1

/ SETTING NAME FOR NEXT GENERATION .CHR FILE

/ C2 loop STARTS
$IF %INT(#GP1) GE 100
$STRING GFIL GEN_%INT(#GP1).chr
$ELIF %INT(#GP1) LE 9
$STRING GFIL GEN_00%INT(#GP1).chr
$ELIF %INT(#GP1) LE 99
$STRING GFIL GEN_0%INT(#GP1).chr
$END IF
/C2 IF loop ENDS

$STRING CA 'cmd /c "if exist'
$STRING CB 'echo 1 >> ran"'
$STRING CC 'cmd /c "if not exist'
$STRING CD 'echo 0 >> ran"'

$CONST #RUN 1

$OS 'del ran'
$STRING XA '&CA& &GFIL&'
$STRING XB '&XA& &CB&'
$OS '&XB&'
$STRING XC '&CC& &GFIL&'
$STRING XD '&XC& &CD&'
$OS '&XD&'

$OPEN 5 ran AUTH=READ
$READ 5 #RUN
$CLOSE 5
$OS 'del ran'
/ NOW #RUN =1 IF GENERATION PREVIOUSLY RUN (NEXT .CHR EXISTS) =0 IF
NOT

/ C3 loop STARTS
$IF %INT(#RUN) EQ 0
```

```
/ C4 loop STARTS
$IF %INT(#GN) EQ 0
$OS 'GAengine.exe'
$PAUSE 2
$OS 'decode.exe'
$STRING STAT FIRST
$PAUSE 5
$END IF
/ C4 IF loop ENDS

/ C5 loop STARTS
$IF %INT(#GN) GE 1
$PAUSE 2
$OS 'fitness.exe'
$PAUSE 2
$OS 'GAengine.exe'
$CONST #FIN 1
$OPEN 6 FILE=finished AUTH=READ
$READ 6 #FIN
$CLOSE 6

/ C6 loop STARTS
$IF %INT(#FIN) EQ 0
$PAUSE 2
$OS 'decode.exe'
$STRING STAT NORMAL
$PAUSE 5
$ELSE
$STRING STAT FINISHED
$PAUSE 5
$END IF
/ C6 IF loop ENDS

$END IF
/ C5 IF loop ENDS

$END IF
/ C3 IF loop ENDS

$END IF
/ C1 IF loop ENDS

/ $END DO
$END DO
$END DO
```

Appendix G- Implemented code for new fitness function module (used in chapter 7)

```
//Computer Optimization of electrical machine, program written by Kao Siang,Chai..
//GAengine module, last modification on 20/12/2003.

#include<iostream.h>
#include<fstream.h>
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

const double Resolution=0.01;
const int i_rel=20;
const unsigned int MS=42;
void get_pmmf(char*, double[2][MS]);
void get_nmmf(char*, double[2][MS]);

char* ge[]= { };

char file_ptr[31];
char c_file_ptr[31];

int main ()
{
 char str[110];
 double fitness[200]={0};
 double P_data[2][MS];
 double N_data[2][MS];
 double diff[21]= {0};
 double Int_ps[20]={0};
 double Int_ns[20]={0};
 double bench_mark;
 double inc_p,inc_n, dec_p, dec_n;
 double F_torque, S_torque;
 int zero, compare, found, convert_f;

//detect for the latest generation of .csr file.
  int ul=0;
  check:
      strcpy(file_ptr,"PG");
      strcat(file_ptr,ge[ul]);
      strcat(file_ptr,"I");
      strcat(file_ptr,"001");
      strcat(file_ptr,".csr");
  ifstream detect(file_ptr,ios::nocreate);
```

```
 if(!detect)
  {
   ul--;
   goto check1;
  }
 else
  {
   ul++;
   goto check;
  }
  check1:
       detect.close(); /*Detection for latest generation completed.*/

//check for the number of individual from current generation.
 int mo=0;
  find:
     strcpy(file_ptr,"PG");
     strcat(file_ptr,ge[ul]);
     strcat(file_ptr,"I");
     strcat(file_ptr,ge[mo]);
     strcat(file_ptr,".csr");

ifstream search(file_ptr,ios::nocreate);
  if(!search)
   {
   goto find1;
   }
  else
   {
   mo++;
   goto find;
   }
   find1:
       search.close(); //Finished searching for the number of individual.

//Calculating fitness value.
   for(int file_count =0; file_count<mo; file_count++)
      {
       strcpy(file_ptr,"PG");
       strcat(file_ptr,ge[ul]);
       strcat(file_ptr,"I");
       strcat(file_ptr,ge[file_count]);
       strcat(file_ptr,".csr");


                get_pmmf(file_ptr, P_data); //Retrieving info of positive current torque

                strcpy(file_ptr,"NG");
       strcat(file_ptr,ge[ul]);
       strcat(file_ptr,"I");
```

```
strcat(file_ptr,ge[file_count]);
strcat(file_ptr,".csr");


            get_nmmf(file_ptr, N_data); //Retrieving info of negative current torque



//Look for torque dip in first area of interest
            for (int f_t_d=0; f_t_d<21; f_t_d++)
            {

             if((P_data[1][f_t_d]==N_data[1][f_t_d])|| (fabs(P_data[1][f_t_d]-
N_data[1][f_t_d])<= Resolution))
             {
              F_torque=P_data[1][f_t_d];
                   goto no_interpolate1;
             }
            }


//Can't find the first torque dip with existing data, interpolation of data pts required

            for (zero=0; zero<21; zero++)
            {
             diff[zero]=fabs(P_data[1][zero] - N_data[1][zero]);
            }
      bench_mark=diff[0];
      for (compare=1; compare<21; compare++)
            {
             if(diff[compare]<bench_mark)
             {
              bench_mark=diff[compare]; //Record data closes to intersection
             }
            }

            //Extract interpolation data points
            found=0;

            while(diff[found]!= bench_mark)
            {
             found++;
            }

// Linear Inpterpolation begins

           if((P_data[1][found]-N_data[1][found])>0) //intersection happened after
closest difference pt.
            {
```

```
                inc_n=fabs(N_data[1][found]-
N_data[1][found+1])/fabs(N_data[0][found]-N_data[0][found+1]);
      dec_p=fabs(P_data[1][found]-P_data[1][found+1])/fabs(P_data[0][found]-
P_data[0][found+1]);

                for(zero=0; zero<=i_rel; zero++)
                {
                 Int_ps[zero]=P_data[1][found]-((zero*(fabs(P_data[0][found]-
P_data[0][found+1])/i_rel))*dec_p);
                 Int_ns[zero]=N_data[1][found]+((zero*(fabs(N_data[0][found]-
N_data[0][found+1])/i_rel))*inc_n);
                }

                for(zero=0; zero<=i_rel; zero++)
                {
                 if((Int_ps[zero]==Int_ns[zero])|| (fabs(Int_ps[zero]-Int_ns[zero])<=
Resolution))
                      {
                  F_torque=Int_ps[zero];
                       goto no_interpolate1;
                      }
                 else
                      {
                  F_torque=0;
                      }
                }
               }//end of if

              if((P_data[1][found]-N_data[1][found])<0) //intersection happened
before closest difference pt.
               {
                inc_n=fabs(N_data[1][found]-N_data[1][found-
1])/fabs(N_data[0][found]-N_data[0][found-1]);
      dec_p=fabs(P_data[1][found]-P_data[1][found-1])/fabs(P_data[0][found]-
P_data[0][found-1]);

                for(zero=0; zero<=i_rel; zero++)
                {
                 Int_ps[zero]=P_data[1][found-1]-((zero*(fabs(N_data[0][found]-
N_data[0][found-1])/i_rel))*dec_p);
                 Int_ns[zero]=N_data[1][found-1]+((zero*(fabs(N_data[0][found]-
N_data[0][found-1])/i_rel))*inc_n);
                }

                for(zero=0; zero<=i_rel; zero++)
                {
                 if((Int_ps[zero]==Int_ns[zero])|| (fabs(Int_ps[zero]-Int_ns[zero])<=
Resolution))
                      {
                  F_torque=Int_ps[zero];
```

```
                    goto no_interpolate1;
                   }
                 else
                   {
                  F_torque=0;
                   }
                }
               }//end of if

no_interpolate1:

//Look for torque dip in second area of interest
                for (int s_t_d=21; s_t_d<42; s_t_d++)
                {
                 if((P_data[1][s_t_d]==N_data[1][s_t_d])|| (fabs(P_data[1][s_t_d]-
N_data[1][s_t_d])<= Resolution))
                  {
                   S_torque=P_data[1][s_t_d];
                        goto no_interpolate2;
                  }
                }

//Can't find the secong torque dip with existing data, interpolation of data pts required
                for (zero=0; zero<21; zero++)
                {
                 diff[zero]=fabs(N_data[1][zero+21] - P_data[1][zero+21]);
                }

                bench_mark=diff[0];
      for (compare=1; compare<21; compare++)
                {
                 if(diff[compare]<bench_mark)
                  {
                   bench_mark=diff[compare]; //Record data closes to intersection
                  }
                }
                //Extract interpolation data points
                found=0;

                while(diff[found]!= bench_mark)
                {
                 found++;
                }
                convert_f=found+21;

// Linear Inpterpolation begins

               if((N_data[1][convert_f]-P_data[1][convert_f])>0) //intersection happens
after closest difference pt.
                {
```

```
                inc_p=fabs(P_data[1][convert_f]-
P_data[1][convert_f+1])/(P_data[0][convert_f+1]-P_data[0][convert_f]);
        dec_n=fabs(N_data[1][convert_f]-
N_data[1][convert_f+1])/(N_data[0][convert_f+1]-N_data[0][convert_f]);

                for(zero=0; zero<=i_rel; zero++)
                {
                 Int_ps[zero]=P_data[1][convert_f]+((zero*((P_data[0][convert_f+1]-
P_data[0][convert_f])/i_rel))*inc_p);
                 Int_ns[zero]=N_data[1][convert_f]-((zero*((N_data[0][convert_f+1]-
N_data[0][convert_f])/i_rel))*dec_n);
                }

                for(zero=0; zero<=i_rel; zero++)
                {
                 if((Int_ps[zero]==Int_ns[zero])|| (fabs(Int_ps[zero]-Int_ns[zero])<=
Resolution))
                      {
                  S_torque=Int_ps[zero];
                       goto no_interpolate2;
                      }
                 else
                      {
                  S_torque=0;
                      }
                }
               } //end of if

               if((N_data[1][convert_f]-P_data[1][convert_f])<0) //intersection happens
before closest difference pt.
                {
                 inc_p=fabs(P_data[1][convert_f]-P_data[1][convert_f-
1])/(P_data[0][convert_f]-P_data[0][convert_f-1]);
        dec_n=fabs(N_data[1][convert_f]-N_data[1][convert_f-
1])/(N_data[0][convert_f]-N_data[0][convert_f-1]);

                for(zero=0; zero<=i_rel; zero++)
                {
                 Int_ps[zero]=P_data[1][convert_f-1]+((zero*((P_data[0][convert_f]-
P_data[0][convert_f-1])/i_rel))*inc_p);
                 Int_ns[zero]=N_data[1][convert_f-1]-((zero*((N_data[0][convert_f]-
N_data[0][convert_f-1])/i_rel))*dec_n);
                }

                for(zero=0; zero<=i_rel; zero++)
                {
                 if((Int_ps[zero]==Int_ns[zero])|| (fabs(Int_ps[zero]-Int_ns[zero])<=
Resolution))
                      {
                  S_torque=Int_ps[zero];
```

```
                    goto no_interpolate2;
                  }
              else
                   {
                S_torque=0;
                   }
              }
            }//end of if

no_interpolate2:

            fitness[file_count]=F_torque+S_torque;

            if(F_torque<=0||S_torque<=0) //1st penalty fn introduces to penalise on
zero torque.
            {
             fitness[file_count]-=0.3;
            }
            if(fabs(F_torque-S_torque)>=0.3) //2nd penalty fn for unbalance torque
            {
             fitness[file_count]-=0.3;
            }
            if((F_torque<=0||S_torque<=0)&&(fabs(F_torque-S_torque)>=0.3)) //3rd
penalty fn if both the conditions above are violated.
            {
             fitness[file_count]-=0.3;
            }

      }//End of calulating fitness.

 //matching the fitness value to its chromosome.
  strcpy(c_file_ptr,"GEN");
  strcat(c_file_ptr,"_");
  strcat(c_file_ptr,ge[ul]);
  strcat(c_file_ptr,".chr");

 ifstream match(c_file_ptr);

   strcpy(file_ptr,"GEN");
   strcat(file_ptr,"_");
   strcat(file_ptr,ge[ul]);
   strcat(file_ptr,".fit");

 ofstream fout(file_ptr);

  for(int qe=0; qe<mo; qe++)
   {
   match.getline(str,sizeof(str));
   fout<<str<<","<<fitness[qe]<<endl;
   }
```

```
   match.close();
   fout.close();
 cout<<file_ptr<<" is created.";


 return 0;
}

void get_pmmf(char* file_ptr, double P_data[2][MS])
{
   int next=0;
        float value;
        char str[110];


        ifstream capture (file_ptr);
  do{
          capture.getline(str,110);
          if(strstr(str, "Rotor angle "))
          {
                  sscanf(str+40, "%f", &value);
                  P_data[0][next]=value;

          }
                :

  }while(!capture.eof());
  capture.close();
}

void get_nmmf(char* file_ptr, double N_data[2][MS])
{
   int next=0;
       float value;
       char str[110];


       ifstream capture (file_ptr);
  do{
         capture.getline(str,110);
         if(strstr(str, "Rotor angle "))
         {
                 sscanf(str+40, "%f", &value);
                 N_data[0][next]=value;
         }
               :

  }while(!capture.eof());
  capture.close();
}
```