On the Simulation of an All Electric Ship Powertrain Utilizing a Surface Piercing Propeller Via a Modular Main Propulsion Plant Model

Zachary Samuel Zisman

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Ocean Engineering

Nikolaos I. Xiros, Chair
William J. Devenport
Leigh S. McCue-Weil
Karl D. von Ellenrieder

May 23, 2011
Blacksburg, VA

Keywords: Marine, Powertrain, Surface, Piercing, Simulation

On the Simulation of an All Electric Ship Powertrain Utilizing
a Surface Piercing Propeller Via a Modular Main Propulsion Plant Model

Zachary Samuel Zisman

ABSTRACT

A modular simulation model of a marine powertrain consisting of a prime mover, propeller shaft, propulsor, and control system was developed, tested, and used to demonstrate the ability to analyze the marine powertrain numerically. The modularity of the model allows for the user to easily substitute different or more advanced modules, or add additional modules to obtain a greater level of detail or simulate more complex interactions of systems with the marine powertrain. Current and historical trends indicate an interest in all electric ship design, and the use of surface piercing propellers for small craft. Due to the availability of towing tank data from a surface piercing propeller, an all electric prime mover module, surface piercing propeller module, propeller shaft module, and PID control module were coded, integrated, and operated, simulating a complete powertrain. Simulations were conducted using full-scale real-world conditions to demonstrate the model functionality and level of detail. Simulation results provided insight into the vibrational excitation, stability, and control of such a powertrain.

ACKNOWLEDGEMENTS

I would like to acknowledge the United States Navy for funding the education and research required to develop this model and write this thesis. Dr. Nikolaos Xiros provided simplified models for use in comparison test efforts and for that I am thankful. I would also like to thank my advisory committee for their assistance and support throughout the development of my work. Special thanks is owed to Dr. Karl von Ellenrieder for the provision of raw test data for a surface piercing propeller, which was utilized in the development of a key portion of this thesis. Last and certainly not least I would like to thank my friend Foster for taking the time to read and comment on this text. I truly appreciate it.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: Introduction to Marine Propulsion Plants

Marine propulsion systems are designed around the primary requirement to propel the vessel at the desired speed, providing sufficient thrust to support maneuvering, stopping and backing [3]. To achieve this goal, a power plant is developed with four primary components; the prime mover, the propeller shaft(s), the propulsor, and a means of controlling the system. The prime mover must supply the power to rotate the propulsion shaft at the desired speed under any load produced by the propulsor. The propeller shaft must transfer the power from the prime mover to the propulsor and transfer the thrust from the propulsor to the vessel's main thrust bearing. The propulsor must generate the thrust to propel the vessel forward. And of great importance, there must be a control system established to convey the operator's desired actions to the main propulsion plant as a system. To complete their functions, these four primary components generate torque, transmit torque, convert torque to thrust, and maintain the commanded speed of the propulsion plant, respectively.

Design of the marine propulsion plant begins with establishment of the vessel speed requirement. This speed requirement is used in conjunction with the desired hull form and vessel function to determine the resistance of the ship. Determining the resistance of the ship is carried out utilizing standard data series such as the Taylor series or Series 60 initially, and by model testing, computer modeling, or full scale resistance tests in a towing tank at later design stages. The methodologies are well established and fairly common in practice within the naval architecture discipline. Once the resistance of the vessel has been determined, a propulsor can be selected that accommodates the desired speed and resistance parameters. After selection of the propulsor, the torque demand of that propulsor must be established in order to robustly design the main propulsion shaft and select an appropriate prime mover. The current state of the art utilizes

standardized formulae developed by organizations such as the American Bureau of Shipping, torque demand curves for available, tested propulsors, and torque capability curves for available prime movers, as well as detailed, finite element modeling.

It was the focus of this research to establish a methodology to model and simulate the marine propulsion plant in an integrated, modular style that allows simple substitution of various components. Further, the establishment of a method to digitize these components using an all electric drive propulsion plant and a non-conventional propulsor as an example was developed. Of key importance to the modeling effort was the desire to integrate the control system, allowing it to dynamically control the propulsion plant. The primary goals of the research effort were:

1) Develop a computational model of a marine propulsion plant that allows substitution of modular component sub-models.
2) Develop a methodology to translate measured propulsor data for non-conventional propulsors into a digital form that is continuous within the range of service of that propulsor, in order to build a propulsor sub-model.
3) Design and integrate an electric motor sub-model, a detailed propulsion shaft sub-model, and a surface piercing propeller sub-model to demonstrate via simulation the flexibility and utility of the simulation model.

1.1 Types of Marine Propulsion Plants

Marine propulsion plants are typically defined based on the selected prime mover. Plants may utilize multiple prime movers or a single one connected to a single or multiple propulsion shafts and propulsors. A desirable feature of any marine propulsion plant, whether simple or

combined, is a central, unified control system for the plant. When considering plants for large ships, typically types of plants used are gas turbine, steam turbine, nuclear, or diesel plants. For certain applications, such as Naval ships, combined plants are used to enable a ship to have good economy at low cruising speeds and available performance for high speed operations when needed [3]. In the cases of combined plants, typical types include combined diesel or gas turbine (CODOG), combined gas turbine or gas turbine (COGOG), combined gas turbine and gas turbine (COGAG), and combined gas turbine and steam turbine (COGAS). Electric propulsion has been considered since the early 20th century [1].

While not currently in common use, all-electric drive plants offer distinct advantages due to their flexibility in arrangements, ability to eliminate reduction gears in many cases, and wide range of available sizes. Of particular interest is the ability to apply electric drive to small vessels where arrangements and maneuverability are of high concern. The accurate control of speed combined with the ability to output fairly constant torque across a wide range of speeds allows application of non-conventional propulsors such as surface piercing propellers to small craft operating at moderate speeds.

1.2 All Electric Ship Powertrains

An all-electric ship powertrain consists of an electric prime mover, a propulsion shaft, and a propulsor, powered by a power source. If properly designed and applied, all-electric drive eliminates the need for reduction and reversing gears and can yield a propulsion plant with wide speed range. Unfortunately, the all-electric drive powertrain needs to be powered by a ship service power generator or battery. This adds weight back to the vessel where it was saved via elimination of the reduction and reversing gears. However, it provides significant flexibility in

location of the prime mover, power generator, and propulsor. The prime mover and power generator can be located together or on opposite sides of a vessel due to the only connection between them being power leads. Through a survey of industry practices, available data, and limits on the scope of this work, selection of the components for demonstration of the marine propulsion plant model were made. The demonstration plant used in simulation consists of an AC induction motor, propeller shaft, surface piercing propeller, and simple PID control system. The simulated plant is direct-drive, eliminating the need for a reduction gear.

1.2.1 Selection of Prime Mover

Selection of an electric prime mover takes into account a number of factors such as power source, operational need, torque demand, and propulsor speed range. Electric drive machines can provide many advantages over internal combustion prime movers such as:

1) A wide range of available power outputs from small motors to power a lifeboat to motors large enough to power a submarine.
2) A wide range of available speed and torque outputs applicable to large screw-type propellers demanding immense torque and low speed as well as high speed racing boat surface piercing propellers. Electric prime movers can be easily reversed without the need for external reversing gears.
3) Adaptability in location and environmental protection for use in enclosed, underwater pod-type propulsion systems or inboard prime movers.
4) Fast response time to demanded speed changes, in excess of ten times that of an internal combustion engine.
5) Higher efficiency and lower standby losses than internal combustion engines.

6) Greatly reduced maintenance requirements.

For the above mentioned reasons and with the advent of modern power electronics, and advances in magnetic materials, the electric prime mover has torque density approaching that of a hydraulic system [4].

Electric machines are classified into two broad categories by power supply as AC or DC electric machines. For many years, AC machines were considered solely applicable to constant speed applications, however with advances in electric machine control they have become commonly used as variable speed electric machines by utilizing techniques such as variable voltage/variable frequency drive.

DC electric machines can be divided into separately excited shunt machines and series excited machines. In the separately excited machine, the speed is manipulated by varied armature voltage and field current, also known as flux manipulation. In order to increase speed, the flux is reduced, and consequently the maximum torque is reduced. Field weakening control is not typically used as it can cause permanent damage to the commutator and brushes at high speed. Low-speed operation is limited as the flux weakening can only bring the flux to approximately one third of the rated value. Furthermore, during reversal of the separately excited machine, the field current must be reduced to zero which releases control of the load due to loss of torque control during the time field current is zero [4].

Application of the series excited machine was more common until the advent of modern power electronics moved AC machines into more common use. The series excited machine is also known as the universal motor and was often used as a traction motor due to the shape of the speed-torque curve. Operation under high toque and low speed can cause damage to the motor due to excessive field current overheating the rotor. Operation at high speed and low torque can cause erratic commutation making the motor a poor choice for control of high speed loads [4].

AC electric machines can be divided into synchronous and asynchronous (also known as induction) machines. AC machines offer a distinct advantage over DC machines in that they utilize generated power without rectification and can be directly coupled to ship service generators, which produce AC power. The synchronous AC machine will produce constant torque from low speed up to its base speed. The speed is varied by varying stator voltage and when stator voltage reaches its maximum, field weakening can be initiated to speed the machine beyond its base speed at the cost of inversely proportional toque output. Unlike the separately excited DC machine, the synchronous AC machine will not suffer mechanical failure due to field weakening, and can increase in speed until centrifugal forces cause mechanical failure [4]. A synchronous machine is typically more expensive than an induction machine and would typically be selected in the case of the requirement to maintain a precise operational speed such as in positioning equipment that utilizes multiple electric machines functioning in unison such as a CNC machining center, or in demanding marine applications such as cable-laying vessels.

Finally, considering only the AC induction machine, current is induced in the rotor circuit due to magnetic induction from the stator excitation. This produces magnetic motive force with no

physical contact between the stator and rotor. An induction motor produces torque through electromagnetic slip, which can be described as the difference in the rotational speed of the magnetomotive force (MMF) and that of the rotor. At zero slip, the motor produces no torque, but as load is applied to the motor, slip increases, increasing torque to compensate. AC induction machines typically have smaller inertia than their DC counterparts and have little to no maintenance requirements. Furthermore, at low speed or stalled operation, the AC induction machine has the current distributed over the entire rotor, removing localized overheating of the rotor and diminishing damage due to such operation. AC induction machines generally have efficiencies equal to or greater than their DC equivalents. For these reasons, the AC induction machine provides an ideal source of torque for the marine propulsion plant and was selected as the prime mover for application in this research effort.

1.2.2 Selection of Propulsor

Selection of a propulsor is highly dependent upon hull form. Typically, towing tank tests are performed to determine the resistance of the chosen hull at various speeds and conditions varying pitch, yaw, roll, surge, sway, and heave. Once resistance values are obtained, torque demand and thrust curves can be reviewed for various propulsors within available data sets to select an adequate propulsor. Alternatively, detailed analytical modeling or computational fluid dynamical modeling can be used to estimate the thrust and torque output for a given design if it has not been manufactured or made available for open water testing. Modern types of propulsors that are typically considered include conventional screw propellers, ducted or banded propellers, pump jets, and less conventional systems such as surface piercing propellers, magnetohydrodynamic drives, and buoyancy gliding propulsion drives. In 1845, the steamer Great Britain crossed the Atlantic propelled by a screw propeller, from which point on the screw

propeller has reigned as the dominant ship propulsor [5]. Consequently, most available data for propulsors is on screw-type propellers. Non-conventional propulsors can provide advantages for some of the same reasons screw propellers showed dominance to water wheels and oars, namely allowing higher rotational speed and smaller propulsor size.

The surface piercing propeller allows for high rotational speed due to its ability to ventilate the propeller on every revolution, considerably reducing damage due to hydraulic slamming when cavitation voids collapse. The ability to operate at high speed allows a smaller diameter propeller to produce greater thrust than a screw propeller of similar size. Furthermore, the surface piercing propeller allows for much of the propulsor to be physically located above the waterline, reducing pressure on shaft seals, and more importantly eliminating appendage drag from supporting structures in the propeller aperture. Propeller struts are eliminated and in the cases of an articulated surface piercing propeller, the rudder itself may be eliminated further reducing hull appendage resistance. For these reasons, surface piercing propellers have been used extensively in Formula 1 racing vessels and surface traveling vehicles moving in excess of 50 m/s [6]. Due to the complexity of the mathematics involved in modeling a surface piercing propeller's interaction with the air-water interface and the effects of air entrainment in the wake, most data must be obtained experimentally. Experimental data on surface piercing propellers is scarce and often does not include relevant information to the designer on the variation of propulsor orientation [6]. Recently, Dr. Karl von Ellenrieder ran a series of towing tank experiments on a small surface piercing propeller at various pitch, yaw, and immersion depths, producing a very good set of raw data and provided access to that data for implementation in this research effort. Due to the inherent advantages of the surface piercing propeller and the

availability of a useful data set, it was selected as the propulsor for use in the model developed under this effort.

1.2.3 Electric Machine Variable Speed Control

Variable speed control of the AC induction machine can be carried out in many ways. One method of control involves the installation of slip rings to allow potentiometers to be installed outboard of the motor, controlling the rotor winding resistances. The installation of slip rings and potentiometers allows for control of the motor speed by effectively re-winding the rotor without taking the machine apart. While this provided a means of adjusting speed, it has the same problem that the DC machines have at high speeds. There is a potential for mechanical failure of the slip rings in the same manner brushes and commutators can fail. It also reduces efficiency of the motor.

Alternatively, terminal or stator voltage can be manipulated to control motor speed. Unfortunately with this simple method, speed is controlled by a reduction in the available torque of the induction machine with decreasing terminal voltage. Efficiency drops as slip increases and speed variation is small with changes in voltage [4]. Terminal voltage range must be large to allow useful motor speed range following this control method.

A commonly used method of speed variation in the induction machine is variation of the frequency of the supply current. Due to the wide availability of power electronics, this is a feasible method of speed control and is the current state-of-the-art. As input frequency is increased, via pulse width modulation, the motor speed is increased by increasing the rotational speed of the magnetomotive force. To improve efficiency of this type of speed control, the

addition of variable voltage control is implemented below the rated voltage. In the operation of variable voltage, variable frequency control, air gap flux is maintained constant by balancing the terminal voltage and current frequency up to the rated voltage, at which point voltage is held constant and frequency is increased as desired to run the induction machine at increased speed [4].

With the ability to vary the AC induction machine, a control system is needed to set and maintain the desired speed for application to the marine propulsion plant. Some systems commonly used include state feedback, output feedback, open loop, and speed-sensorless control. Control systems implemented for the control of AC induction machines are vast and their optimization is beyond the scope of this work. Research efforts for this work are focused on the marine propulsion plant and the reader is referred to [15] and [4] for in depth treatment on the subject of electric machine control. For these reasons, a simple PID with lookup table control scheme was selected for development and application in this work.

CHAPTER 2: Development of a Surface Piercing Propeller Model Module

While having been in use for decades, surface piercing propellers have not been modeled for use in simulations extensively due to the complexity of the physics involved in their operation. Available data on these propellers is scarce, which makes reliable performance predictions of propulsion plants using them difficult. Collection of data via towing tank measurements is useful, however its application to simulation models is not trivial due to the form of recorded data. For propulsion plant modeling purposes, the physics of how the propeller generates torque is not of interest, but the torque demand at any condition is. For this reason, the present surface piercing propeller modeling efforts were focused on conversion of available data to a black box model which outputs torque demand of the propeller for a given set of input parameters.

2.1 Available Forms of Non-Conventional Propulsor Data

In order to construct a numerical model of a propulsor, one must either analytically, numerically, or experimentally obtain data for the propulsor and implement that data as needed in a model. Conventional screw-type propeller data is available in the form of thrust and torque curves, data series, computational fluid dynamical models, analytical approximations, and raw test data. Non-conventional propulsors are less documented. Torque and thrust curves are typically derived from experiment and the number of non-conventional propulsors tested pales in comparison to that of conventional propellers. This presents a problem to the modeling effort as data is difficult to obtain. Design of experiments, open water testing, and the subsequent post-processing of the data is time consuming and expensive. Analytical models typically require large numbers of assumptions to be made tractable problems and do not provide the fidelity of the experimental data or finite element models. Finite element models utilizing computational fluid dynamics take enormous effort to reproduce the number of conditions tested in a typical

towing tank experiment, and often are simplified to reduce processing effort. To implement a computational fluid dynamical module in a main propulsion plant model that is simply concerned with the torque load under various dynamic conditions can be a wasteful practice. Excessive time would be required at each time step of the overall model simulation in order to obtain the numerical solution for the propulsor at that time step. Furthermore, when considering a finite element method solution of the surface piercing propeller, one must consider free surface effects, multiple fluids (air and water), entrainment of water on the propeller surface impacting the inertia of the propeller, entrainment of air in the wake affecting wake properties, and wake development; many factors not typically encountered in propulsor modeling. For the main propulsion plant model of interest in this research effort, a rapidly converging input/output type model is sufficient and appropriate.

The data obtained from Dr. Karl von Ellenrieder was in the form of time series, voltage signal data for various propeller orientations. Details on the experimental setup can be retrieved from the referenced report [6]. The raw data contained a wealth of information, the desired pieces of which were transformed into a black box model of the specific surface piercing propeller tested. The result of the data reduction effort provided a model that accurately interpolates the data in an intelligent manner.

2.2 Reduction of Tow Tank Data

Towing tank data was obtained for a Dewald Propellers, Inc. surface piercing propeller characterized by an overall diameter of 9.7 inches with a pitch to diameter ratio of 1.9. The propeller is a left-hand cleaver, stainless steel, four-bladed surface piercing propeller. Having a fixed pitch of the propeller blades, the term pitch as used in this text shall refer to the shaft

inclination angle, or the pitch of the propeller shaft. The received data consisted of 95 combinations of the following test matrix [6]:

Immersion Ratio(%)	Pitch Angle (degrees) [shaft inclination angle]	Yaw Angle (degrees)	Advance Ratio
33	0, 7.5, 15	0, 15, 30	0.8 – 1.9
50	0	0	0.8 – 1.9

Table 2-1: Surface Piercing Propeller test matrix (Adapted from [6])

The first step of the data reduction effort was to review the data. Included with the raw data was a series of Matlab scripts written to reduce the data for a different effort. All scripts used in the formulation of the models presented may be reviewed in Appendix A: Source Code. The following scripts were modified to perform the functions listed below:

1. AnalyzeDataZZ.m: Reads in the text data files, converts the signals to Matlab arrays and saves each test run as a Matlab data file.
2. Plot_Analysis.m: Generates and saves .jpeg image files of each data run from the Matlab data files produced by AnalyzeDataZZ.m. Plots were used to manually generate a list of valid data ranges.
3. Volt_Conversion.m: Reads in Matlab raw data files and converts them to data structures in new files for use in the steady state data extraction script PlotVDR.m.
4. PlotVDR.m: Takes as input ValidDataRanges_reduced_ZZ.xls listing the stable, non-transient sections of the data and converts the data structures to new reduced data structures, saving them in a reduced data folder.

Upon execution of these scripts, a set of reduced data files was generated and saved to a directory for reference later when developing a black box model. In the above manipulations of the data set, the only changes made were the conversion of voltage signals to data values and the removal of the start-up and shut-down transient portions of the data. No scaling was performed to the data set and no stray data points were removed. To model operation within common advance ratio values, shaft speeds between 800 and 2400 rpm were selected. While most surface piercing propellers are operated closer to 10,000 rpm for racing applications, the research efforts presented here are targeted toward an alternative application of this type of propeller on a lower speed, larger vessel such as a fishing boat or patrol boat.

2.3 Development of a Black Box Torque Demand Model Module

As an initial effort to develop a simple input-output black box model, data maps appeared to be a promising route. A script generating a series of three dimensional maps was coded, implementing linear interpolation between points on the mapped surfaces. The mapping layout was based on the test matrix such that individual maps of torque versus advance ratio and pitch were established for each of the three tested yaw conditions. The script was expanded to an early effort toward a black box model, which included a lookup function that would prompt for inputs and return the output interpolated between the mapped data points. While the mapping effort was functional and provided a way to quickly access the data collected, the interpolation scheme implemented was poor and the output was limited to the precision of the maps generated. To execute the effort of creating a black box model more elegantly, neural networks were investigated. Great success was achieved with implementation of a neural network black box model, and this path was selected and refined.

2.3.1 Limitations of Data Maps

While functional, the data maps initially generated provided fairly rough precision and poor interpolation. With increased effort to improve precision it was obvious that many assumptions would have to be made about selecting interpolation functions and with no physical basis to select such functions, the mapping effort was aborted in favor of a neural network that would consider trends in all the data at once. As can be seen in the data maps, a good representation of the data can be extracted from the maps, but drastic assumptions would have to be made to obtain values not explicitly on the maps. It is easily seen that using linear interpolation, one simply increases the density of points between those generated by real data without changing the map. In Figure 2-1 below, only the real data points are plotted, while in Figure 2-2 an increased mesh density reveals a nicer looking map that provides no additional information.

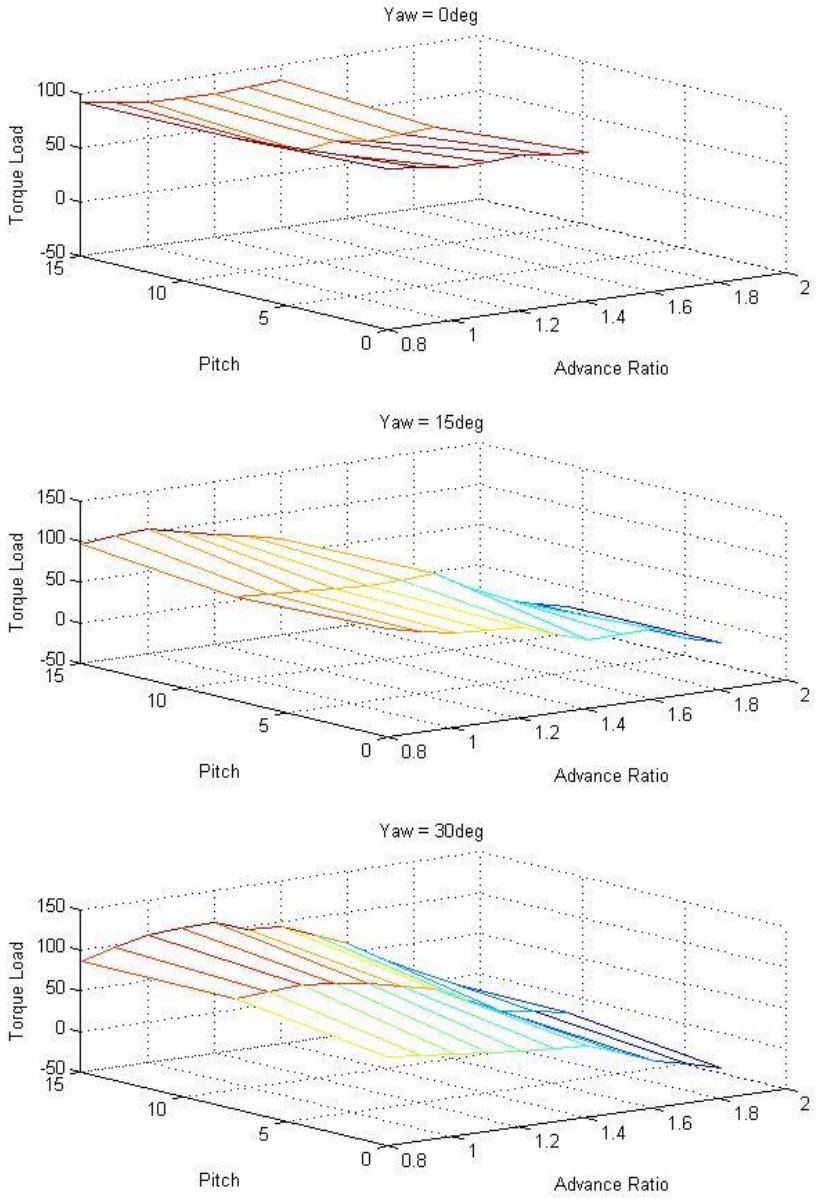

Figure 2-1: 3D Plots of recorded data corrected with a torque multiplication factor of 10

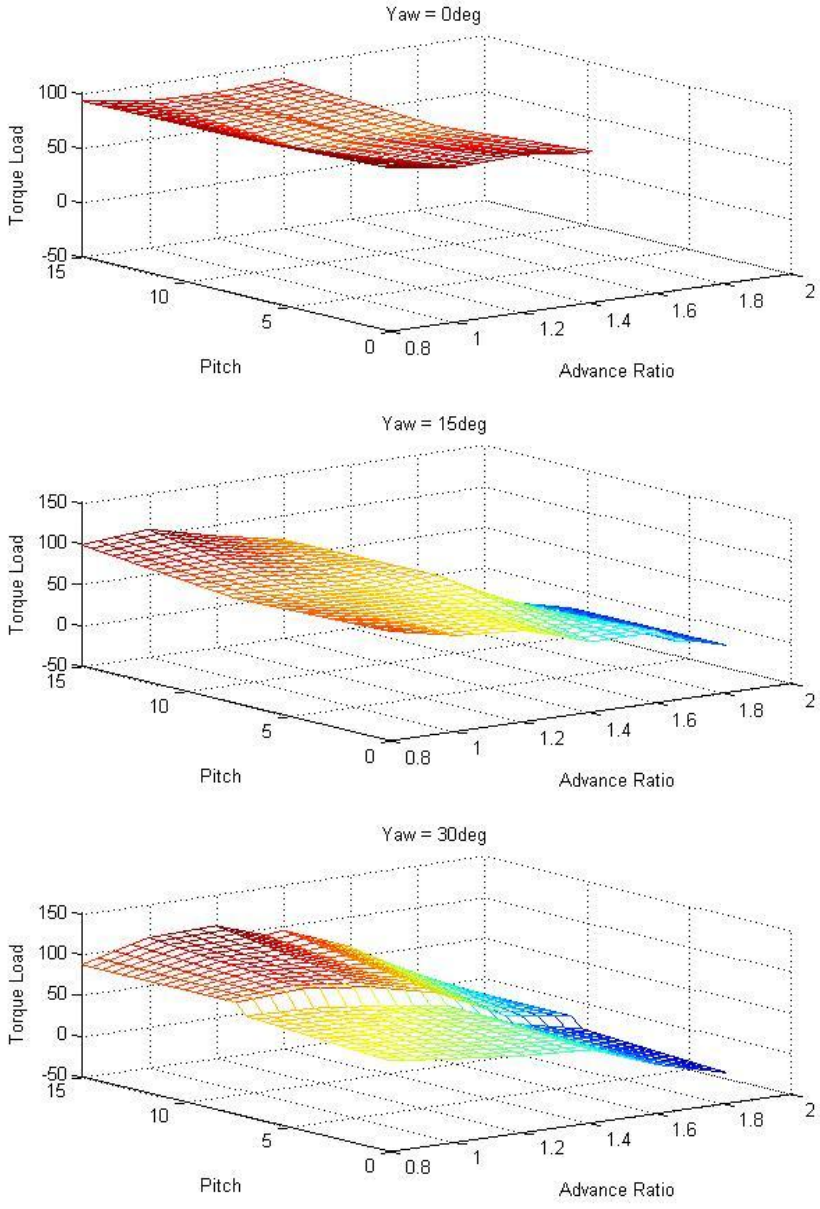

Figure 2-2: 3D plots with increased mesh density showing interpolated data points

In both figures 2-1 and 2-2, a torque correction factor of 10 was applied to the data due to a recognized instrumentation error in the torque measurement [6]. Due to the number of assumptions required to develop and interpolation script, the data map method of generating a

black box model was aborted in favor of training an artificial neural network to represent the surface piercing propeller data.

2.3.2 Neural Network Model

The concept of the artificial neural network dates back to 1948 when Alan Turing wrote a report titled 'Intelligent Machinery' in which he challenged the "unwillingness to admit that mankind can have any rivals in intellectual power" [7]. Early in this work, Turing responds to the rejection of the existence of machine intelligence because machines must not make mistakes. He responds by arguing "…this is not a requirement for intelligence" [7]. Modern artificial neural networks sometimes make mistakes when providing responses to inquiry, much like the human brain. However, also like the human brain, with increased training, the likelihood of these mistakes is reduced. In his work, Turing describes the concept of A-type and B-type unorganized machines which resemble human neurons, forming the basis of modern artificial neural networks. In describing the development of a "thinking machine" Turing suggests "In order that the machine should have a chance of finding things out for itself it should be allowed to roam the countryside"[7]. Turing goes on to explain how a machine given discipline and initiative can think like a human being. Modern neural network algorithms such as those built into Matlab's Neural Network Toolbox have the initiative and the discipline through what Turing describes as interference. The interference he describes is outside stimulation, which can be provided in the form of training data. Discussion of the theory of connectionism and the basis of artificial neural networks is outside the scope of this research effort and the interested reader is referred to [7], [8], and [9].

The concept of the artificial neural network offers great promise confronting the present problem of mapping the torque demand of the surface piercing propeller to multiple input parameters. The desired propulsor black box requires as input the vessel speed, the shaft speed, the pitch, yaw and immersion of the propeller, and provides as output the instantaneous torque demand. With the reduced data presented in section 2.3.1 the required training data fulfilling Turing's discipline requirement was available. Selection of a device that provides the initiative led to using Matlab's Neural Network Toolbox. This toolbox has pre-defined, modifiable algorithms for the development of artificial neural networks. While these canned algorithms exist and are fairly simple to implement, a quality network that accurately represents the problem at hand requires tuning the network parameters. A key difference between Turing's early B-type machines which exist in a state of either passing or inhibiting information and modern implemented artificial neural networks is the concept of weighting this connection. Neurons can be weighted to smooth passing information through them and thus more accurately converge to a solution.

To implement an artificial neural network as a black box using Matlab, the reduced data was fed into a customized network built up using a feed-forward network as the basis. This type of network is typical in the processing of nonlinear engineering systems. Feed-forward networks differ from recurrent network in that they do not contain feedback connection and allow strictly forward flow of information. Various types of networks were experimented with and feed-forward networks seemed to provide the best network in that they trained accurately, producing close to 95% accurate networks when compared to the provided training data. According to research efforts conducted by Kolmogorov in 1957, "any scalar map…of ν variables can be

represented by a three-layered, feed-forward neural net with a (2ν+1)-neurons hidden layer", where ν is the number of variables in the map [8]. Using Kolmogorov's architecture, a standard (Matlab pre-configured) feed-forward, three-layer, fully connected neural network was coded for 4 inputs utilizing a 9 neuron hidden layer. To avoid the need to normalize the targets, a purelin (pure linear) output layer transfer function was used. The program was modified to operate with normalized inputs and outputs and no improvement was obtained so input and output data were not modified for use in the final network. The transfer functions for the first two layers were selected as logsig, (logistic-sigmoid function), which is the usual transfer function used for scalar maps according to Xiros [8]. The logsig function used in Matlab is given by equation 2-1 [8]:

$$\Phi(\zeta) = e^{\zeta} / (1 + e^{\zeta}) \qquad (2\text{-}1)$$

Output layer size was set to 1 neuron as only a single output value is desired, torque demand of the surface piercing propeller. The input layer size was based on the thought that the hidden layer (decided by Kolmogorov's architecture) should be approximately 10% of the size of the input layer, yielding 90 neurons as the starting point [10]. Slight changes were made to the input layer size during experimentation with the network generation script with the best results obtained for an input layer size of 130 neurons. Other parameters modified to tune the network included setting the number of inputs to one and applying a vectorized input with 4 elements (advance ratio, pitch, yaw, and immersion). The input connections were established to provide the input vector to both the first and the hidden layer, while the output connection took output from only the output, linear layer. Biases, input weights, and layer weights were all set to self-learn which drastically improved performance as little is known about the system to accurately

specify these values. The training function was selected as the traincgp (conjugate gradient back-propagation with Polak-Ribiere updates), which is based on the application of the calculus backpropogation theory by David Rumelhart and David Parker in the 1980's. This method compares the obtained results [11].

With the network architecture established and coded and the towing tank test data reduced, the artificial neural network describing the operation of a surface piercing propeller could be executed. The final step of actually producing the network is quite interesting in that every time the network generation script (NeuralNetBuilderV5.m) is executed, a different network is created and will have different accuracy. For this reason, many versions of the network were executed and the most promising networks were saved. The basis of determining a promising network was the comparison of actual test data (training data) to the output of the network (simulated data). Matlab has a function called postreg which post processes the neural network response using linear regression. This is a good measure of the accuracy of the network and was used extensively. However, since the network can take many paths to reach a comparable solution, just like a human brain, some networks can show promising results without providing realistic results compared to manually reduced model test data. A very good example of this is seen when comparing the very best network from a linear regression standpoint to the second best. Figure 2-3 shows the best network regression plot, while figure 2-4 shows the second best network obtained. In these two plots, Y represents the simulated output, while T represents the training data targets. Plots of the actual versus simulated target values are shown in figures 2-5 and 2-6 for the network representing the best and second best linear regression analyses, respectively. These two networks will from here on be referred to as neural network 1 and neural network 2, respectively.

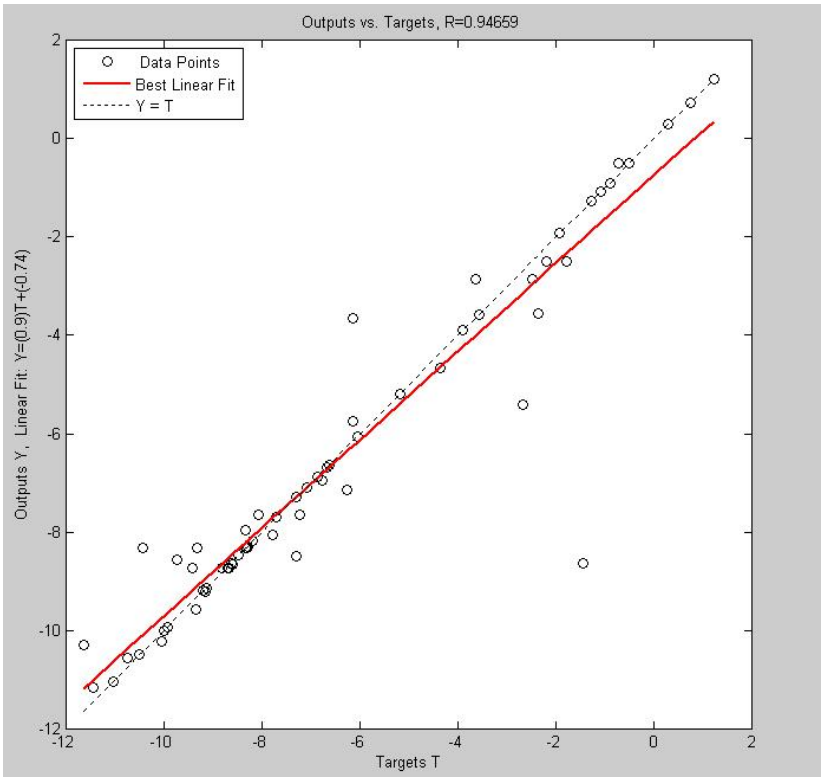

Figure 2-3: Linear regression plot representing neural network 1

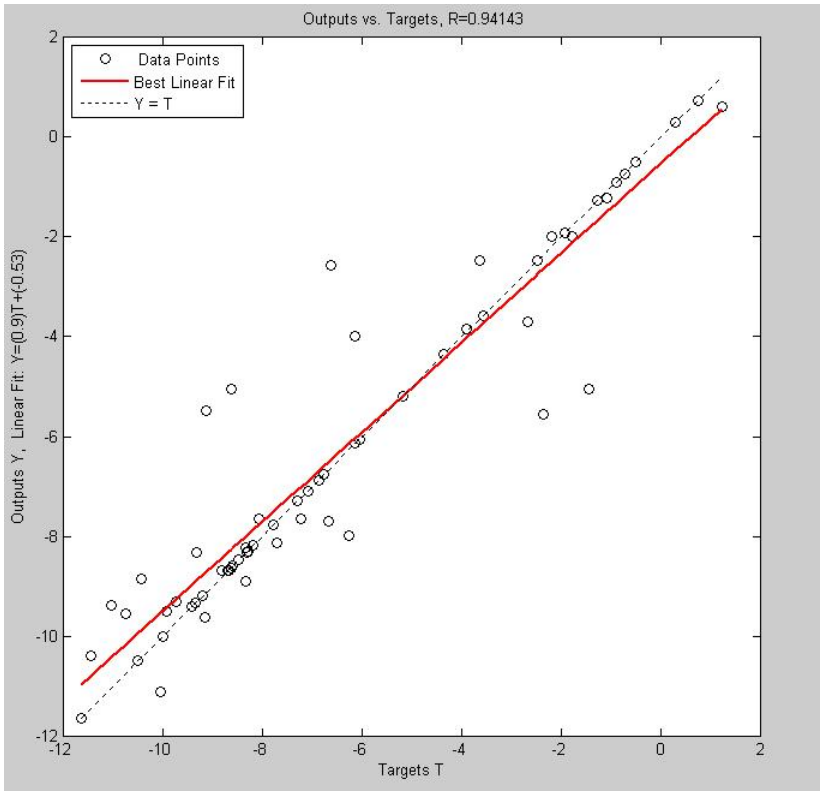

Figure 2-4: Linear regression plot representing neural network 2

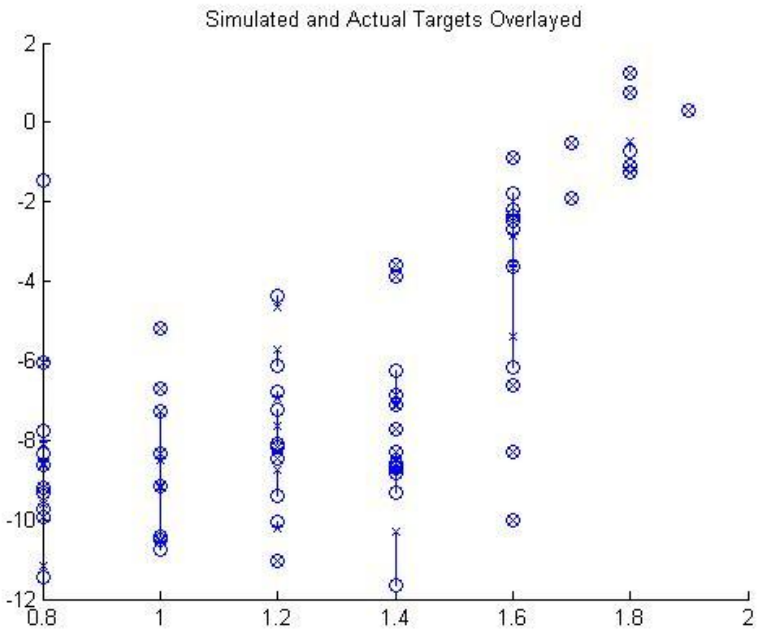

Figure 2-5: Simulated versus actual targets of neural network 1

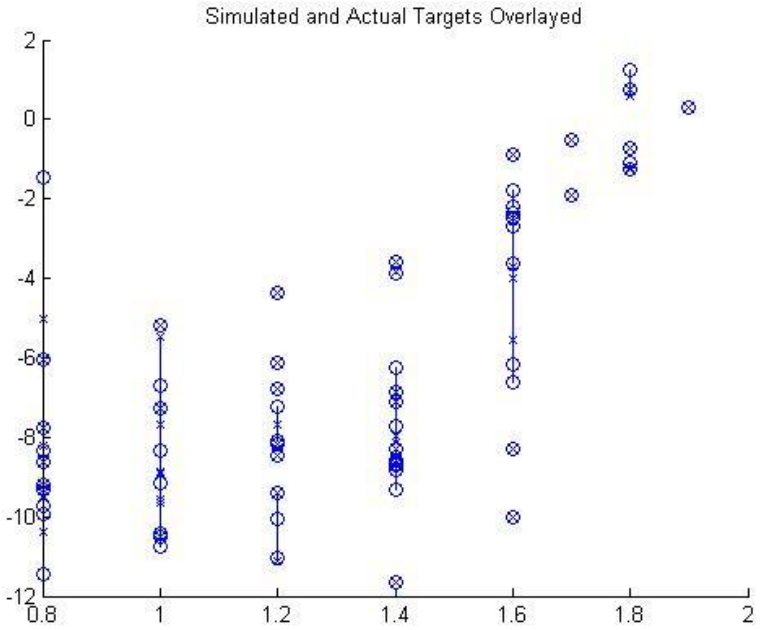

Figure 2-6: Simulated versus actual targets of neural network 2

When considering the linear regression plots presented in figures 2-3, 2-4, and 2-7, it should be noted that a perfect fit will have the equation $Y = 1.0 \cdot T$. The plot shows the fit between the

simulated data and the target data, with a perfect, ideal fit represented by the solid line and the dotted line being collinear.

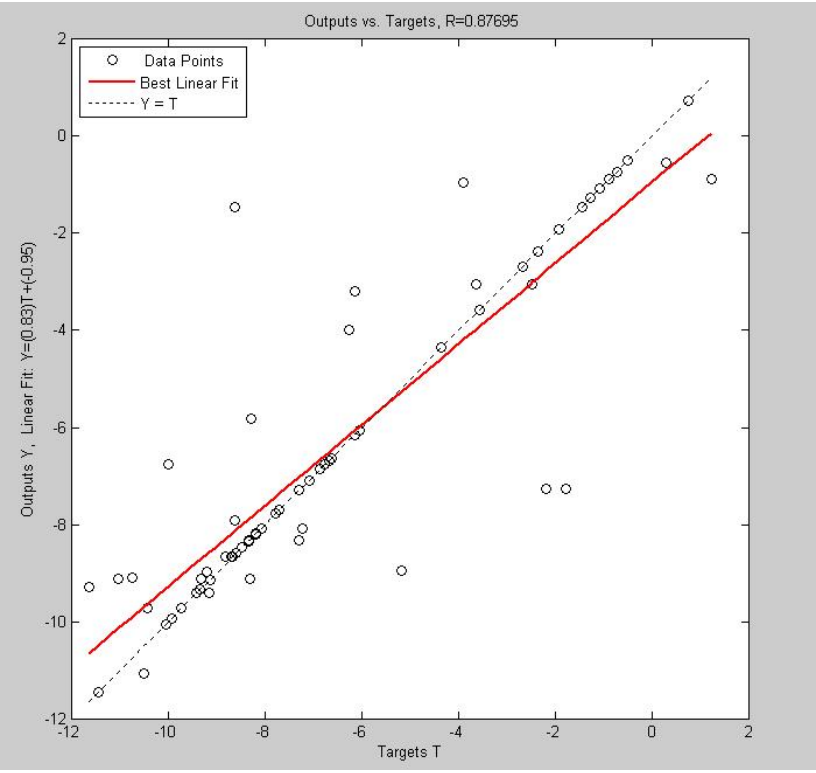

Figure 2-7: A network generated with identical configuration parameters showing the ability of the machine to self-organize in different ways every time it is trained

An example of a poor network generated using the same network configuration parameters (NeuralNetBuilderV5.m) can be seen in figure 2-7. This clearly shows the concept that the network (or self-organizing machine) will organize itself differently every time it is trained and therefore the network can represent many different physical processes, some of which will not be applicable. While the linear regression plots are useful for weeding out poor networks, having good linear regression results cannot be the only factor in deciding on which network to use. To establish a better method of selecting the appropriate network, the best networks were analyzed further to determine which ones model the surface piercing propeller best. To that end, comparison to existing data was turned to by considering the shape of speed torque curves.

2.3.3 Generation of Speed Torque Curves

While not readily available for surface piercing propellers, speed torque curves for conventional propellers can provide some insight into the quality of a neural network. It is expected in practice that during forward operation of the vessel at low propeller speed, the motion of the water across the propeller will cause the propeller to drive the propulsion plant or generate torque. This is realized by the efforts of the fluid speed across the propeller attempting to speed the propeller. The propeller, being dragged, will produce a driving torque on the propeller shaft. This type of operation was described during propeller reversal by Commander S. M. Robinson and illustrated with Robinson curves as shown in figure 2-10 [1]. It is also expected that in normal operation, the propeller will demand torque and for the most part the torque load due to the propeller will increase with speed until fluid dynamical phenomena cause a reduction in torque load at very high speeds. Investigation of these phenomena, such as cavitation and entrainment of air, which cause ventilated cavities and the subsequent reduction in torque demand, is outside the scope of this work. For details concerning wake cavity shape and its effects on efficiency of the propeller, the reader is referred to [5] and [6]. Two basic forms of speed torque curves were derived from the best performing neural networks. These two shapes are presented below in figures 2-8 and 2-9. Generation of the speed torque curves is carried out with the Matlab script SPPTorqueDemandGenV3.m. As can be seen, neural network 1 indicates tremendous torque generation at low propeller rpm with forward vessel speed. Conversely, neural network 2 presents a set of speed torque curves with some torque generation at low rpm, which diminishes with reduced vessel speed. Figure 2-9 can be compared with Robinson curves for conventional screw-type propellers as seen in 2-10. As can be seen clearly in the comparison, at low speed, the propeller can be driven by the forward speed of the vessel, thus generating torque output from the propeller. Neural network 2 demonstrates this ability and even

shows the correct curve shape with a negative torque demand at low speeds that rapidly increases to the loaded condition upon increasing the propeller speed. Based on this matching, neural network 1 was abandoned in favor of neural network 2.

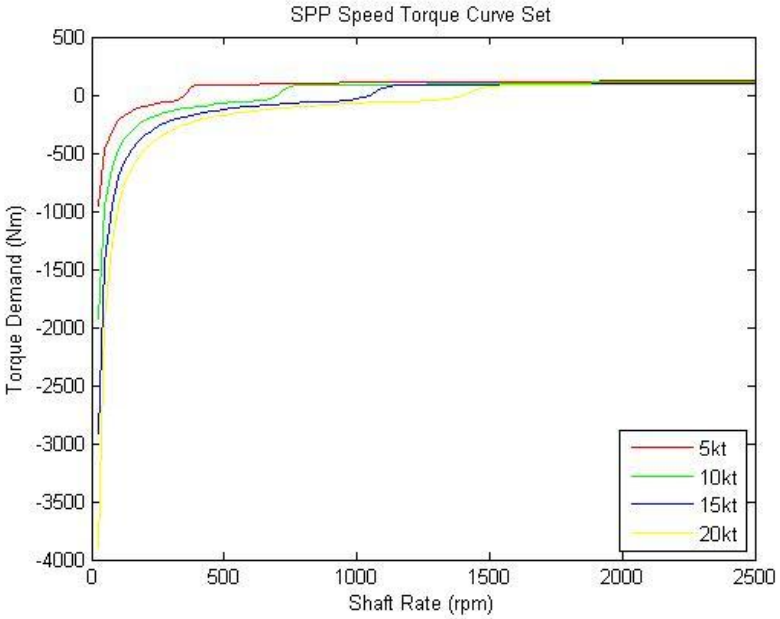

Figure 2-8: Speed torque curve of SPP based on neural network 1

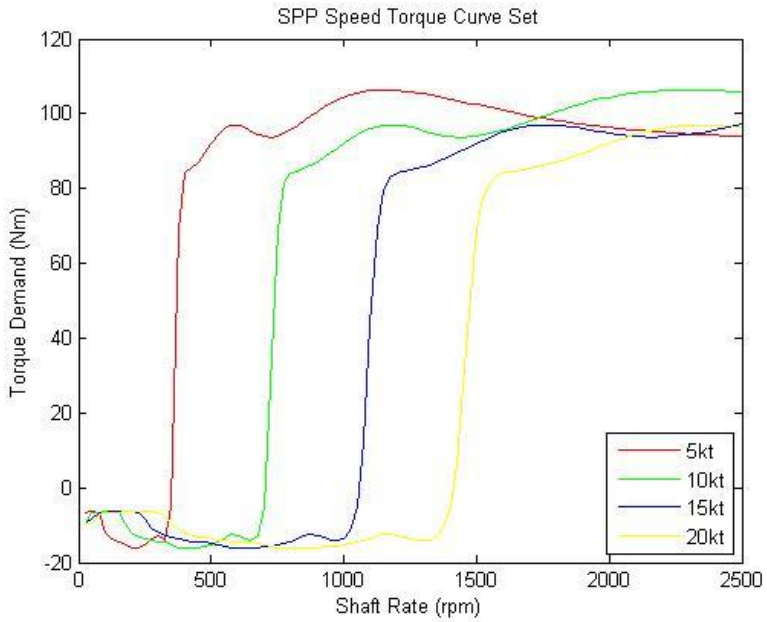

Figure 2-9: Speed torque curve of SPP based on neural network 2

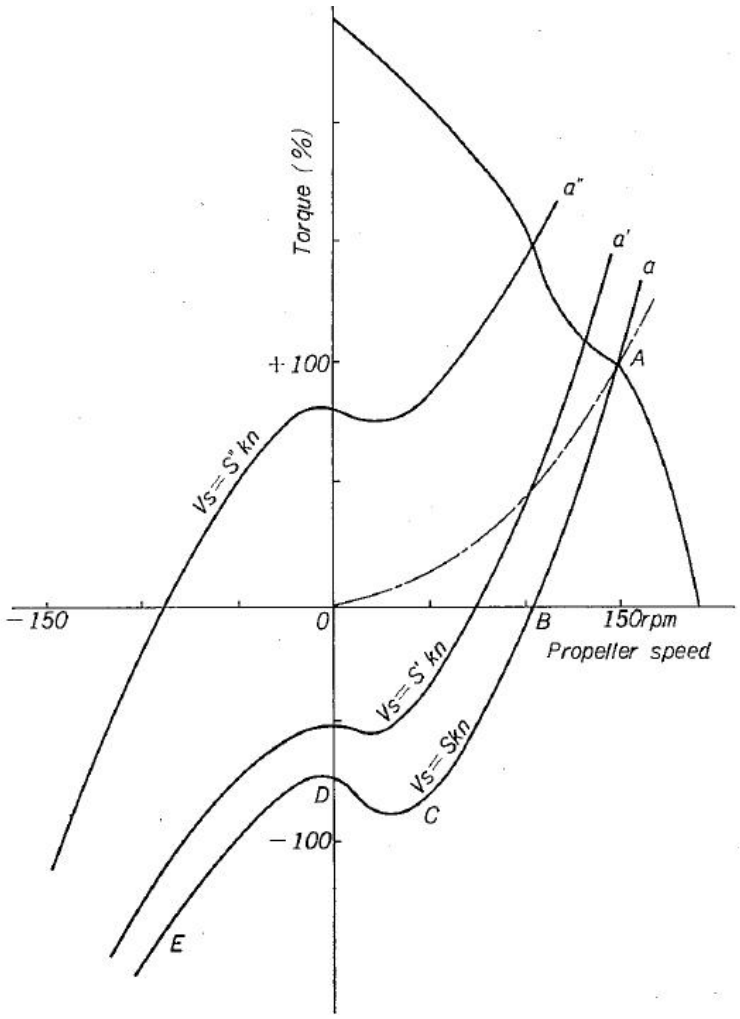

Figure 2-10: Robinson curves demonstrating torque generation [Public Domain]

2.3.4 Completion of the Surface Piercing Propeller Model Module

With the establishment of a satisfactory propeller model that does not consider the physics of the process of propeller torque demand, but yields accurate results, finalization of the model module was fairly simple. Matlab and Simulink were selected as the tools for modeling and simulation of the marine propulsion plant. It has been established that the surface piercing propeller exhibits torque demand dependency on vessel speed, rotational speed, propeller pitch, yaw, and immersion. These inputs were built into the model module with the only output of the module being the torque demand of the surface piercing propeller. Due to the primary desire to build a modular propulsion plant model, Simulink was selected as the environment in which to build

each module because it does an excellent job integrating multiple models and automatically varying the integration time steps to ensure that all models will be correctly stepped in time. The neural network surface piercing propeller model is a very fast module in terms of computation time as the neural network was tuned specifically for the data set used, and it acts as a very intelligent lookup table capable of using intuition that it developed during training. The Simulink model of the surface piercing propeller is shown below in figure 2-11.

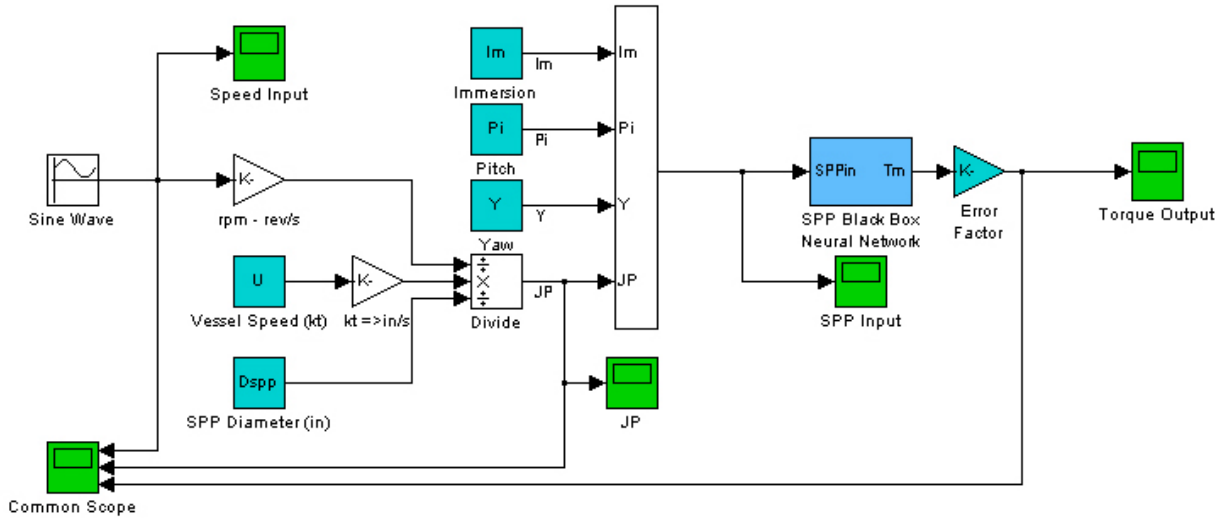

Figure 2-11: Surface piercing propeller module

To demonstrate the typical output of the module, a sinusoidal input shaft speed command was applied for two full periods applied in 100 seconds of simulation time (slow variation) between 700 rpm and 2500 rpm. The immersion was set at 33% (the typical operational value for this propeller), and the pitch and yaw were set at 0 degrees. The vessel speed was held constant at 10 knots to demonstrate torque unloading and generation as propeller speed is dropped. The output of this simple test is presented below in figure 2-12. In the figure, the top scale displays the instantaneous shaft speed in rpm, the middle scale displays the advance ratio, and the bottom

scale the torque demand of the propeller in Nm. As can be seen from the plots, the output is continuous and the propeller behaves as would be expected.

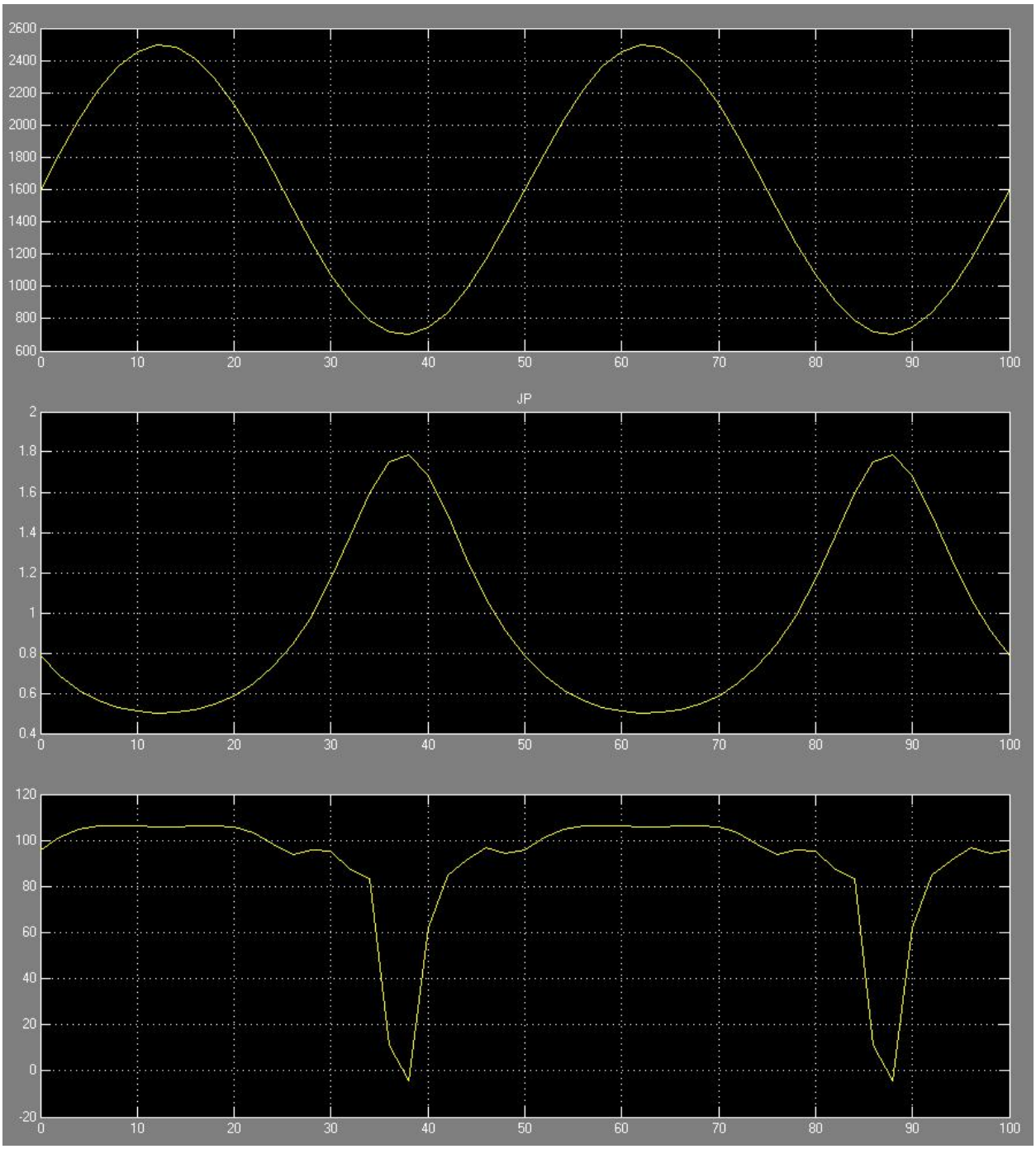

Figure 2-12: Sinusoidal speed variation of surface piercing propeller module

The neural network developed in the preceding sections is shown in figure 2-13. This figure visually represents the neural network parameters as finalized during training. The top of the figure shows the overall network and the three smaller networks beneath are the individual layers including the transfer functions used for each.

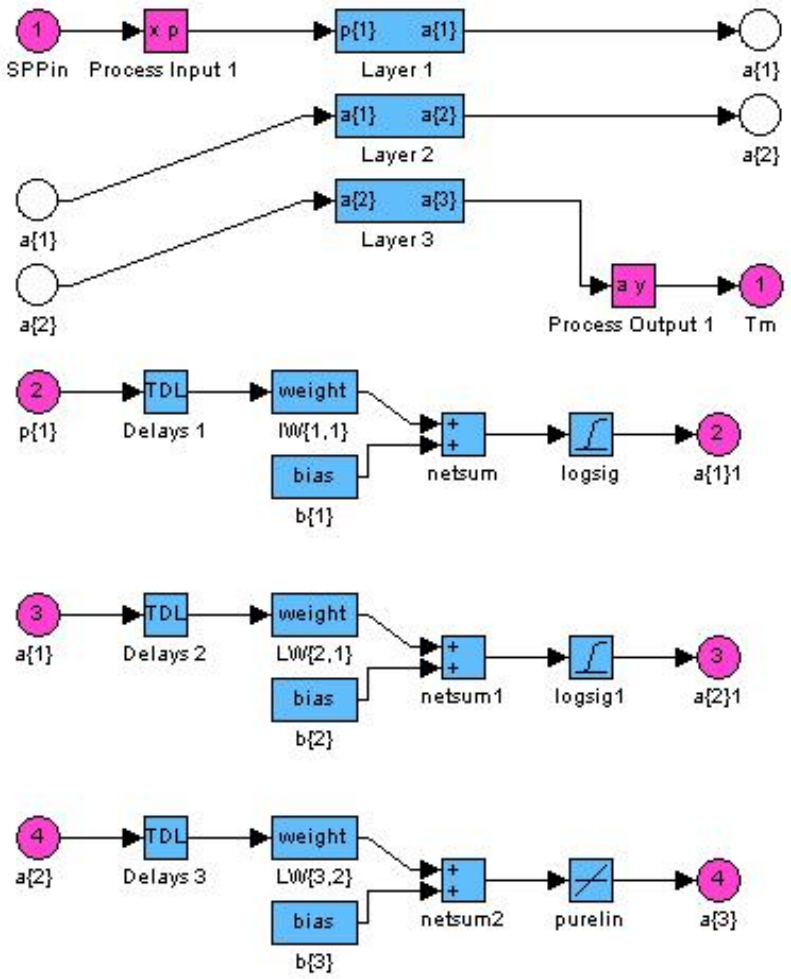

Figure 2-13: Neural network block diagram showing individual layers and transfer functions

CHAPTER 3: Development of a Shafting Model Module

To transmit rotation from the prime mover to the propeller is the purpose of the propeller shaft. In order to drive the propeller at the speed the prime mover imposes, the propeller shaft must transmit the torque demanded by the propeller, plus the torque lost along the shaft due to twisting of the shaft and frictional losses back to the prime mover. In other words, there must be a torque balance on the shaft itself between the prime mover output torque, the propulsor torque demand, and these losses. To derive a shafting model capable of realizing this level of detail requires a vibrational-type analysis to be conducted. Furthermore, if the shaft is anything but constant cross section, it must be broken up into segments of different cross section. The level of detail described by a finite element model is not of interest to the propulsion plant model, however the model does take into account the varying cross sections of the shaft introduced by hollow shaft elements, connecting flanges, and the propeller itself.

3.1 Desired Inputs and Outputs of Propulsion Shaft Module

To begin the task of modeling the shaft as a mechanical system, it is helpful to first define the input and output parameters for the model. To integrate between a propeller model that outputs torque for a given speed (and other non-shaft driven parameters) and a prime mover model that outputs shaft speed for a given torque load, the shafting model itself should take as its input shaft speed from the prime mover and torque demand from the propeller. It must transmit these quantities to the opposite ends taking into account shaft twist, vibration, and friction due to bearing loads. To be modeled realistically, the model should also accept as parameters the shaft material properties, an estimate of frictional losses, and the physical geometry of the shaft. With the inputs and outputs solidified the mechanics of the shaft can be derived.

3.2 Mechanics of a Rotating Shaft

The shaft is modeled a series of inertias (masses) and springs with frictional losses on the mass elements. The forcing function for the system is based upon a time-dependent torque load on the propeller end and a time-dependent shaft speed at the prime mover end of the shaft. The mechanical model can be represented by the one-dimensional system of masses and springs rotating on bearing surfaces as depicted below in figure 3-1. In the figure, the green springs represent the torsional springs between inertias that allow for shaft rotational flexure, the red arrows at the left and right ends represent the input speed and torque, respectively, and the orange lines coming out of the masses represent the frictional losses.

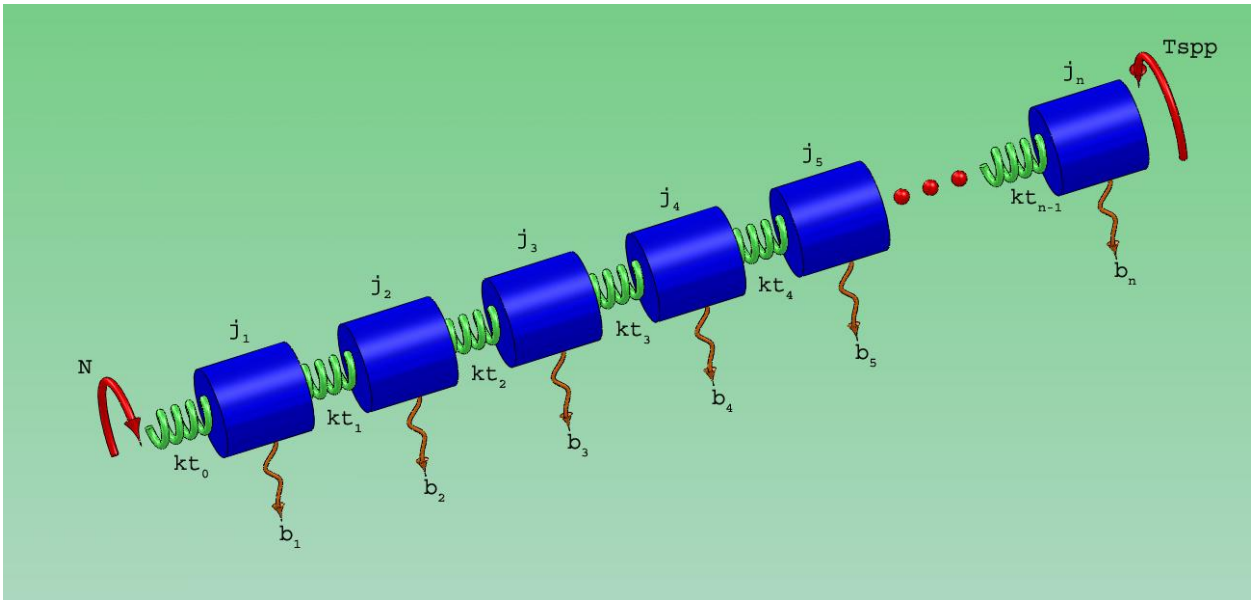

Figure 3-1: Shafting model mechanical system

The governing equation of the system shown in figure 3-1 is given as:

$$j \cdot \omega` = kt \cdot \delta\theta - b \cdot \omega + T \qquad (3\text{-}1)$$

Equation (3-1) can be re-written in terms of each shaft element as:

$$\begin{aligned} j_1\cdot\omega`_1 &= kt_0\cdot\delta\theta_0 - kt_1\cdot\delta\theta_1 - b_1\cdot\omega_1 \\ j_2\cdot\omega`_2 &= kt_1\cdot\delta\theta_1 - kt_2\cdot\delta\theta_2 - b_2\cdot\omega_2 \\ &\cdots \\ j_i\cdot\omega`_i &= kt_{i-1}\cdot\delta\theta_{i-1} - kt_i\cdot\delta\theta_i - b_i\cdot\omega_i \\ &\cdots \\ j_n\cdot\omega`_n &= kt_{n-1}\cdot\delta\theta_{n-1} - T_{spp} - b_n\cdot\omega_n \end{aligned} \qquad (3\text{-}2)$$

This can be written in vector form as:

$$\{\omega`\} = [Kt]\cdot\{\delta\theta\} - [B]\cdot\{\omega\} + [J]\{T\} \qquad (3\text{-}3)$$

Where [Kt], [B], and [J] are matrices given as follows:

$$Kt = \begin{pmatrix} \frac{kt_0}{j_1} & \frac{-kt_1}{j_1} & 0 & 0 & \cdots & 0 \\ 0 & \frac{kt_1}{j2} & \frac{-kt_2}{j_2} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \frac{kt_{i-1}}{j_i} & \frac{-kt_i}{j_i} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \frac{kt_{n-1}}{j_{n-1}} \end{pmatrix} \quad B = \begin{pmatrix} \frac{b_1}{j_1} & 0 & \cdots & 0 \\ 0 & \frac{b_2}{j_2} & 0 & 0 \\ 0 & \cdots & \frac{b_{n-1}}{j_{n-1}} & 0 \\ 0 & \cdots & 0 & \frac{b_n}{j_n} \end{pmatrix} \quad J = \begin{pmatrix} 0 \\ \cdots \\ 0 \\ \frac{-1}{j_n} \end{pmatrix} \qquad (3\text{-}4)$$

It should be noted that $\delta\theta`$ does not equal ω, but instead is given as:

$$\delta\theta` = \omega_{i+1} - \omega_i \qquad (3\text{-}5)$$

$$\delta\theta' = \begin{pmatrix} \delta\theta'_0 \\ \delta\theta'_1 \\ \cdots \\ \delta\theta'_{n-1} \end{pmatrix} = \begin{pmatrix} \omega_1 - N \\ \omega_2 - \omega_1 \\ \cdots \\ \omega_n - \omega_{n-1} \end{pmatrix} = L\cdot\omega + Ln\cdot N \qquad (3\text{-}6)$$

And this leads to the use of the permutation matrix L and Ln as follows:

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & -1 & 1 \end{pmatrix} \quad Ln = \begin{pmatrix} -1 \\ 0 \\ \cdots \\ 0 \end{pmatrix} \qquad (3\text{-}7)$$

L is used to transform the instantaneous shaft rotational velocity (ω) into the instantaneous shaft deformation rate ($\delta\theta$`), which is needed in order to integrate the deformation rate to the deformation angle. Ln is required to convert the input shaft rotational rate to the instantaneous shaft deformation rate of the first shaft element for addition to the remainder of the shaft elements as depicted in the Simulink module layout described in the following section and shown in figure 3-2.

3.3 Development of a Numerical Propeller Shaft Model Module

With the governing equations written in vector form and the matrices defined as in section 3.2, the shafting module was constructed in Simulink as shown in figure 3-2. As can be seen, all the matrices developed are built into the model. To populate the matrices, a Matlab script was generated that initiates the model. A single model initialization script, called PropulsionSystemConfigParamsV5.m, was written to initialize all the model modules at once and can be seen in Appendix A: Source Code. All the matrices will scale and self populate using this script upon variation of the input parameters as discussed in section 3.3.1. Tracing shaft rotational speed, the input comes from the prime mover output shaft speed at N, is multiplied by a gain factor of Ln (converting it to a shaft element deformation rate), then summed into the shaft element deformation rate vector. Once summed, the shaft element deformation rate vector is integrated to obtain the shaft element deformation vector, which is multiplied by a gain factor of the shaft stiffness matrix Kt. At that point, it is representative of the shaft torque losses due to shaft twist and is summed with the load torque (Tspp) and the bearing frictional loss torque. The sum of torques vector is then multiplied by the inverse of the inertia vector of the shaft elements yielding the acceleration vector. This acceleration is subsequently integrated to obtain the shaft rotational velocity vector, which is used to output shaft center of mass velocity and compute

shaft frictional losses. For the shaft speed output, the shaft rotational velocity vector is run through another gain block which takes an inertial average of the shaft speed. By inertial average, it is meant the shaft inertia vector is normalized by the total shaft inertia, which when vectorially multiplied by the shaft rotational speed vector yields the rotational velocity of the shaft's center of mass. For the frictional loss calculation, the same shaft output vector is multiplied by a frictional loss gain factor, B, and fed back into the torque load summation block. Finally, to complete the model, the shaft rotational velocity is multiplied by the gain factor, L, to convert it to the shaft deformation rate vector. To extract the torque load on the prime mover end of the system, the shaft deformation vector is multiplied by the shaft-prime mover flange stiffness divided by its inertia at gain block Kt0.

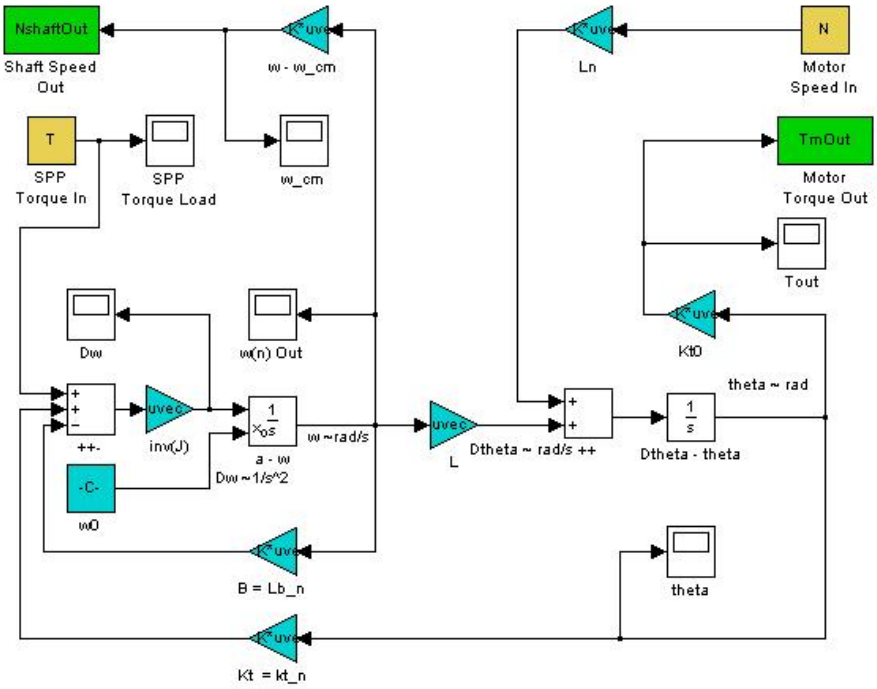

Figure 3-2: Shafting module Simulink model

3.3.1 Modular Model Inputs and Outputs

The shafting module accepts a torque load from the propulsor module where the model in figure 3-2 has the labeled input 'SPP Torque In'. The workspace variable T sets this value in Matlab. The other input to the shaft model is in the upper right of the figure, labeled 'Motor Speed In' and has the workspace variable N linked to it. The shafting module outputs a torque load to the prime mover model and speed to the propulsor model. To set the parameters, the initialization script (PropulsionSystemConfigParamsV5.m) is run, establishing the Kt, B, L, Ln and J matrices. Within the script is a section for modifying the shaft element lengths, diameters, densities, shear moduli, frictional losses, bearing locations, and the user can elect to use hollow shaft elements as desired. This gives the model a high degree of flexibility and allows the user to quickly and easily modify the shaft design. With the selection of Simulink as the modeling environment, additional outputs may be added to obtain data on shaft angle of twist, shaft rotational speed, acceleration, torque loss due to friction, and torque loss due to shaft twist easily. With these parameters available, the shaft model displays a powerful tool for dynamic shaft analysis.

3.3.2 Configuration with Realistic Friction and Stiffness

Built into the model is a default method of computing shaft torsional spring stiffness (which represents the stiffness of the shaft material itself) and frictional loss. The stiffness equation is the same as one would find in a mechanics text:

$$kt = G \cdot jp / Le \qquad (3\text{-}7)$$

where G is the shear modulus of the material, Le is the element length, and jp is the element's polar moment of inertia. The only parameters within the model that require modification are the

shear modulus and the density of the material. The frictional loss was left as a simple 5% overall frictional loss, and can be easily modified as desired. The theory involved in computing bearing friction by advanced fluid dynamic techniques is outside the scope of this research effort and was not considered. For practical purposes in the maritime world, selecting an overall frictional loss value, as was done, is typical and reasonable. The shaft material selected was steel and the minimum size of the shaft selected was based on limiting angle of twist to 1 degree per 2 meters of shaft length. Three different methods of computing minimum shaft diameter were executed: first by static failure analysis with a factor of safety applied, then with the limited angle of twist calculation (again based on static loading), and finally by the American Bureau of Shipping formula found in [3]. In comparing the values, it was startling to see that with low power output propulsion plants, the ABS calculation appears to be highly inaccurate, specifying a required shaft diameter less than one-fourth of the value obtained by static failure analysis. The limited angle of twist method is far more conservative than the other two imposing a shaft diameter of nearly four times that determined by failure analysis, using a FOS of 3 and an angle of twist limit of 1 degree per 2 m of shaft. The shaft length was chosen as 4 meters plus the propulsor for simulation purposes, and the shaft diameter was selected as a hollow shaft with 8 cm outer diameter and 1.5 cm wall thickness. Three bearings were placed on the shaft line: one at the prime mover flange, one at the propulsor mount, and one centered along the shaft line. Figure 3-3 shows a model of what the selected shaft looks like.

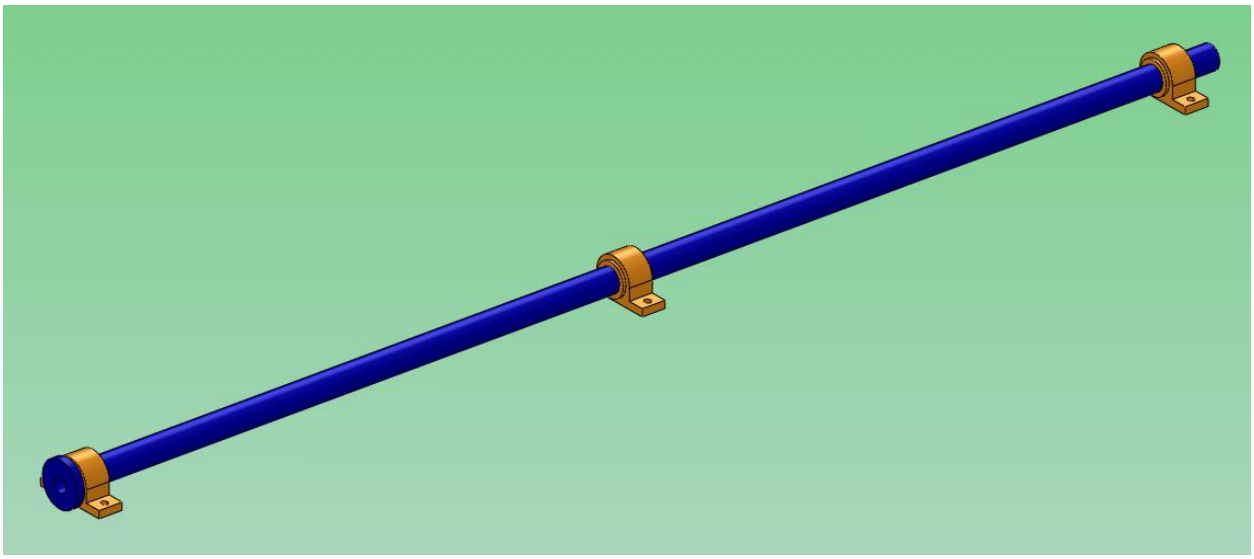

Figure 3-3: 3D CAD model of shaft as designed for simulation

CHAPTER 4: Development of a Prime Mover Model Module

The prime mover module implemented was the AC induction machine for reasons explained in section 1.2.1. Matlab's modeling package, Simulink, has a pre-programmed three phase asynchronous machine block based upon the simulation methods developed in 2002 by Krause et al. and in 1995 by Mohan et al.. Prior to the text referenced in the Matlab help files, Krause and Thomas released their 1965 paper titled "Simulation of Symmetrical Induction Machinery" in which they derived the basic equations describing the induction machine. It was decided that this model having absorbed considerable development effort should be investigated to determine its suitability for the prime mover module. To that end, the basic theory of induction motor modeling was investigated.

4.1 Basis of an Induction Motor Model Based on Equivalent Circuits

The traditional equivalent circuit for the AC induction motor in figure 4-1 is useful for the analysis of the steady state, however it can produce poor results when connected to power electronics and under transient operation [12]. This poses a major disadvantage and consequently a different model must be considered. It is fairly common to find the direct-quadrature (DQ) model of the equivalent circuit, as implemented in Matlab's three-phase asynchronous machine block, based upon an ideal transformer in which the secondary winding is rotating [4], [12], [13], [14]. Typically, the analysis is done on a per-phase basis, assuming balanced loads, and the DQ equivalent circuit is drawn as shown in figure 4-2.

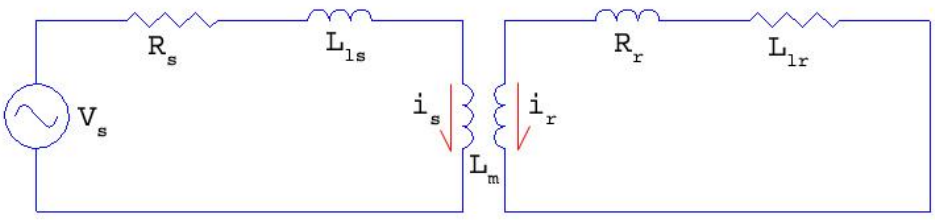

Figure 4-1: Equivalent per-phase circuit of the AC induction motor (steady state)

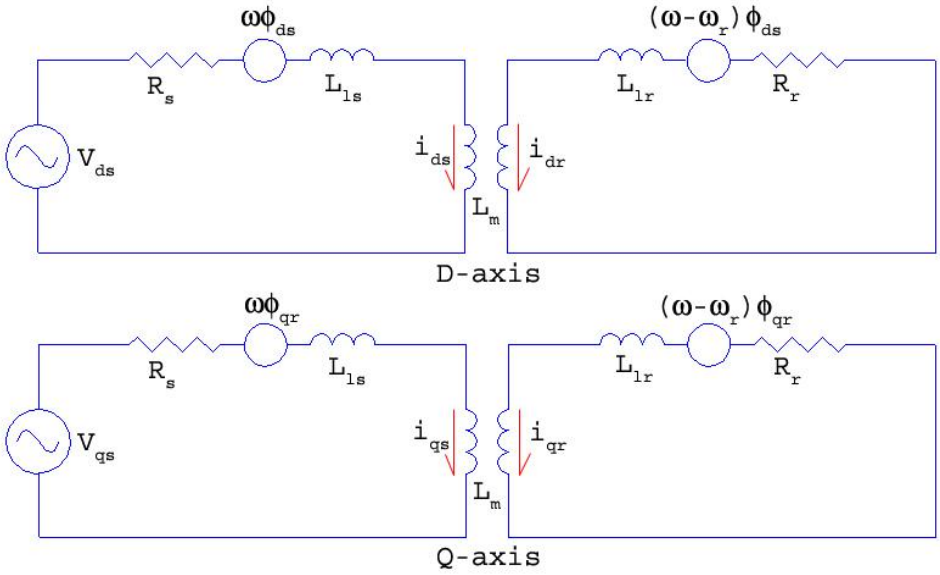

Figure 4-2: Equivalent per-phase circuit of the AC induction motor (dq)

The most detailed, commonly used AC induction motor model is the ABC/abc model, which does not transform the stator or rotor phase currents, allowing them to be tracked directly during the transient operation of the motor [12]. Due to the need to invert the time varying inductance matrix at every time step, this method of analysis is computationally intensive and therefore not used as frequently as the somewhat simplified ABC/dq, DQ/abc, or DQ/dq models. The primary assumptions made with the ABC/abc models are:

1. Sinusoidal magnetomotive force in the air gap. This assumption neglects the harmonic content and non-linear effects of the magnetomotive force.
2. Negligible saturation in the core.
3. Negligible iron losses in the core.

By further imposing two additional assumptions:

4. Balanced electrical loads in the stator.
5. Constant air gap (with a consequently constant flux).

The load in the rotor will be balanced and the inductance matrix can be simplified to the point it may be inverted analytically [12]. By further imposing changes in reference frame, the DQ/dq model can be developed. There are three general variations of the DQ/dq model:

1. The stationary reference frame DQ/dq model (Park transform). This model is useful for analysis of stator transients or of stator driven variable frequency drives.
2. The rotor reference frame DQ/dq model (Clarke transform). This model is useful for analysis of the rotor transients.
3. The synchronous reference frame DQ/dq model. This model can allow for larger time steps and the analysis of the motor stability.

DQ models cannot accurately model unbalanced loads and because of this, hybrid models have also been developed. Hybrid models are mixed ABC/dq or DQ/abc models and allow for direct analysis of either the stator or the rotor variables, respectively. In the hybrid models, the inductance matrix is not time varying so computation is not as time-consuming as in the ABC/abc model. For an in depth treatment of induction motor modeling, the reader is referred to [14].

4.2 Theoretical Basis and Limitations of the Matlab Asynchronous Machine Block

The Matlab three-phase asynchronous machine block within Simulink's SimPowerSystems Toolbox is based on the standard DQ/dq model described in section 4.1. It is a fourth-order state-space model of the electrical circuit, and although highly non-linear, provides some advantages to the control of the machine [15]. Using the DQ/dq model specifications, the three-phase asynchronous machine block assumes all of the assumptions listed in section 4.1 except assumption 3. The DQ/dq model in Matlab does allow for simulation of saturation. All three reference frames are available by modifying options within the block so it is suited to analysis of transients in the rotor, the stator, or for analysis of the field oriented control (using the synchronous reference frame). The block allows access to stator and rotor current in the abc and dq frames, the stator and rotor flux and voltage in the dq frame, and the rotor speed, angle and electromagnetic torque. According to the Matlab help file, care must be executed when using the model in discrete systems and with the type of power supply used. Details on the two issues should be referred to the Matlab help file on the asynchronous machine. Finally, it does not represent saturation of leakage fluxes.

4.3 Development of an Induction Motor Model Module

To establish the induction motor module, a method of adjusting speed must be implemented. Using the Simulink asynchronous machine block, the motor parameters must be specified, the outputs must be connected as required for interaction with other modules, and the input currents must be applied to the motor. Since the module has a requirement to execute variable speed operation of the induction motor, an appropriate power supply must be connected to the system. Three-phase AC power is typically supplied in either the Delta (Δ) or Wye (Y) topology. In the United States both topologies are available and equipment is still built for both types. However,

in many other parts of the world, Delta-connected power is not available, and Wye-connected power is the standard. This is partly because Wye-connected power includes a neutral point where all three phases come together from which one can obtain line-to-neutral voltages of 120V (for 208V 3-Phase Wye systems) or 266V (for 460V 3-Phase Wye systems). Use of Wye-connected power is advantageous as it can power either Delta or Wye designed loads, while Delta-connected power cannot supply power to Wye-connected loads that rely on use of the neutral point (for example, if there are low voltage control circuits in the load). For these reasons, it was decided to build a Wye-connected power source in the induction motor module. The 460V line-to-line Wye system was chosen as most marine generators output this industrial voltage and most high power motors are designed for high voltage. Furthermore, use of a step-down transformer to operate a 208V Wye-connected load is less efficient due to the resistive heating losses through the transformer. This leaves the means for variation of speed as the final component of the induction motor module power electronics package. Variable frequency drive was selected for open loop control of speed. While it is known that a variable voltage–variable frequency drive is more efficient, it was not selected as it is more costly, involving both a variable voltage source and high-speed switching devices in the power electronics as opposed to requiring only the high-speed switching devices.

A variation of the intended Matlab induction motor module was developed to show the effect of frequency variation on torque and speed output. Figure 4-3 shows the Simulink model used to produce speed torque capability plots of the selected induction machine under various drive frequencies, showing that by modifying the frequency, the speed for a given torque load can be varied. Figure 4-4 shows the speed torque capability curves for various drive frequencies

developed using the script MotorCapabilityPlotterV1.m. The script is available for review in Appendix A: Source Code. The speed torque capability curves presented in figure 4-4 were generated by plotting the electromagnetic torque output of the motor against its speed output for each of a series of frequencies used to excite the motor under a sinusoidal load. The load was selected based on the desired load rating of the system and was therefore varied slowly between 0 Nm and 150 Nm. The resulting curves represent the typical curve of a NEMA type-B motor if the initial transient is ignored, showing that the Matlab asynchronous machine represents a good model of the AC induction motor of NEMA type-B. From the literature, it may be seen that variation of terminal voltage on a NEMA type-B induction motor has only a small effect in speed variation, another reason to focus on frequency modulation for speed variation [4]. A speed perturbation correction control was added as an additional model module and is described in detail in Chapter 5.

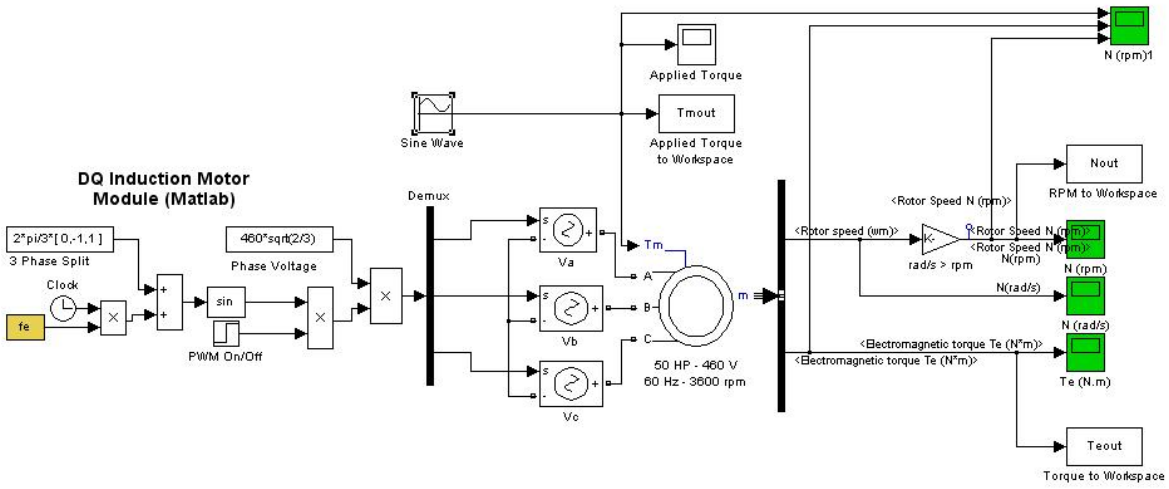

Figure 4-3: Matlab induction motor model used to generate speed torque capability curves

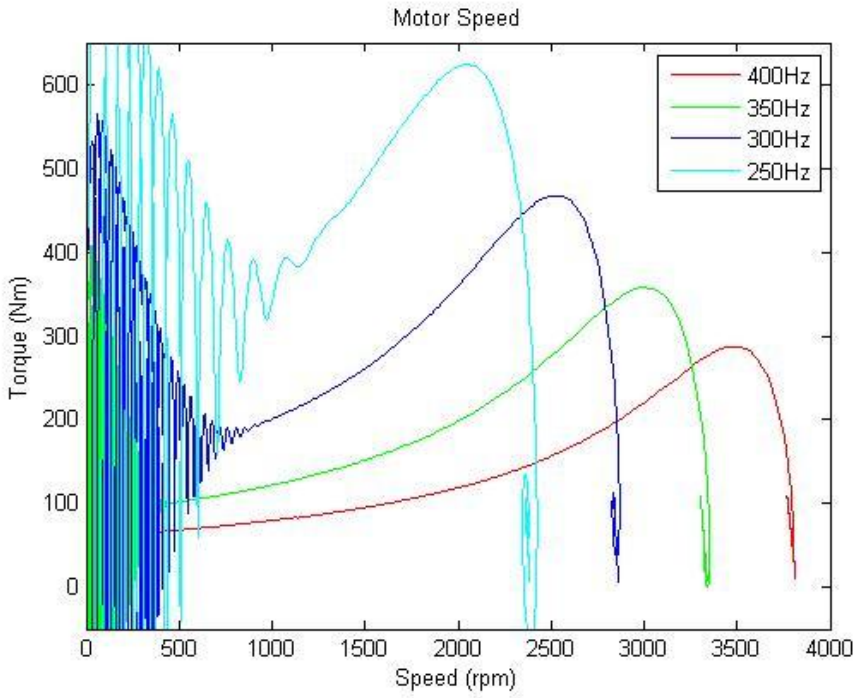

Figure 4-4: Speed torque curves of induction motor model sized at 50 HP,3-phase, 60 Hz, 3600 rpm

4.3.1 Modular Model Inputs and Outputs

The Asynchronous Machine block has available the input and output connections listed in Table 4.1. Table 4.2 lists the internally pre-set model parameters for the block. These parameters are not modified dynamically during the operation of the module, but are set within the model. The external inputs to the motor model are fed by the connected power electronics. With the selection of a variable frequency drive, the motor current input must be of variable frequency and therefore a pulse width modulation (PWM) inverter circuit was connected to the supply terminals of the asynchronous machine block. To achieve pulse width modulation, the system needs a current source and a high frequency switching device. PWM inverters are readily available and a simple model was developed. The PWM inverter modeled takes as input parameters the phase voltage which can be computed from the line voltage through rating the line voltage by the

square root of three. Further, the line voltage can be obtained from the rms line voltage by multiplying by the square root of two. The phase voltage is therefore given as:

$$V_{phase} = V_{line}/3^{0.5} = V_{rms}\cdot 2^{0.5}/3^{0.5} = V\cdot(2/3)^{0.5} \quad (4\text{-}1)$$

Available Input	Available Output	Utilized	Description
v_a, v_b, v_c		Y	Stator voltage for individual phases
	i_a, i_b, i_c	N	Stator current for individual phases
	i_d, i_q	N	Stator current in dq plane
	v_d, v_q	N	Stator voltage in dq plane
	ϕ_d, ϕ_q	N	Stator flux in dq plane
$v_a`$, $v_b`$, $v_c`$		N	Rotor current for individual phases (wound rotor only)
	$i_a`$, $i_b`$, $i_c`$	N	Rotor current for individual phases
	$i_d`$, $i_q`$	N	Rotor current in dq plane
	$v_d`$, $v_q`$	N	Rotor voltage in dq plane
	$\phi_d`$, $\phi_q`$	N	Rotor flux in dq plane
	ω_m	Y	Rotor mechanical rotational velocity
	θ_m	N	Rotor mechanical angular position
	T_e	Y	Electromagnetic torque

Table 4-1: Asynchronous Machine block input and output connections

Parameter	Value	Description
R_s, L_{ls}	0.09961Ω, 0.000867H	Stator resistance and leakage inductance
$R_r`$, $L_{rs}`$	0.05837Ω, 0.000867H	Rotor resistance and leaking inductance
L_m	0.03039H	Mutual inductance
Pn	37.300kW	Nominal power
V_{line}	460V	Line-to-line voltage (rms)
f_n	60Hz	Nominal supply frequency
J	0.2426kg·m^2	Rotor inertia
P	1	Number of pole pairs
F	0.02187	Friction factor

Table 4-2: Asynchronous Machine block parameter settings

4.3.2 Configuration for Variable Frequency Drive

Following the circuit of the PWM inverter section of the induction motor module in figure 4-5, a frequency input value is multiplied by the simulation time and added to a vector of phases [0, $2\pi/3$, $-2\pi/3$] to obtain a vector of instantaneous phase angles. The sine of the instantaneous phase angle vector is taken to obtain the modulated frequency. The modulated frequency is then

multiplied by the phase voltage to give a fixed voltage at the specified modulated frequency. Finally, the modulated frequency, fixed voltage vector is de-multiplexed and passed through a controlled voltage source block sending the output frequency modulated phase voltages to the asynchronous machine block.

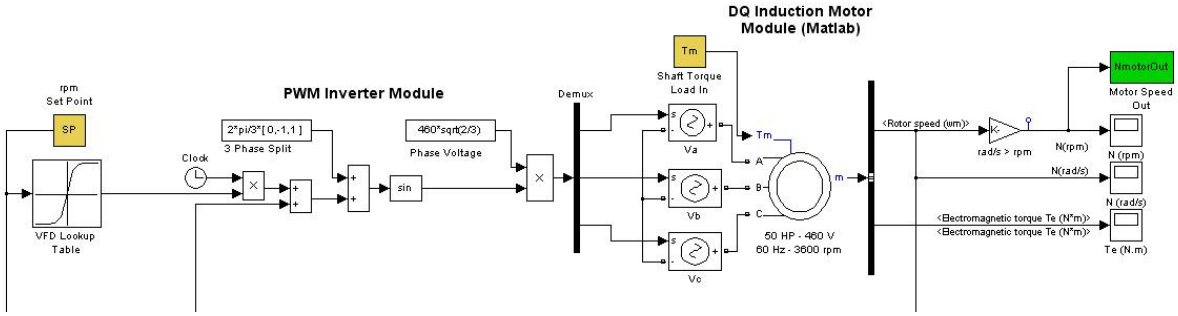

Figure 4-5: Induction motor module Simulink circuit diagram

In figure 4-5 there are three broken circuit lines which connect the motor to the perturbation correction controller described in chapter 5. Working from left to right, the disconnected lines are the supply frequency set point, the returned controller output, and the rotor speed output to the controller. Since the pulse frequency is dependent on both the torque load and the specific motor being driven, it is not an intuitive input for the operator and therefore the operator input is programmed to be an input set point in rpm. To convert the speed input to a pulse frequency a frequency lookup table must be generated. To generate the lookup table, a modified motor model was developed that has direct frequency input. With this model, shown in figure 4-6, a torque-demand load can be applied, the model runs until it reaches steady state, and the steady state rotor mechanical rotational velocity output is observed. If the torque demand for a given motor speed is a known desired value, the output speed can be compared to that value. The frequency can be adjusted in small increments until a satisfactory agreement is reached between the desired operational torque, speed and the motor's actual output speed. For the current effort,

the induction motor module is to drive the surface piercing propeller module, which provides a torque demand for a given speed and hence a speed-frequency lookup table can be generated.

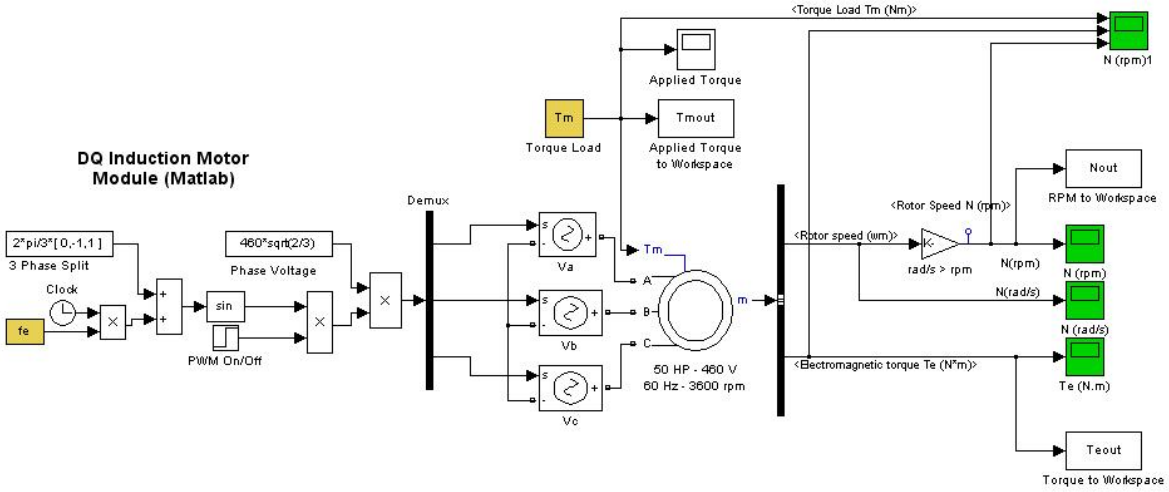

Figure 4-6: Induction motor module with direct frequency input

The surface piercing propeller module has a known torque demand as a function of speed (among other variables) as described in section 2.3.3. Since a perturbation correction controller was implemented in the overall propulsion plant model, the lookup table need not control the system 100% error free. Small deviations will be quickly corrected by the speed controller and therefore the lookup table can be based on the single speed torque curve of typical operational conditions. The typical operational conditions for the surface piercing propeller being modeled were selected as a forward speed of 10 knots, at 33% immersion, with zero degree pitch and yaw. These were the setting used to generate the speed torque curve followed for the speed-frequency table generation. As the iteration of each data point is a tedious task, a Matlab script was written to automate the task of generation of the speed-frequency table. This program, VFDLookupTableGenV3.m, and the explanation of its operation is covered in APPENDIX A: Source Code. The process of the speed-frequency table generation is illustrated in figure 4-7, and the generated speed-frequency table is plotted in figure 4-8.

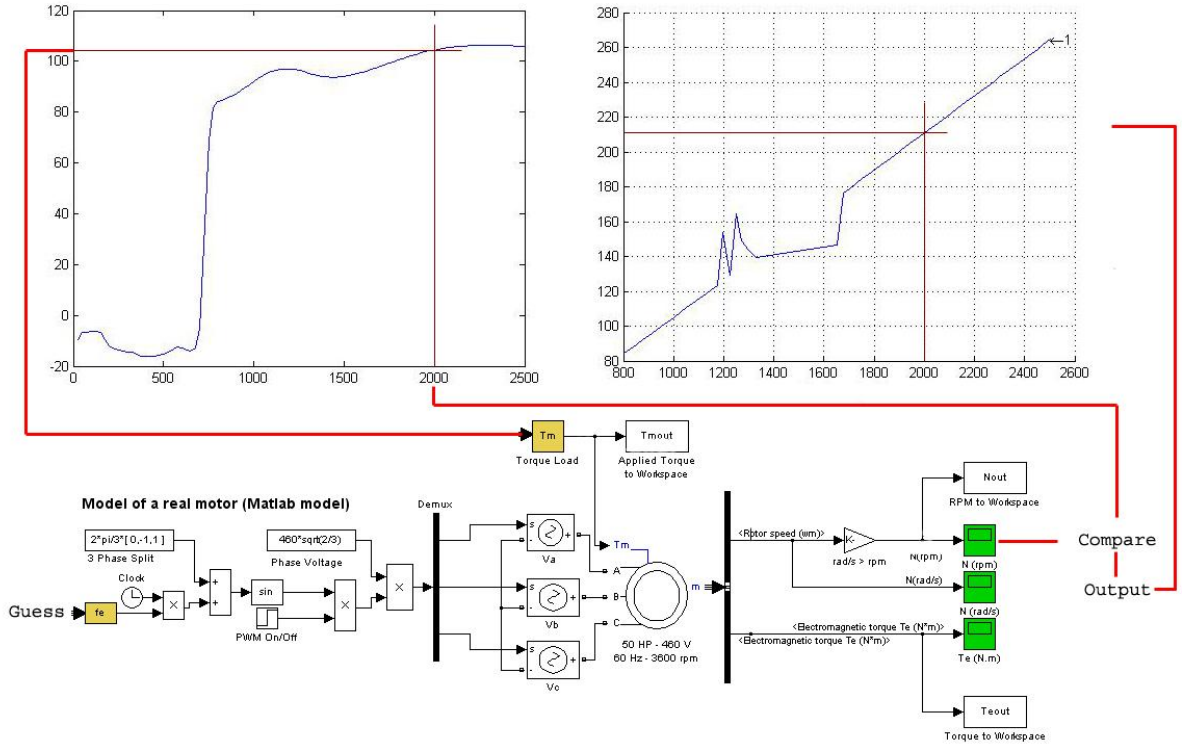

Figure 4-7: Generation of a speed-frequency table

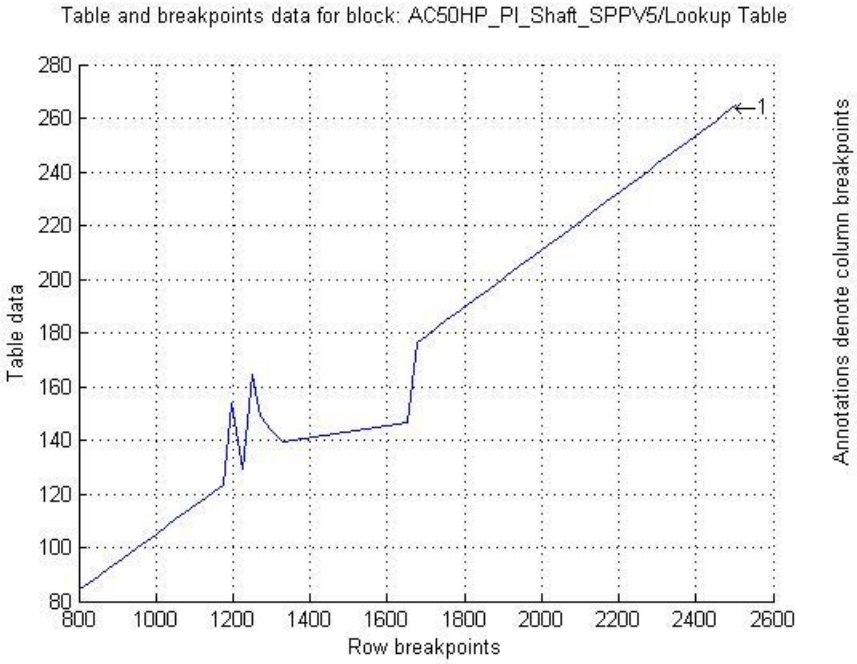

Figure 4-8: Generated speed-frequency table for induction motor

Upon completion of the speed-frequency generation curve, it was clear that a motor resonance occurs in somewhere in the speed range of 1175 – 1677 rpm. The resonance is excited at a speed referred to as the critical speed of the motor. As the rotor inertia plays an important role in the critical speed of the motor, changing the inertia of the motor will shift the critical speed. It was decided to re-generate the speed-frequency curve using the inertia of the motor plus that of the notional shaft and propeller as the inertia of the shaft and propeller is coupled to the motor and will shift the resonant frequency of the system. The motor parameters were modified in the direct frequency input induction motor module (see Table 4-1) to include the additional inertia (0.3911 kgm^2) of the shaft and propeller. Figure 4-9 presents the modified speed-frequency table generated considering the additional inertia of the shaft and propeller. This table was used for the simulations as it provides a more stable speed-frequency conversion and it more accurately carries out the conversion due to the consideration of the total connected inertia of the system.

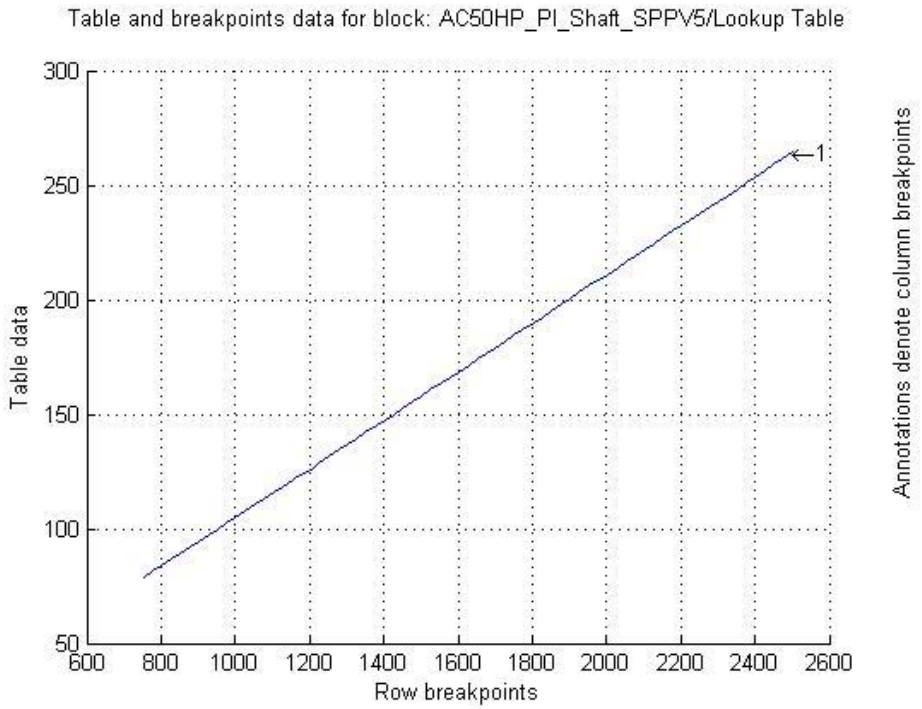

Figure 4-9: Generated speed-frequency table for induction motor with shaft and propeller inertia

4.3.3 Configuration with Realistic Motor Parameters

The induction motor module parameters were presented in Table 4-2. The electrical parameters of an AC induction motor are typically obtained experimentally from the particular motor to be used. As this model was designed to simulate the prime mover driving the surface piercing propeller, the peak torque of the propeller was estimated from the propeller module as approximately 110 Nm at approximately 2300 rpm under normal conditions. To provide for degradation of the hull due to fouling, loss of thrust due to wear and cavitation on the propeller and shaft, and unknown environmental conditions such as high sea state that may affect required torque, the output of the prime mover should be over-rated to allow the vessel to operate at rated speed under these adverse conditions. The over-rating factor commonly used is referred to as the sustained sea speed. Large ships, such as container and break-bulk cargo ships can have a sustained sea speed of about 80%, meaning they will operate at a maximum speed of 80% of the rated value under degraded operation (toward the end of a deployment period between dry-dockings). Prime movers are measured by their continuous service power, which is the power level provided during normal operations [3]. Often times the rated power of the prime mover is based on the torque load demanded during sea trial conditions in which the peak load of the propulsor is absorbed by the prime mover at a speed slightly above rated speed. For this reason, without a specific hull form selected, the rated torque of the prime mover was chosen with a 30% margin, producing a rated torque of approximately 143 Nm at a shaft speed of 2300 rpm. Equation 5-2 defines the power output of an electric motor, in which T_m is the load torque in Nm, P is the rated power in W, ω is the operating rotational speed in rad/s, and η is the efficiency of the motor.

$$P \cdot \eta = T_m \cdot \omega \qquad (4\text{–}2)$$

Using equation 5-2 with the torque and speed selected yields a required power of 34.442 kW. Rounding up to a standard motor rating yields a 50 HP (or 37.3 kW) motor. One commercially available motor is the WEG W21, 50-HP, 3600-rpm rated TEFC (totally enclosed fan cooled) motor. From the WEG website the efficiency of this motor at 75% load is 93.6% yielding an output torque of 145 Nm at 2300 rpm [16]. This makes the motor an ideal selection and with a total weight of 575 lbs the motor offers a potential weight advantage over a comparable CAT 3056 diesel of similar capability weighing in at approximately 1312 lbs. Furthermore, the CAT 3056 can develop 342.5 Nm at rated speed (2600 rpm), but only 49.9 Nm at 1000 rpm [17]. Clearly, with a surface piercing propeller being operated at highly varying shaft speed, the electric motor is a better choice. From the WEG website, the motor is listed with rotor inertia of .2426 kgm^2 [16]. The motor is a 2 pole, 3600 rpm, 460 V, three-phase motor with rated full load draw of 55.5 A. While the specific motor internal electrical parameters could not be located, Matlab has pre-programmed motors included in the asynchronous machine block, one of which is a 50-HP, 1800-rpm motor. The motor was selected and modified to contain 2 poles (and therefore a rated speed of 3600 rpm), and the proper rotor inertia value of 0.2426 kgm^2. The motor was tested at zero load (with only the internal friction load of the rotor applied) to confirm these modifications produced an output of 3600 rpm at rated input frequency. This test also provided the slip of the motor, and finally the motor was tested to determine the peak, full-load frequency input the motor can handle. To achieve the first goal, the motor was operated under the parameters, and yielded the outputs listed in Table 4-3.

Input Frequency (fe)	**Input Torque Load (T_m)**	**Output Speed (rpm)**	**Result**
60 Hz·2·π = 376.99Hz	0 Nm	3596.6 rpm	Figure 4-4
294 Hz (max)	143 Nm	2769.4 rpm	Figure 4-5
295 Hz (motor crash)	143 Nm	N / A	Figure 4-6

Table 4-3: Motor Test Parameters

The formula used to compute slip is given as:

$$s = (\omega_s - \omega)/\omega_s \qquad (4\text{-}3)$$

In equation 4-3, s is the slip, ω_s is the synchronous speed and ω is the rotor speed. The no-load slip of the machine (due to the internal friction) is just 0.094%. The output of the model under no-load conditions is shown in figure 4-10. At full load, if the frequency is adjusted up until just before the model fails, the peak speed at full load can be obtained, and the motor has significant slip at 23.072%. This situation is illustrated in figure 4-11. If the motor is driven with a higher frequency, the motor will not be able to develop additional speed and the model will crash as shown in figure 4-12. With the modified Matlab pre-programmed motor parameters producing good results, the motor was finalized with the realistic parameters listed in Table 4-2. With the prime mover, the shafting, and the propulsor modules complete, a perturbation controller was established, the development of which is described in Chapter 5.

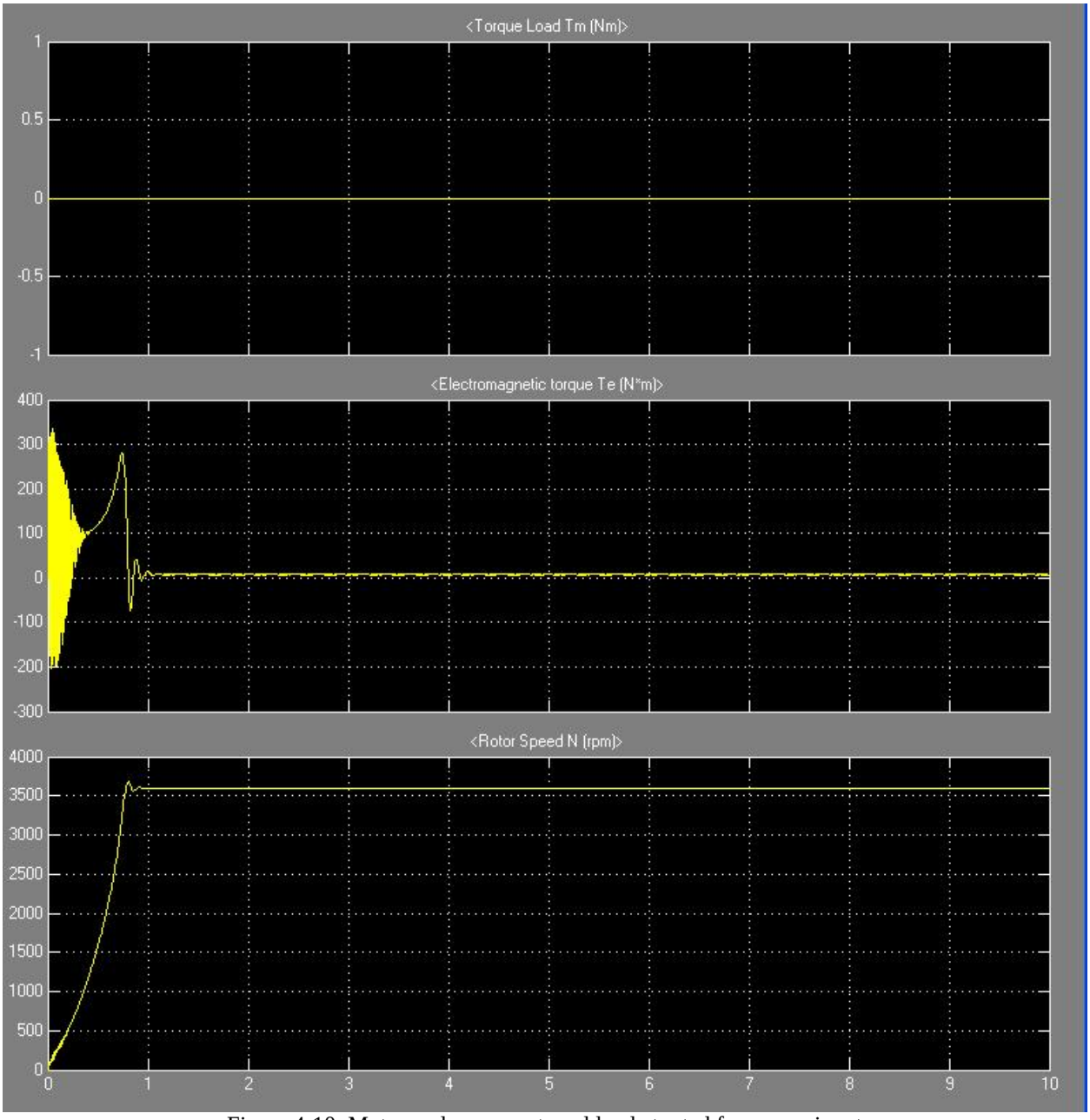

Figure 4-10: Motor under zero external load at rated frequency input

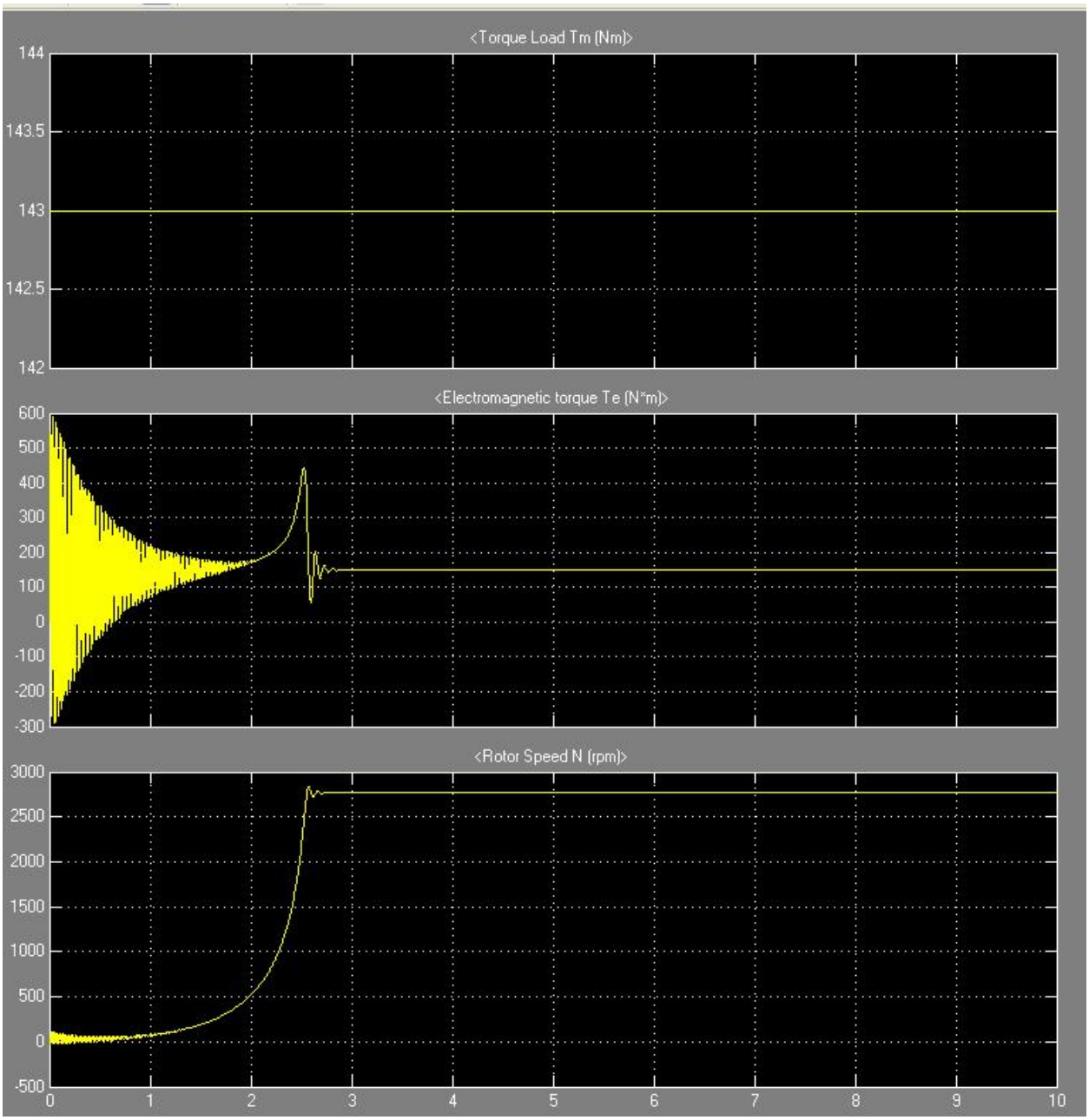

Figure 4-11: Motor under full load conditions at maximum speed

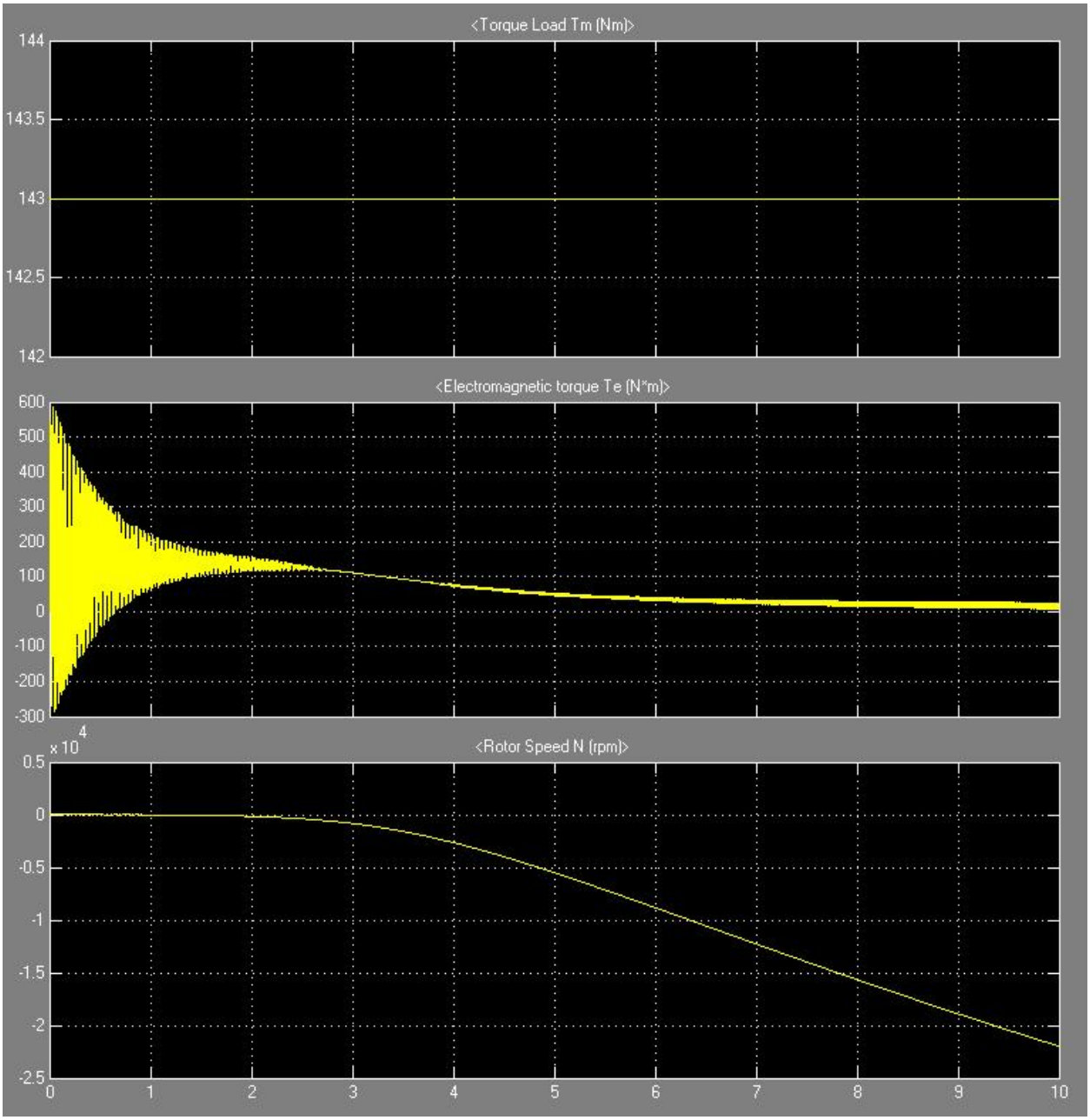

Figure 4-12: Motor under full load operated above maximum speed

When operated with the additional inertia of the shaft and propeller included in the induction motor module's inertia term, the curves generated for no-load, full-load and crash frequencies produced the same ultimate values, but took longer to reach the steady states. The modified parameter figures are presented as figures 4-13 and 4-14 for the no-load and full-load conditions, respectively. The values presented in Table 4-3 remained consistent.

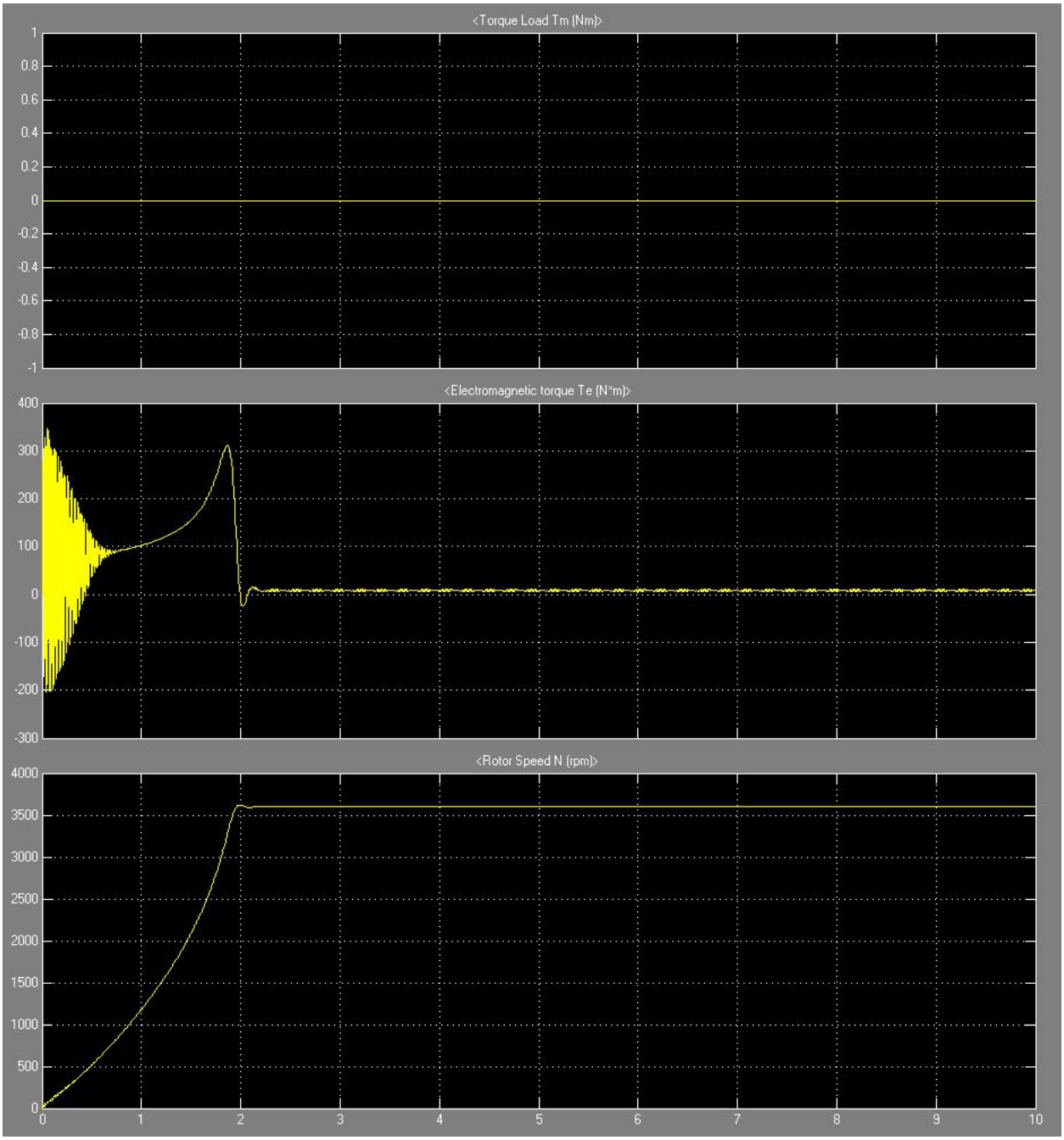

Figure 4-13: Motor under no-load conditions considering total system inertia

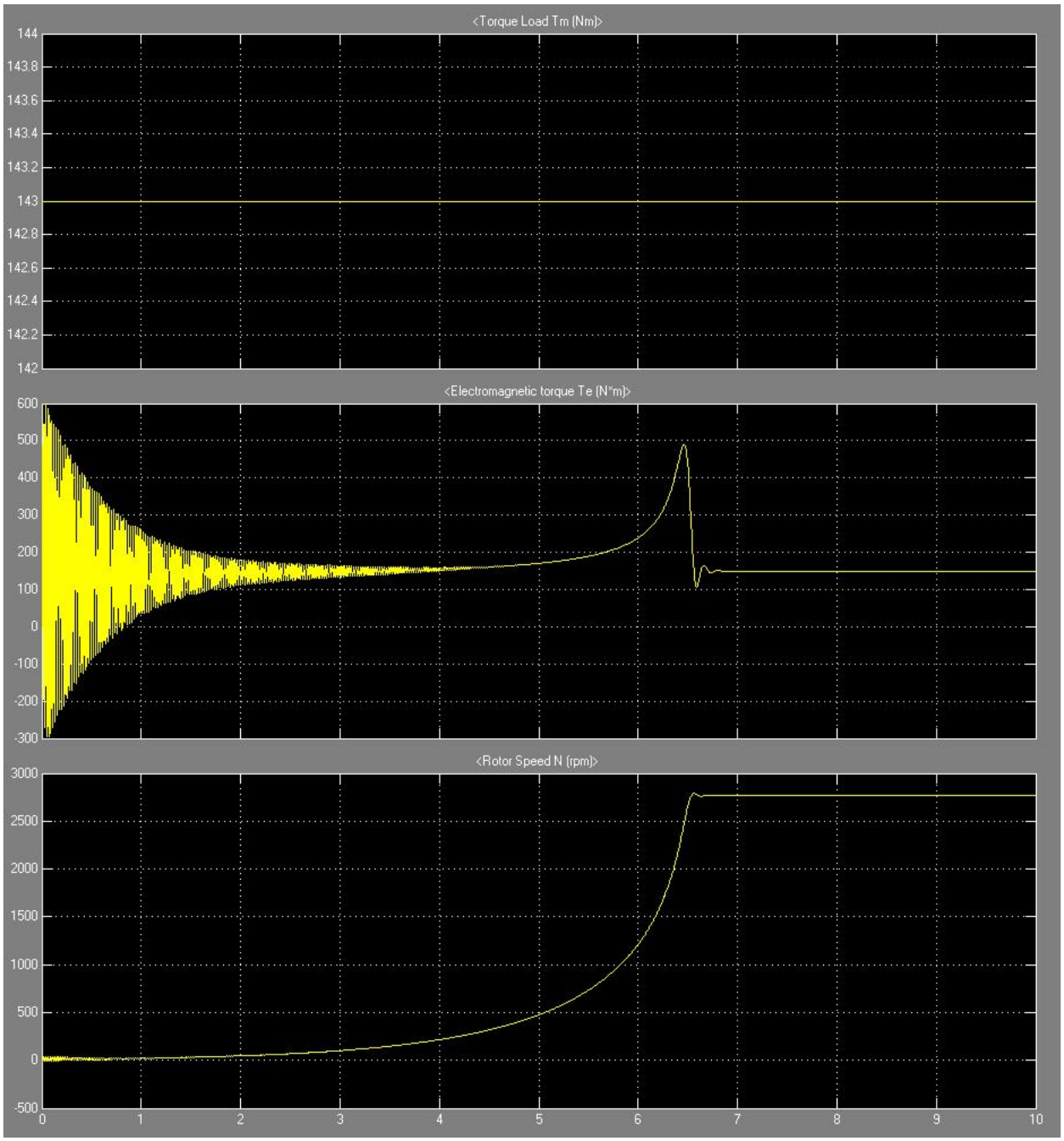

Figure 4-14: Motor under full-load conditions considering total system inertia

As can be seen in the plots, the time to reach full speed is increased, but the transient oscillation upon reaching full speed is diminished. Of note is the periodic torsional vibration seen in the steady state no-load electromagnetic torque.

CHAPTER 5: Development of a Control System Model Module

An open loop controller such as a speed-frequency lookup table can only accurately set speed under a given torque load if that speed and torque load combination is on the table. Unfortunately, a different speed-frequency table would be needed for each different propeller condition that modifies the toque load, such as changes in vessel speed, pitch due to planing, yaw under steering conditions, along with parameters not modeled by the propulsor module, including hull fouling, shaft and propeller wear, and sea-state conditions. As an alternative to gathering load data and mapping it for all anticipated conditions, the predominant set of conditions can be selected and the speed torque curve for those conditions used to generate a speed-frequency table as described in section 4.3.3, supplemented with a perturbation correction controller. The speed-frequency table will set a pulse width very close to what is needed and if one or more physical conditions causes a deviation from the expected load, the controller will add a correction signal to modify the pulse width, zeroing in on the speed set point.

5.1 Types of Control Systems Used for Induction Motor Control

There are few parameters that can be modified to control the speed of an induction motor. Following the equation for calculation of synchronous speed, one can clearly see that frequency of the supply voltage and the number of poles are the most direct way to vary speed.

$$N_s = 120f_e/P \quad (5\text{-}1)$$

In equation 5-1, N_s is the synchronous speed, f_e is the input voltage frequency and P is the number of poles in the motor. Modification of the number of poles in the motor is a physical change to the hardware of the motor and practically should only be done during motor selection,

based on required maximum speed. This leaves modification of frequency as the primary means of varying speed. To achieve this end, a PWM inverter is the standard solution, and with the invention of the thyristor in 1960, is a readily available technology today [4]. A survey of the literature will show significant research into induction motor control, which is not the focus of the present effort, and the reader is referred to [4], [15], [18] for detailed treatment of vector field control, speed sensorless control, and feedback control (state and output types). In his Master's Thesis, titled 'Electric Motor Control System with Application to Marine Propulsion', Camilo Carlos Roa developed a DQ model of the induction machine very similar to that of the Matlab asynchronous block for use with vector field control. In his text, Mr. Roa claims that a vector field control, which controls the internal currents of the induction motor as well as the input frequency, is capable of controlling both speed and torque output, at a cost of a more complex controller, which he recommends an effort be extended to develop [18]. For the present effort, the PWM inverter drive is based on a speed-torque demand generated lookup table with a PID controller implemented to compensate for deviations from the speed torque parameters defined as the most common curve.

5.2 Development of a PID Motor Control Module

The PID controller implemented for the control of the induction motor speed observes the actual mechanical speed of the motor via a speed sensor such as a hall effect sensor, or optical encoder, compares the value to the set point speed, and based on the difference between the two, generates a perturbation of the drive frequency. This perturbation is then summed back into the drive frequency and fed to the motor electrical power input pursuant to the speed-frequency lookup table described in section 4.3.3. To execute the modeling of such a controller, the basic theory is briefly presented. PID control stands for proportional, integral, derivative control and operates

by taking the control variable, and applying a proportional gain, an integral gain, and a derivative gain. Proportional gain accelerates convergence of the system, but if the gain is set too high, will cause instabilities manifested through oscillation of the control variable. Integral gain helps to smooth overshoot, and derivative gain provides damping of the system oscillations. In systems such as the present drive system where there is a large inertia connected to the motor, derivative gain often does not improve the control. The equation governing the control variable G(s) in the Laplace domain is given by:

$$G(s) = (K_p\, s + K_i \cdot + K_d \cdot s^2)/s$$

$$G(s) = K_p + K_i \cdot 1/s + K_d \cdot s \qquad (5\text{-}2)$$

$$G_{pi}(s) = K_p + K_i \cdot 1/s$$

The control equation can then be implemented directly in Simulink as the Laplace form is a direct input to the modeling software. The final form of the equation $G_{pi}(s)$ is the simplified PI form that assumes the inertia of the propulsion plant is enough to damp out oscillations in the system and lists only the proportional and integral terms [2]. While the full PID controller was established in the model, derivative gain can be set to zero effectively making the controller a simple PI controller. To implement the system, the inputs and outputs must be established.

5.2.1 Module Inputs and Outputs

The controller module inputs include actual speed of the induction motor and set point speed from the operator, and the output is the perturbation frequency. The actual motor speed is one of the available outputs from the prime mover module and was directly connected. The input set point speed was pulled from before the speed-frequency lookup table and converted to set point

speed in rad/s as speed error is the control variable. The induction motor output speed is fed to the controller directly as it is in the correct units, rad/s. The output of the controller is then converted to frequency and passed to the PWM inverter section of the induction motor module where it is summed into the frequency control signal. The controller module has PID control variable parameters of proportional gain, integral gain, and derivative gain. These three gains must be carefully set to be able to control the system they intend to regulate. Tuning of the controller is described in section 5.2.3.

5.2.2 Writing a simple PID Controller

The PID controller implemented in Simulink is depicted in figure 5-1. The module was built directly into the prime mover module rather than developing a separate module, as the prime mover module cannot serve a useful function without a control system, and the control system is selected and tuned to match the prime mover.

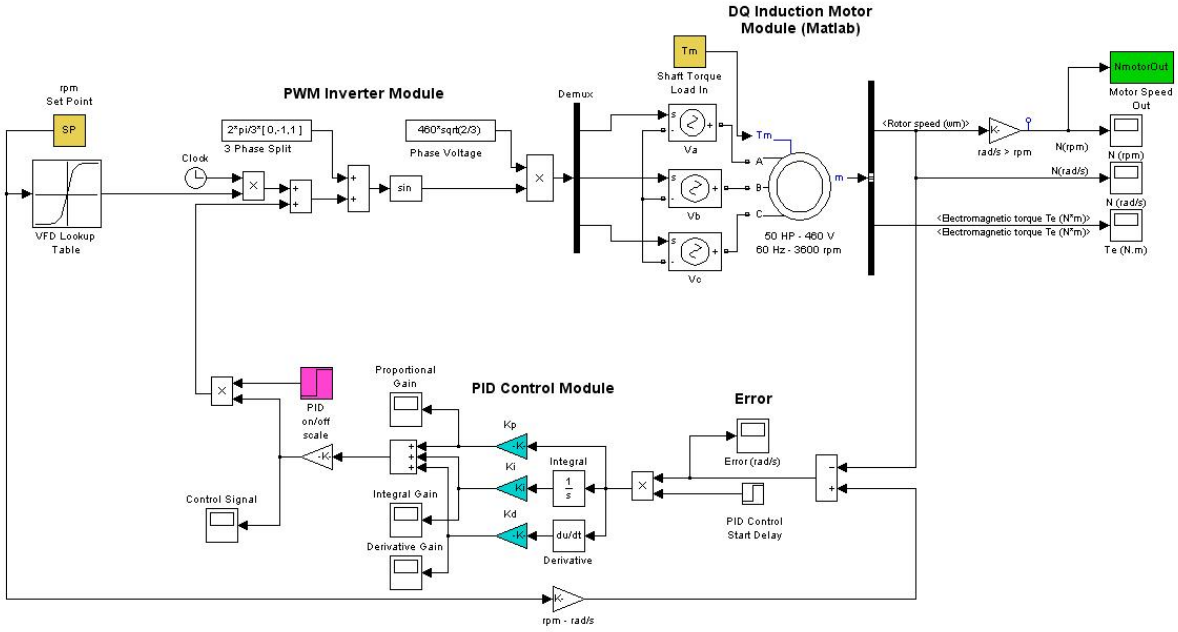

Figure 5-1: PID control module added to prime mover module

Due to the mathematical nature of the Simulink blocks, programming the controller, which consists of primarily mathematical manipulations is straightforward. Following figure 5-1, the set point speed is taken in and converted from a commanded rotational speed to a speed signal via a gain block marked 'rpm – rad/s'. This speed set point is then compared via a difference block forming the control variable, speed error. From this point, the control variable signal is split and simultaneously multiplied by the proportional gain, the integral gain, and the derivative gain. The signals multiplied by the integral and derivative gains are added to the signal multiplied by the proportional gain completing the manipulation of the PID control equation given by G(s) in equation 5-2. The last step is to convert the signal from a speed (rad/s) to a correction frequency (Hz) to be added to the current lookup table value. Along the path, one will notice two step source blocks. The first one, labeled 'PID Control Start Delay' allows the controller to be set to not act on the system at system start up. This avoids the PID controller trying to correct the large errors present during system start-up transients, which can slow convergence or in some extreme cases crash the controller due to large control signals causing system instability. The second one is labeled 'PID on/off' and simply turns the PID controller on or off. In off mode, the prime mover operates in open-loop mode with the only control implemented being the lookup table. With a complete model of the prime mover, the last step in controller development is tuning the controller for the system.

5.2.3 Tuning the PID Controller

While tuning of the PID control gain parameters, K_p, K_i, and K_d, requires running the complete propulsion plant, it is presented before Chapter 6 describes module integration to preserve the control system description as a self-contained chapter. The following information will assume a complete, integrated and tested model has been developed. For details on the integration and

testing of the modules, see Chapter 6. As PID control has been thoroughly developed, methods of tuning are simple and straightforward. The most common method used to obtain good results with a simple procedure is the Ziegler-Nichols tuning method, published in 1942 [19]. Following the Ziegler-Nichols method, the integral and derivative gain parameters are first set to zero and the proportional gain is slowly increased from zero until oscillation of the control variable (shaft rotational velocity error in the present case) initiates. This value is recorded as the ultimate value, K_u. The period of oscillation T_u is also recorded at K_u. With these two values obtained, the controller can be configured depending on how it was set up according to Table 5-1. To use the table, one must decide in advance if P, PI, or PID control will be used, and based on that decision, set the tuning parameters. Theoretically, the basic settings will produce $\lambda/4$ tuning, which implies that the system will stabilize within four cycles [2].

Control Method	**K_p**	**K_i**	**K_d**
P	$K_u / 2$	N/A	N/A
PI	$K_u / 2.2$	$0.55 \cdot K_u / T_u$	N/A
PID	$0.60 \cdot K_u$	$1.2 \cdot K_u / T_u$	$.075 \cdot K_u \cdot T_u$
Improved Speed of Convergence	Manually increase K_p	N/A	N/A
Improved Oscillatory Response	Manually reduce K_p	N/A	N/A

Table 5-1: PID Tuning Parameters (adapted from [2])

The integral and derivative gains are theoretically dependent on the system being controlled and therefore should not be manually modified to improve performance. Of importance in tuning the system is the fact that tuning at an abnormal operational condition in which transient effects are present can lead to significantly reduced performance. If a system resonance exists, the tuning should be done at operational conditions away from this resonance. To begin tuning, the shafting model was used to determine the vibrational natural frequencies of the shaft using

eigenfrequency analysis. The eigenfrequencies of the shaft can be determined by taking the eigenvalues of the following matrix of shaft design parameters:

$$\begin{pmatrix} 0 & -L \\ Kt & -B \end{pmatrix} \qquad (5\text{-}3)$$

Using the shafting module configuration parameters for a 10 element shaft, the eigenvalues of the system are given in Table 5-2.

Natural Frequency (rad/s)	Exciting Speed (rpm)	Exciting MMF Frequency (Hz)
-9133.2	N/A	N/A
9133.2	87,216 (out of range)	9133.2
-2469.1	N/A	N/A
2469.1	23,578 (out of range)	2469.1
-2003.5	N/A	N/A
-1563.4	N/A	N/A
-1222.0	N/A	N/A
-1149.4	N/A	N/A
2003.5	19,132 (out of range)	2003.5
-857.4	N/A	N/A
-676.8	N/A	N/A
1563.4	14,930 (out of range)	1563.4
-378.6	N/A	N/A
-130.3	N/A	N/A
130.3	1,244	130.3
378.6	3,615 (out of range)	378.6
676.8	6,463 (out of range)	676.8
857.4	8,187 (out of range)	857.4
1222.0	11,669 (out of range)	1222.0
1149.4	10,976 (out of range)	1149.4

Table 5-2: Natural frequencies of 10 element shaft

As Table 5-2 makes clear, the only excitation within the operational speed range of the shaft occurs at approximately 1244 rpm. For this reason, tuning was carried out at two separate set points, 800 rpm and 2400 rpm, both within the range of surface piercing propeller operation and

away from the shaft resonance point. The selection of actual tuning parameters was completed during the simulation portion of this research effort and is presented in Chapter 7.

CHAPTER 6: Integration of Powertrain Model Modules

Integration of the propulsion plant modules must consider the inputs and outputs of each model and how they should be interconnected. Each module should only interact with those portions of the propulsion plant to which it is physically connected. To that end, the control system module interacts with the prime mover module, receiving a speed measurement from the prime mover output shaft and feeding back a speed correction signal to the PWM inverter of the induction motor module. Similarly, the shafting module takes in the prime mover rotational speed and transmits that speed to the propulsor module while simultaneously accepting the torque demand of the propulsor module and feeding it back to the prime mover module. Finally, the propulsor module accepts the commanded rotational speed from the shafting module and feeds back the torque demand generated by realizing that propulsor speed. By combining all the sub-system modules of the propulsion plant model, an overall model is generated that can operate within a realistic operational envelope.

6.1 Overall Model Inputs and Outputs

The integrated model has many setup parameters based on the system design. The number of inputs, however, is reduced to a single commanded shaft speed. This was done as shaft speed is typically the only parameter that a helmsman operates to speed up or slow down the vessel. With the complete model running at the commanded speed, model output can be tailored to those details of interest to the operator. For example, one may be concerned with the electromagnetic torque generated by the induction motor, the torque demanded by the propulsor, absorbed by the shaft due to twist, or that lost to bearing friction. The level of detail can be drilled down to the shafting element where the torque absorbed by a single element due to the amount of element twist is available in the model. If the primary concern is the vibration of the shaft, one can

consider the accelerations of the available shaft elements. Any signal passed through the overall model can be viewed or recorded by connecting scopes or workspace outputs in the Simulink model.

6.2 Using Simulink to Combine Modules into the Complete Model

One major concern with combining multiple models into overall system models, as was carried out in the present research, is properly setting the integration time steps of the models so that they provide sufficient detail. Simulink was selected primarily for this purpose as it automatically accounts for the minimum time step required by the simulation. If one model requires a 1.3-second time step while another connected model requires a 0.3-second time step and a very detailed portion of the model requires a 3-millisecond time step, Simulink will configure the overall model to run at the smallest time step. Although this is potentially inefficient, the program does not miss major information due to operation at a time step too large for the most detailed model. Simulink further allows one to easily change the duration of the simulation and the graphical interface makes it very easy to add output signals and plot data.

To complete the model, each of the individual modules was pasted into a common model document and connected directly. Simulink allows the definition of sub-models, but this makes access to individual signal lines more difficult and the flow of information less clear. For those reasons, all modules were placed into a common model file where alteration simply involves removing and replacing that portion of the model associated with the relevant module. The final Simulink model of the complete marine propulsion plant is presented in figure 6-1. The color coding of the model was established to simplify identification of outputs (Orange) workspace

inputs (cyan), disabling switches (dark blue) and self-contained input-output blocks (light blue). The script PropulsionSystemConfigParamsV5.m populates the workspace with inputs.

6.3 Model Comparison Testing Using a Simplified Model as a Baseline

To validate the model modules prior to assembly into a single marine propulsion plant model, each of the three primary mechanical systems was validated against a greatly simplified version provided by Dr. Xiros, specifically for this purpose. The details of the comparison testing and stability analysis of the system may be reviewed in Appendix B: Model Comparison Testing. While the comparison testing does not verify model parameters are specified correctly or realistically, it does provide evidence that the models function correctly and produce data consistent with an alternative model. With the modules validated, assembly was carried out as explained previously in section 6.2.

While testing the overall model, it was discovered that with extended simulation times, the system would crash suddenly while operating under PID control and the controller would not be able to recover. The modularity of the model made evaluation of errors far simpler than a single unified model without the modularity would allows. Simple models were substituted one model at a time to determine which sub-model was causing the crash. It was quickly discovered that the motor model was the source of the error and in applying sinusoidal signals to different positions on the model, it was learned that the PWM inverter was originally designed incorrectly. The speed correction input from the PID controller was being fed to a point too early in the system and was being amplified by the clock signal making the error grow with time. This not only led to instability of the control system, but to incorrect tuning parameters as well. The summing block was moved, and the controller performed flawlessly.

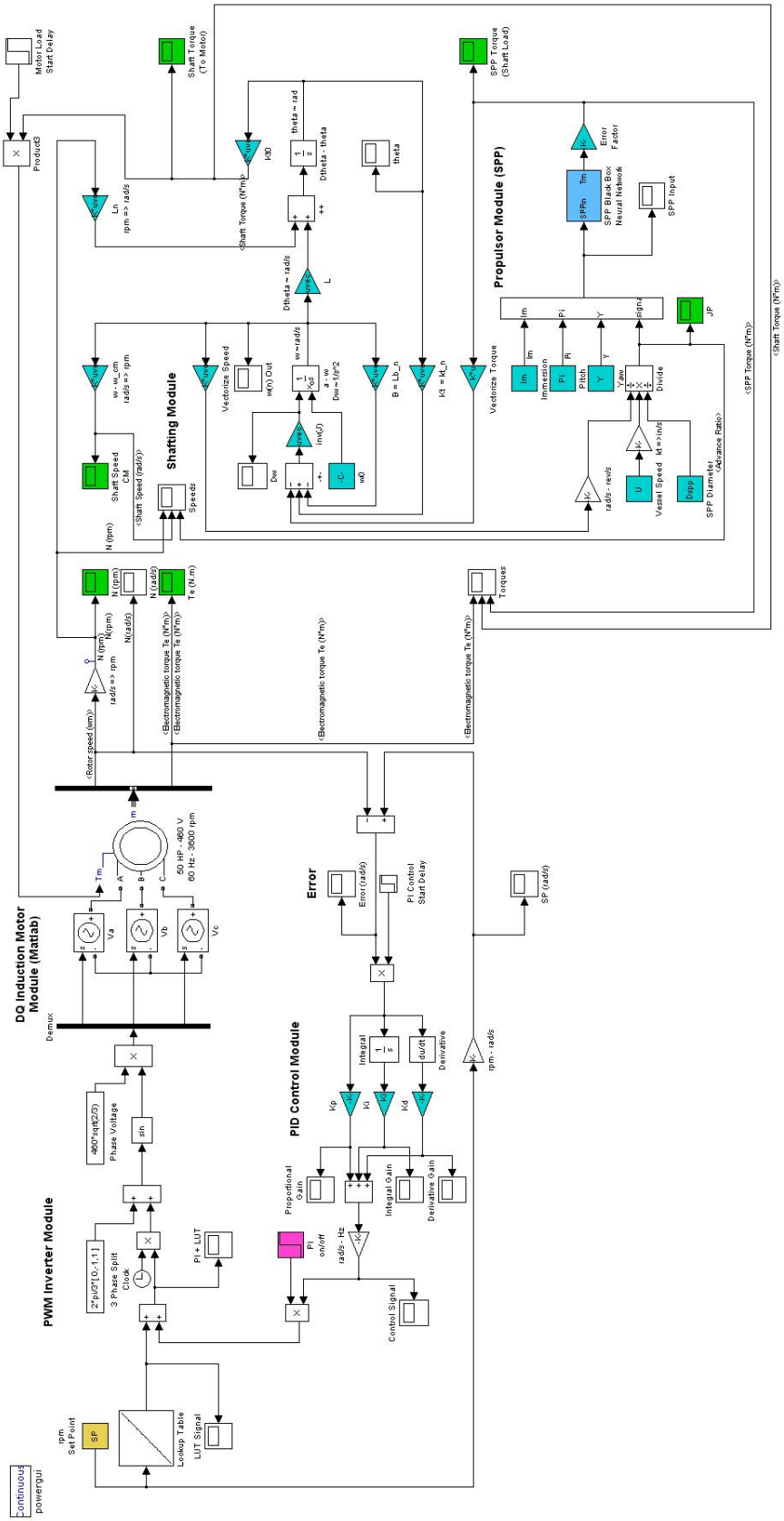

Figure 6-1: Complete marine propulsion plant model

CHAPTER 7: Simulation of a Marine Powertrain with Realistic Parameters

The simulation of a marine powertrain composed of an AC induction motor, a hollow steel propulsion shaft, and a surface piercing propeller was carried out to investigate the effect of shaft resonance on the stability of the model. The first step in the process was the definition of the powertrain components, followed by a series of simulations at constant shaft speed without the use of the perturbation correction controller, and then application and tuning of a PID controller.

7.1 Simulation Vessel Powertrain Specification

The vessel to be modeled was assumed to be approximately 8 m in length with a propeller shaft length of 4 m. The propeller to be used is the Dewald surface piercing propeller analyzed in Chapter 2. The AC induction motor was selected as a 50-HP motor as discussed in section 4.3.3. The 4-m shaft was assumed to have three bearings, one at each end, and one centered along the shaft. The shaft end bearings each occupy a 15-cm long section of shaft and the center support bearing a 10-cm length section. The hull form and thrust output would ultimately dictate what the vessel's capabilities are, but integration of a hull form module is left to future expansion of the present research effort.

7.2 Simulation Run Parameters Specification

Based on the notional vessel powertrain, the run parameters are listed in Table 7-1. While selected based on realistic powertrain component sizes, there was no effort made to match the parameters to a specific hull form, so the purpose and performance of the overall vessel cannot be known from simulations carried out as described herein. However, the performance of the propulsion plant can be observed and analyzed in detail with this model. Furthermore, modifications can be made to the run parameters to change the performance based on observations made during simulations.

Parameter	Value	Description
SPP Torque	SPPnet	Neural network, black box of measured surface piercing propeller data with inputs of: pitch, yaw, immersion, and advance ratio
n	10	Number of shaft model sections
Le	[.15, .6, .6, .6, .1, .6, .6, .6, .15, .1	Length of shaft elements: each element length can be different and if an element is set to have a bearing at that location, the bearing frictional loss acts on the length of that element
nb	[1, 0, 0, 0, 1, 0, 0, 0, 1, 0]	Location of shaft bearings: 1 indicates a bearing exists at the given shaft element
Lb	.0083	Bearing frictional loss in N·m·s/rad
ρ	7850	Shaft material density in kg/m^3
G	8.2e10	Shaft material shear modulus in Pa/rad
D1	[.08, .08, .08, .08, .08, .08, .08, .08, .08, .2464]	Shaft element outer diameters: last element represents the propeller
D2	[.05, .05, .05, .05, .05, .05, .05, .05, .05, .00]	Shaft element inner diameters: last element represents the propeller (modified inner diameter to produce appropriate propeller inertia)
H	[1, 1, 1, 1, 1, 1, 1, 1, 1, 0]	Hollow shaft element indicator: 1 indicates an element is hollow
Lf	.05	Length of shaft-motor flange where motor and shaft meet
Df	.15	Diameter of shaft-motor flange where motor and shaft meet
Kp	0.18	Proportional gain setting
Ki	4.7143	Integral gain setting
Kd	7.875e-4	Derivative gain setting
Pidelay	0	Time delay for initiation of PID controller
Tfactor	-10	Torque error correction factor discussed in section 2.3.1
Dspp	9.7	Propeller diameter (in)
Im	33	Immersion of propeller (%)
Y	0	Yaw of propeller (degrees)
Pi	0	Pitch of propeller (degrees)
U	10	Vessel speed (knots)

Table 7-1: Simulation run parameters

7.3 Demonstration of Shaft Resonance

Based on calculations made in section 5.2.3, there exists a shaft resonance at 1244 rpm. To demonstrate that this resonance exists, a series of plots were generated for the torque and shaft speed at various speeds from 800 rpm to 2400 rpm with a vessel speed of 10 knots. The range was selected based upon the speed-torque plots obtained for the surface piercing propeller. The

800 rpm is toward the low end of shaft speed that will still produce a positive torque demand at a vessel speed of 10 knots, and 2400 rpm produces an advance ratio of approximately 0.5, just beyond the range of the data collected by Dr. Karl von Ellenrieder. A Matlab script called, ResonancePlotterV1.m, was created to cycle through a series of constant speed runs of the integrated propulsion plant model AC50HP_PI_Shaft_SPPV8.mdl and generate plots of the speeds and torques. To execute the test without the effects of a controller, the tests were conducted with the PID controller gains set to zero. Because the open loop lookup table does not account for all system losses, a slight offset exists between actual speed and set point speed. Figures 7-1 and 7-2 show the speeds and torques of the system stepping the set-point speed from 800 to 2400 rpm in 200-rpm increments. Shaft resonance is clearly visible examining the 1200 rpm (blue) lines in both figures. Further investigation was carried out in an attempt to zero in on a more accurate value for the resonance point. To that end, speed and torque plots were generated for steps of 50 rpm between set point speeds of 1000 and 1300 rpm. In examining these two sets of plots, reproduced in figures 7-3 and 7-4 below, it is apparent that the largest amplitude oscillations occur at the 1200 rpm (magenta) line, demonstrating that the resonance is actually closer to 1200 rpm than 1250 rpm, indicating the 1244 rpm computed value of shaft resonance does not predict the overall system resonance conclusively.

7.3.1 Tuning a PID Controller Around Shaft Resonance

To avoid selection of poor PID gain parameters, tuning should be carried out where system anomalies are not present as discussed in Chapter 5. With evidence pointing to an actual system resonance existing at approximately 1200 rpm, tuning was carried out at both extremes of the operational envelope, 800-rpm and 2400-rpm set points. At 2400 rpm, an ultimate gain value of 0.4 and ultimate period of oscillation value of 0.035 s was observed. At 800 rpm, the ultimate

gain was 0.2 and the period was 0.035 s. Using these values, the averages were taken and the gain parameters listed in Table 7-1 were computed via the Ziegler-Nichols method and used as a baseline perturbation controller.

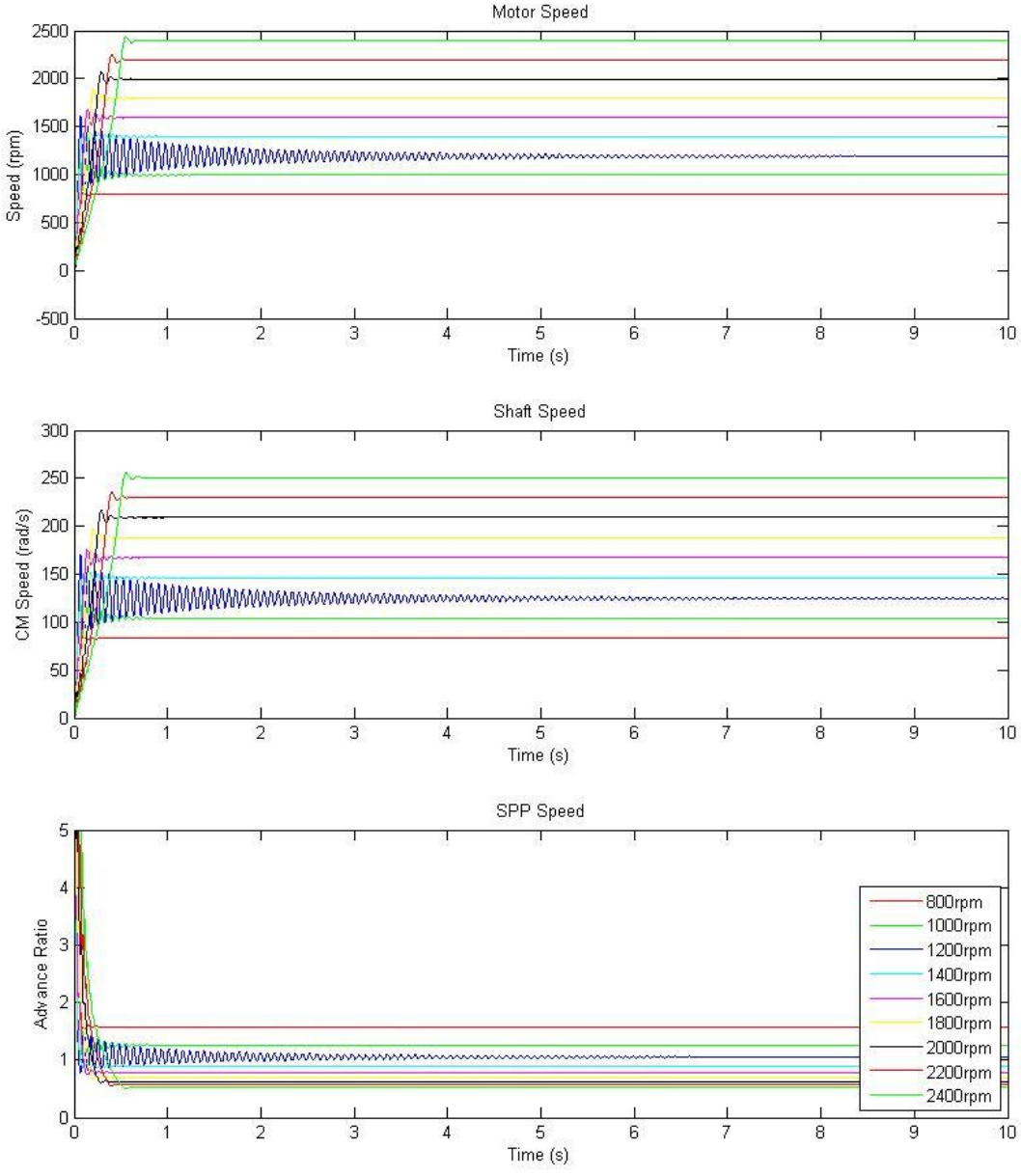

Figure 7-1: Constant set point simulation output of actual speed

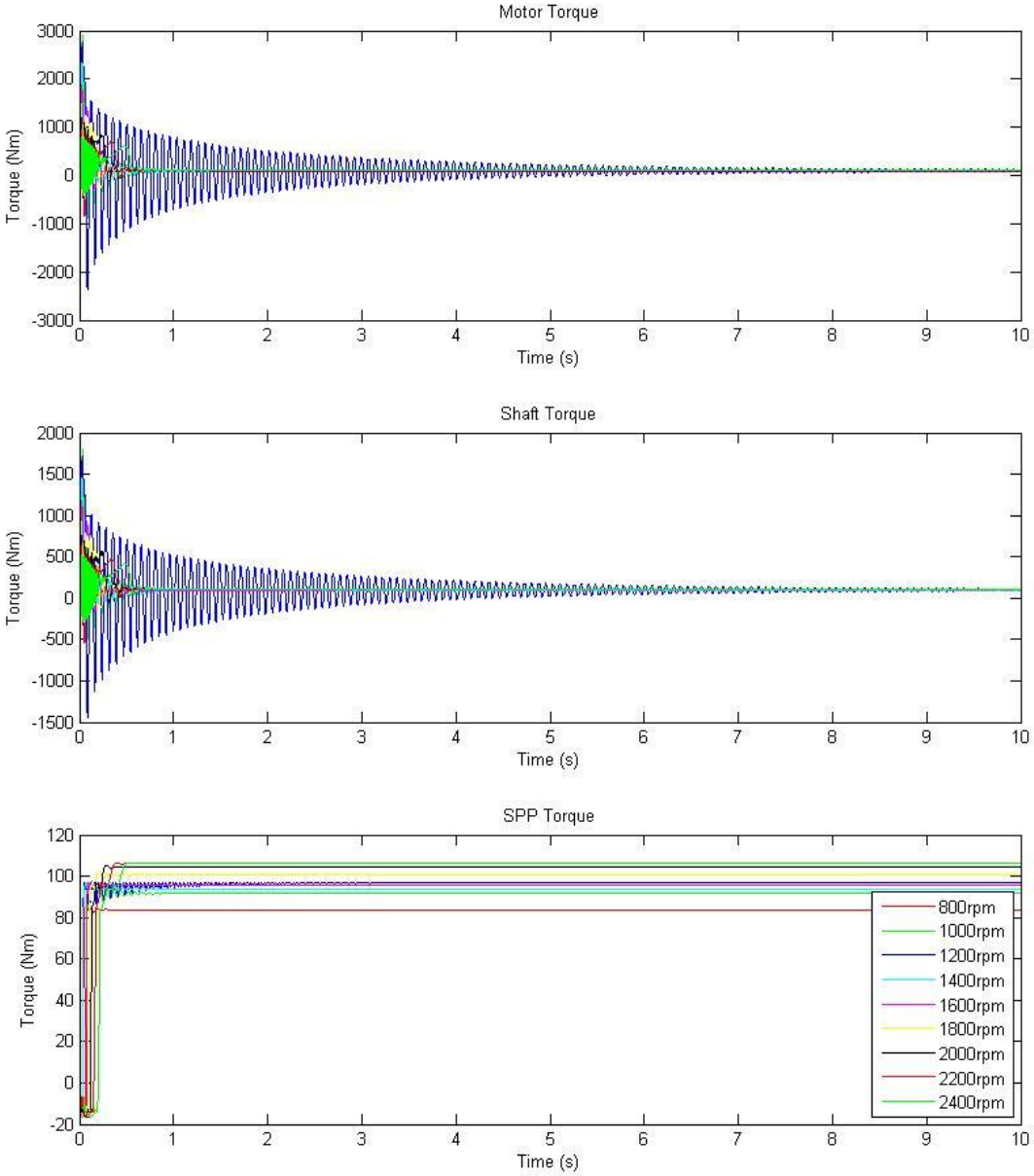

Figure 7-2: Constant set point simulation output of torque

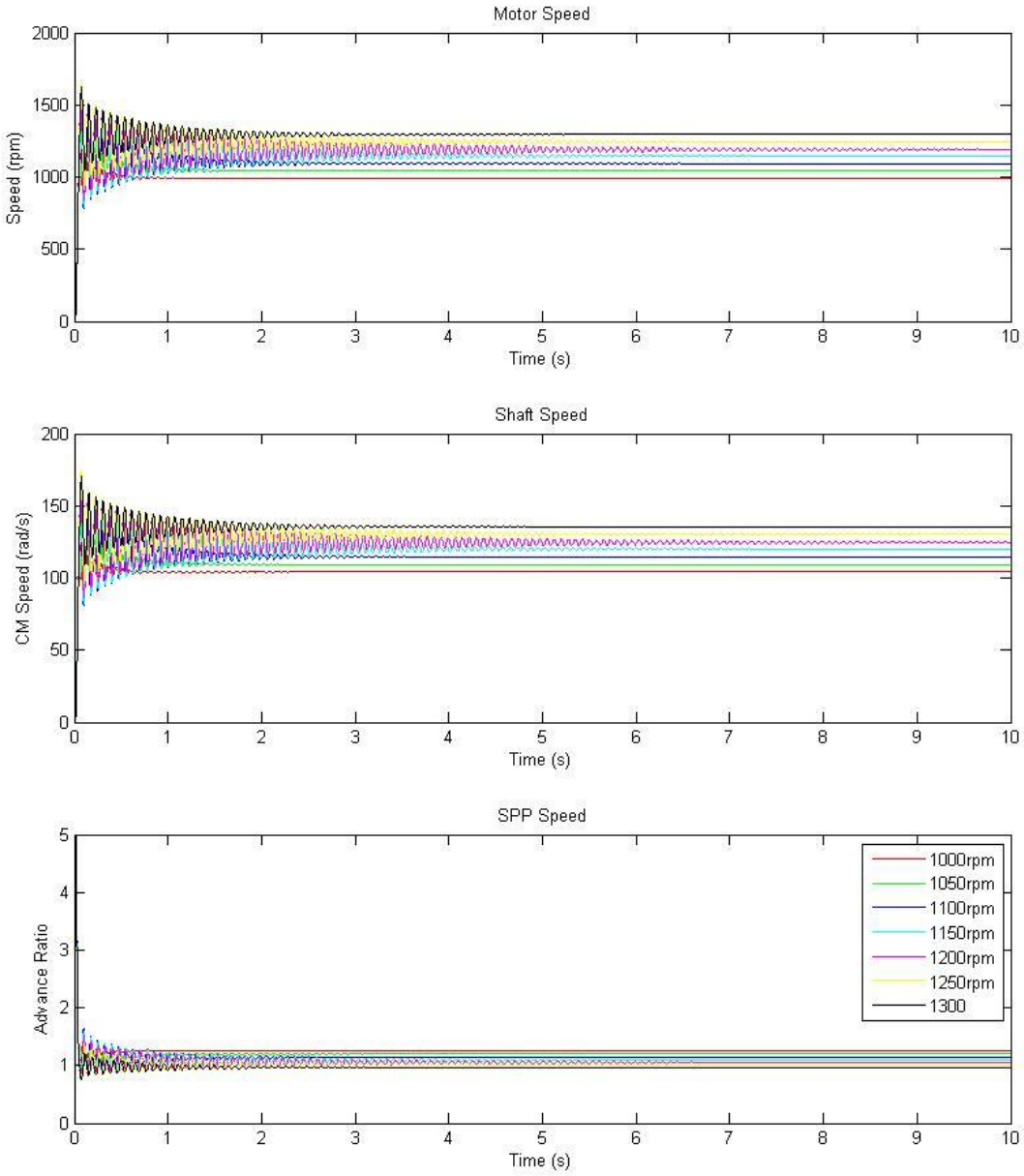

Figure 7-3: Constant set point simulation output of actual speed

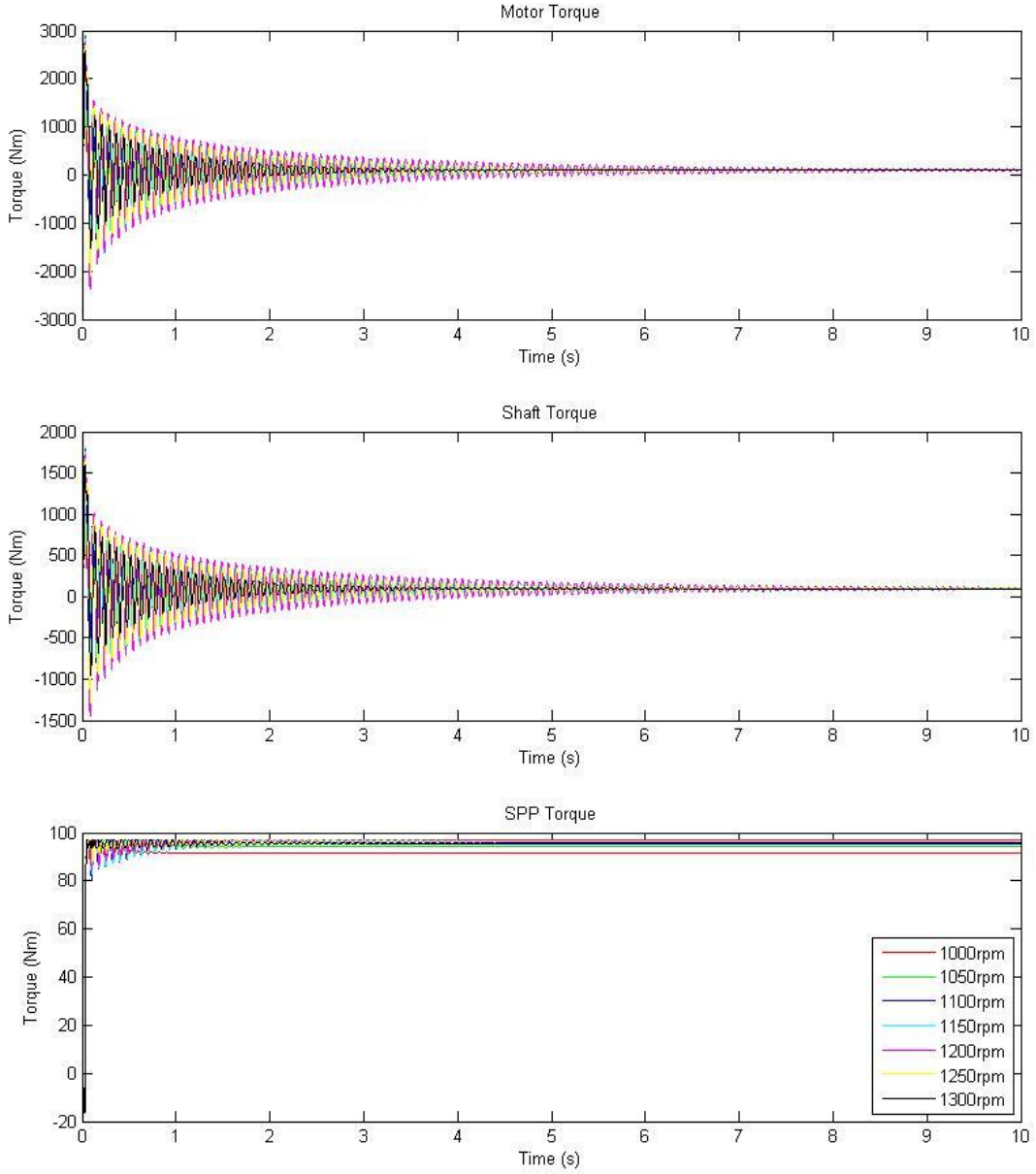

Figure 7-4: Constant set point simulation output of torque

7.4 Vibrational Output of Model

Most commonly, vibrations are observed as the oscillatory deviation of the position coordinate relative to the position it is expected to be in during the steady state. In the case of a rotating shaft, it is convenient to observe the shaft speed instead of the shaft position. As the model

developed can be easily configured to output the shaft's speed, angular deformation, or acceleration, the choice of output should be based on the problem of interest. Controlling the system through the entire speed range is of primary importance to the marine engineer intending to implement a propulsion plant. For that reason, this discussion is focused on oscillatory deviations in shaft speed. The output of the model as simulated in the present text was chosen to be the rotational speed of the motor, that of the propulsion shaft, and the advance ratio of the propeller. By considering these three speeds, vibrations can easily be seen and understood. As torque loads on the induction motor are creating the demand for additional power input to the induction motor, electromagnetic torque, shaft torque (to the induction motor), and propeller torque (to the shaft) were also selected as a set of outputs. The model was configured to display these speeds and torques and to record the data to the Matlab workspace for plotting and manipulation as needed. With the shaft resonance existence verified and the vibrational outputs selected, the shaft resonance and control of the system with its existence was explored in sections 7.4.2 and 7.5.

7.4.2 Shaft Resonance Excitation

As discussed in section 7.3, the shaft has a theoretical critical speed of 1244 rpm and the model predicts an actual excitation speed of closer to 1200 rpm. Reasons for the difference include the additional inertia of the AC induction motor's rotor and the nature of the forcing function driven by the magnetic motive force of the AC induction motor not considered in the theoretical calculation. With the presence of the shaft resonance in the middle of the operational speed envelope for the propulsion plant, control of this resonance is of great importance. To address controlling a shaft resonance without explicitly knowing its source, two concepts readily come to mind:

1. Re-design the physical parameters of the shaft to shift the resonant excitation frequencies and thereby move the critical speeds outside the operational envelope.
2. Implement an electric motor control system such as the already established, simple PID controller, or a more complex active cancellation system such as in-revolution torque control.

Following the re-design method, parameters were varied with an attempt to maintain a simple design and in order to shift the resonance to a critical speed of 2549 rpm, just above the operational speed envelope, the shaft would need to be modified from a hollow 8-cm diameter shaft to a solid 11 cm diameter shaft, increasing the weight from 140 kg to 342 kg. Clearly, this solution of the problem by simply increasing the strength of the shaft is not ideal. Modification of shaft bearing locations could also be attempted and was investigated as well. The addition of two more shaft bearings centered between the existing bearings seems to have little effect on the shaft resonance. Similarly, the increase in bearing frictional loss per bearing does not change the shaft resonance frequencies in a measureable way. This is due to the large difference in magnitude of the shaft stiffness and shaft frictional losses, making the resonance dominated by shaft stiffness. As the vibrations being observed are torsional in nature, a bearing provides little loss (in proper accord with its intended design function), and it makes sense that it would produce little effect on changing shaft torsional resonant frequencies. Although not considered in the present effort, non-torsional vibrations are significantly affected by bearing placement as they change the effective length of shaft between restrained points (bearing locations).

To establish a more elegant solution to the vibration control problem, electric motor control was implemented. First the PID control was activated to see if it could control the resonance of the

system. The system was operated at 800, 1200, and 2400 rpm bounding the operational envelope and exploring the critical speed directly using no PID control and using the PID control with established tuning parameters found in section 7.3.1. The Matlab script VibrationControlPlotterV1.m was written and executed to simulate operation of the system at the above mentioned three constant speed set points and generate vibrational output with and without PID control. Figures 7-5 and 7-6 plot the resulting vibrational output for the speeds and torques of the system, respectively. As the plots make clear, the PID control will dampen the oscillation and reduce the vibrational output of the system, but slows convergence to the set point speed. As the system must control a marine propulsion plant, long-term stability is important since the plant may be operated at cruise speeds for extended durations. The open loop controller (the speed-frequency lookup table alone) provides long-term stability and fast convergence, but will not set speed accurately in all cases and will not do anything to control system vibration. The ideal control system should have the stability and speed of convergence provided by the open loop system with the vibration damping of the tuned PID controller. With the PID control system adequately addressing the control of the propulsion plant, other options were explored in search of increased performance. Recent developments in direct torque control drives offer promise for high-performance AC induction motor control and are commercially available [20]. The concept of in-revolution variation of torque via superposition of torsional torque on the mean motor torque can be applied based on the fact that the motor has a finite number of windings and therefore the torque is not constant. This concept will be explored in section 7.5 Stability of Control Systems.

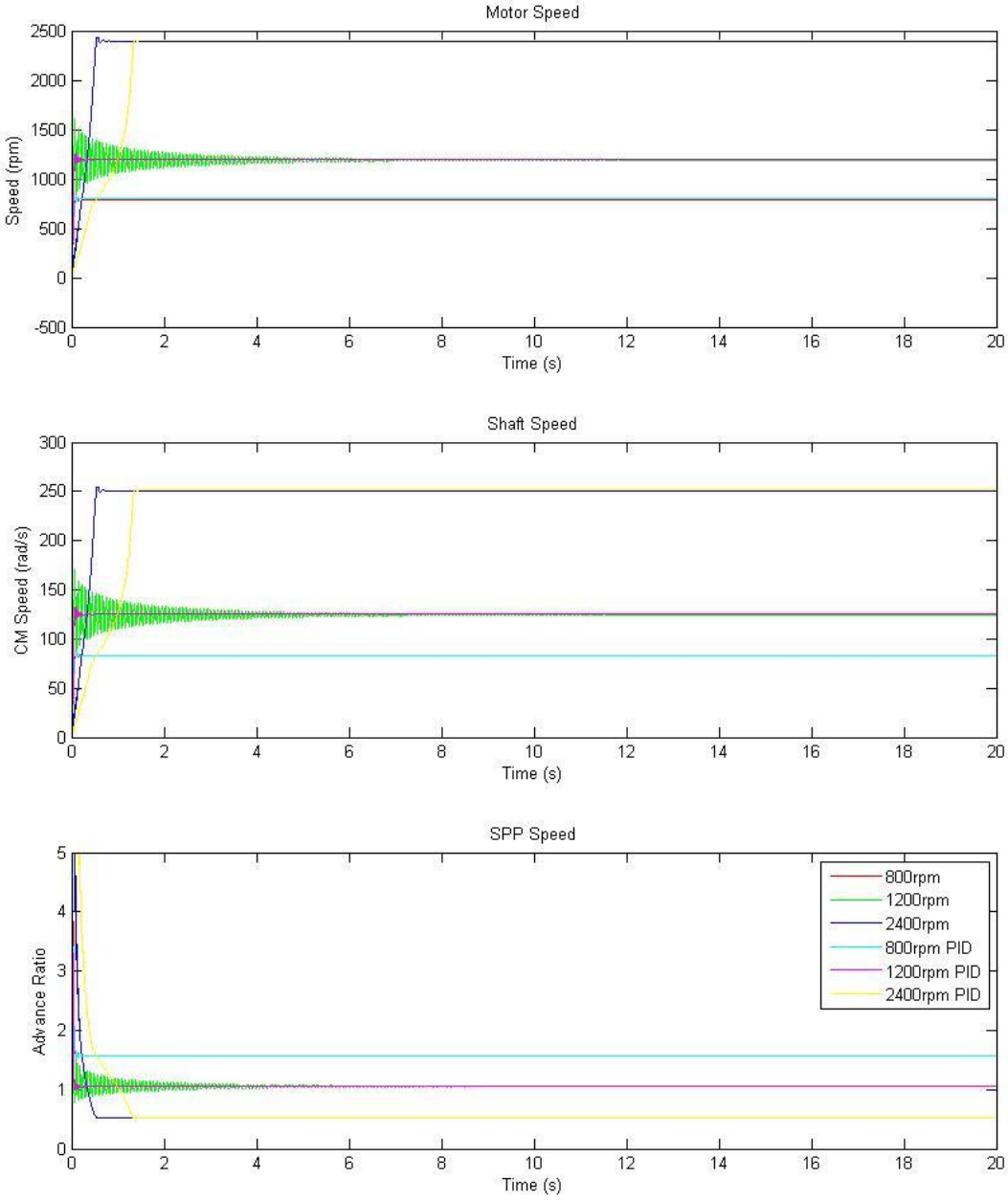

Figure 7-5: Vibrational speed output of propulsion plant with and without PID control

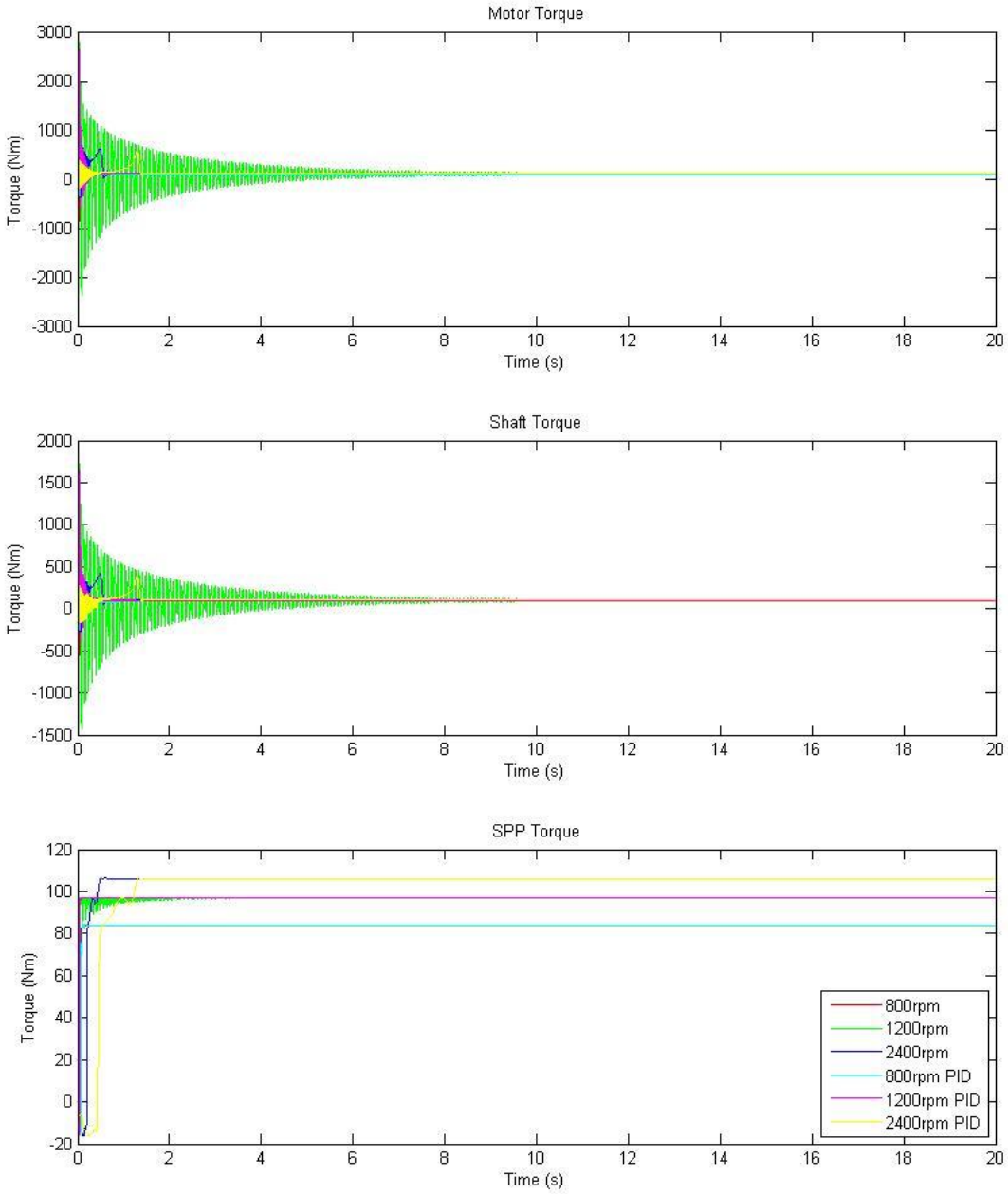

Figure 7-6: Vibrational torque output of propulsion plant with and without PID control

7.5 Stability of Control Systems

With the application of the PID control loop to the system producing stable control of vibrations with diminished speed of convergence, alternative motor speed control algorithms were

explored. Of interest was the elimination of torsional vibration through the utilization of variations in motor torque within a revolution of the motor based on the fact that the motor has a finite number of windings. To delve into the subject, one must first analyze the motor's instantaneous torque as a function of instantaneous current draw in each winding. Both of these quantities are available outputs of the integrated propulsion plant model via the asynchronous machine output de-multiplexing block. The functional concept of in-revolution torque control is to generate torque pulsations via input current control at specific times in order to actively cancel the vibration detected by the instantaneous electromagnetic torque output of the motor. If this can be achieved without degradation of the lookup table's speed of convergence, it may prove to be a better performing control scheme than the PID correction controller.

To analyze the torque versus current draw for each of the three windings on the motor a script called TorqueVcurrentPlotterV1.m was generated to plot the electromagnetic torque versus the individual stator currents at 800, 1200, and 2400 rpm. The plots enable the analysis of the relationship between the instantaneous torque and the instantaneous stator winding currents. Each speed was run separately to show the characteristic relationships of winding current and torque during resonance and steady state operation. The plots are presented as figures 7-7, 7-8, and 7-9 and show some instabilities in all three, but in figure 7-8, the resonance clearly shows torque fluctuations. As may be seen in the plots, stable torque output is reflected by oscillation of individual stator currents having no effect on the torque other than to maintain it. By ignoring the initial transients jutting back and forth in figures 7-7 and 7-9, one can see that stable torque output produces a vertical line on the plot at the appropriate torque load. Figure 7-8 clearly shows the torque wandering back and forth, but with a tendency to average to the expected

torque load, as seen in the higher density area of the plot near 100 Nm. It is this wandering that must be controlled in order to reduce or eliminate the shaft excitation.

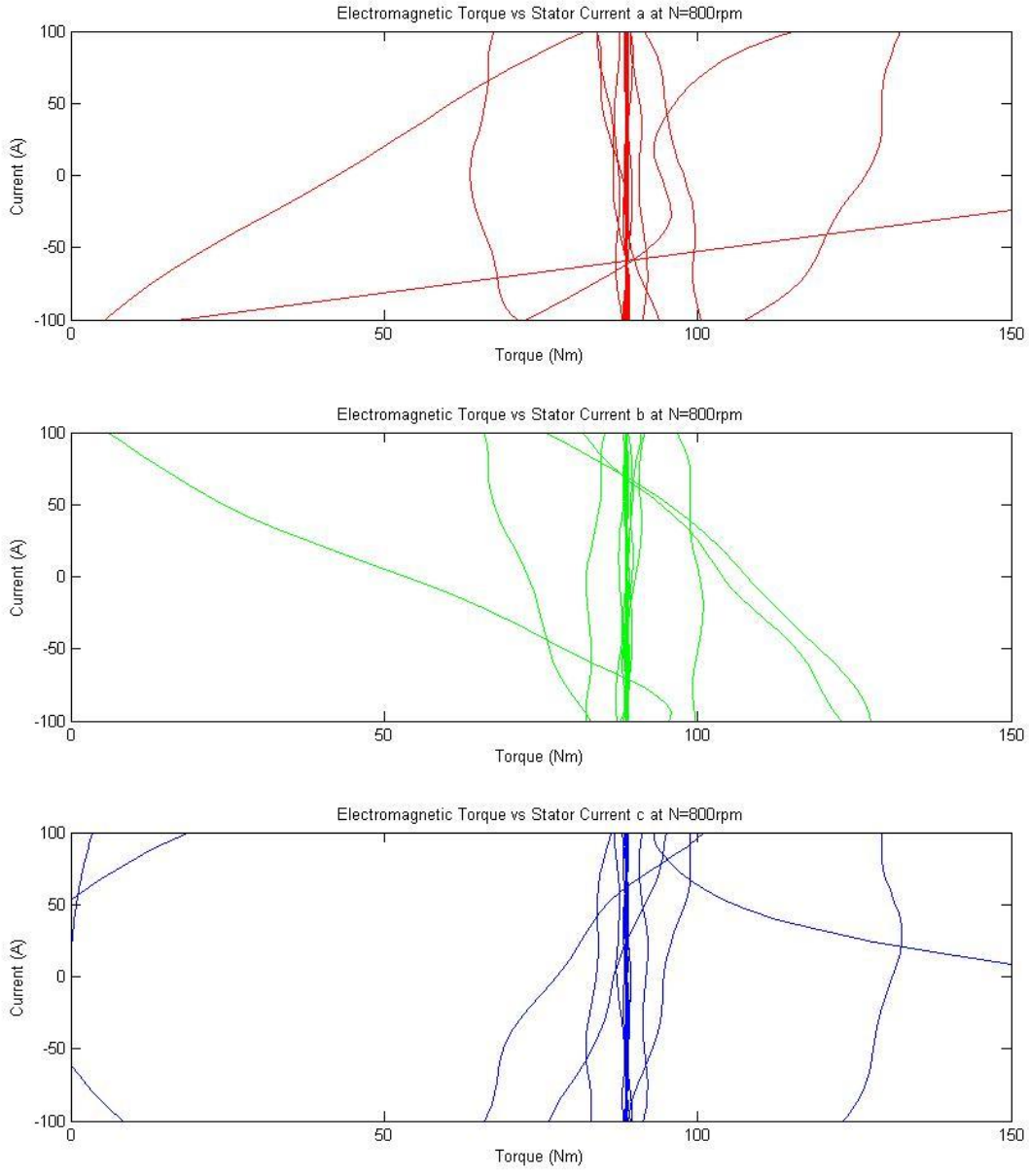

Figure 7-7: Torque-current relationship as seen on a per-winding basis at 800 rpm

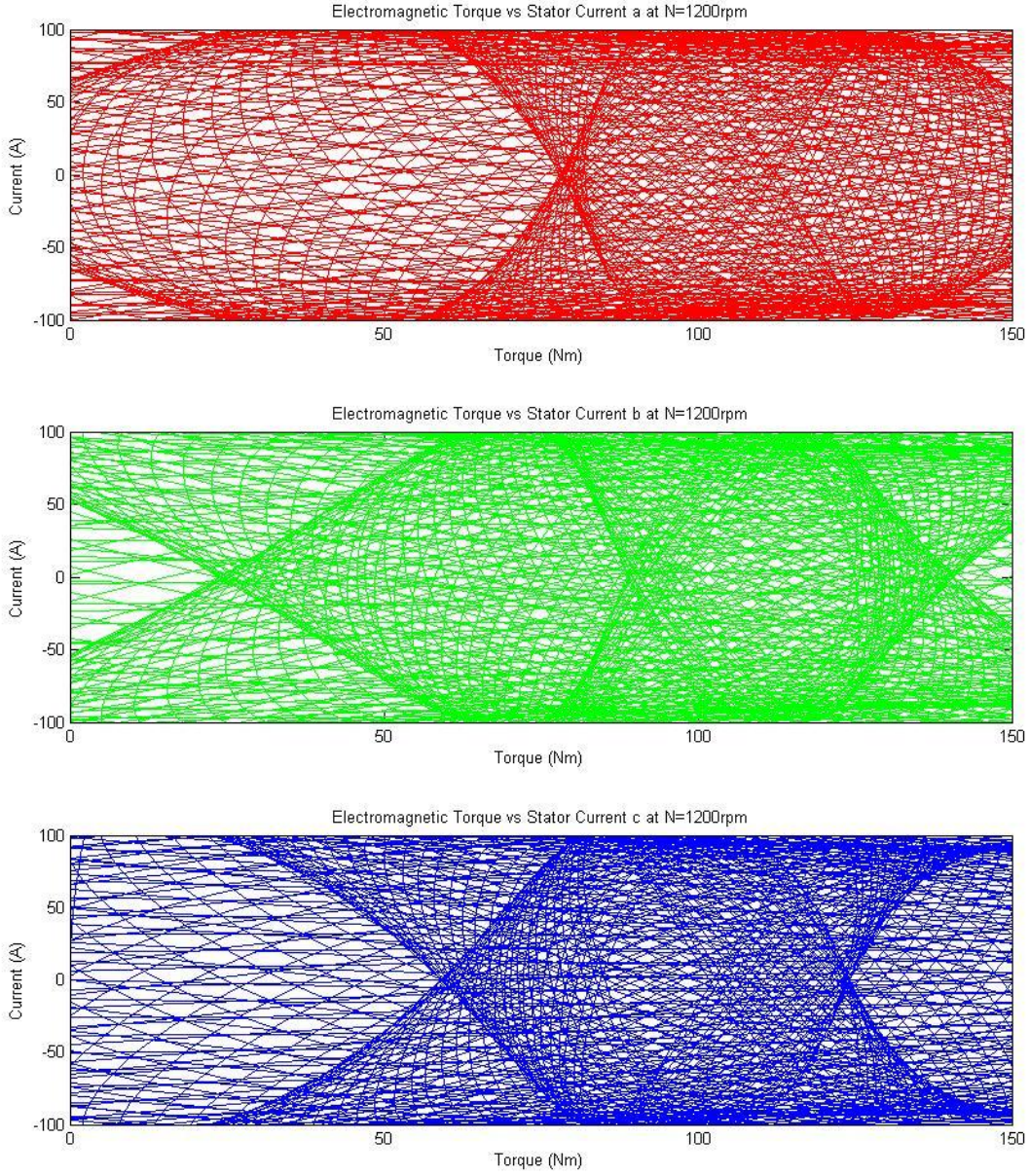

Figure 7-8: Torque-current relationship as seen on a per-winding basis at 1200 rpm

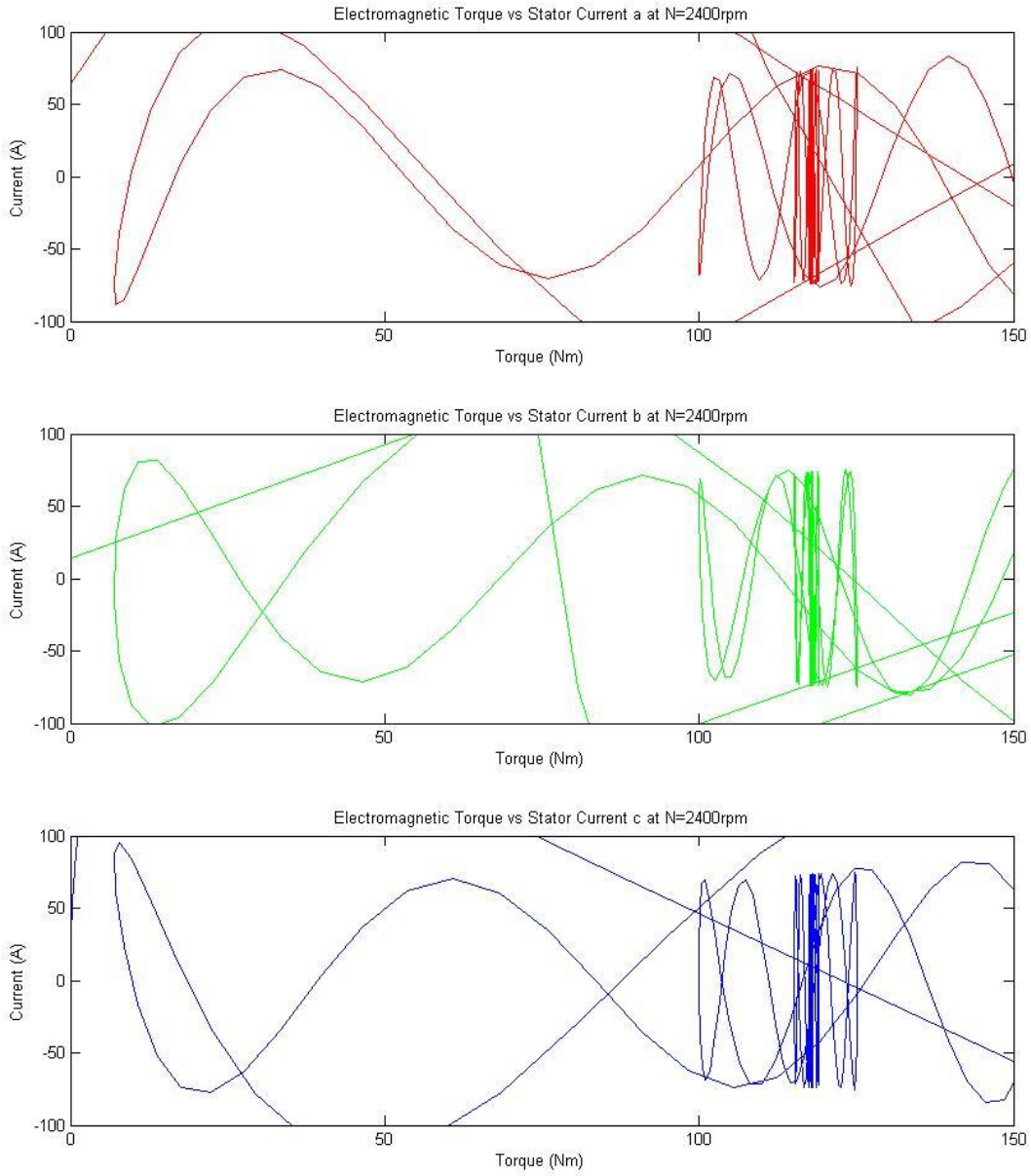

Figure 7-9: Torque-current relationship as seen on a per-winding basis at 2400 rpm

Two means of modifying the overall torque output come to mind. First, one might consider shifting the phase of the windings dynamically using power electronics, however, this will most likely cause damage to the stator as the current will become unbalanced. Another method would be individually controlling the current through each winding by installing a PWM inverter for

each winding, while maintaining the proper phase angles between them. Controlling the deviation of input frequency from one winding to the others becomes the primary focus with this method. Unfortunately, with the choice of DQ model for the motor, as stated in Chapter 4, the model will not support unbalanced loads and so firing one phase at a different frequency or driving it with a different current (and thereby making the input unbalanced), will not provide accurate results with the DQ model. For this reason, an alternative means of controlling the torque output as well as the speed output of the model leads to consideration of vector field control, capable of both. The concept of in-revolution torque control was not explored further as part of this research as the models developed do not support implementation of a control system utilizing this technique. Future research in this direction could potentially lead to better performing control of the system, but will require development of a motor model that can support such unbalanced loads.

With good success of the PID control system using basic Ziegler-Nichols tuning, manual fine tuning was carried out to with the goal of improving convergence speed. The gains were tuned manually by adjusting the proportional and integral gains up and down while paying attention to system torsional vibrations and speed of convergence. With the original Ziegler-Nichols parameters listed in Table 7-1, using the critical speed set point of 1200 rpm, the system stabilized torsional vibrations within approximately 0.5 seconds and reached a stable speed of 1192 rpm within approximately 0.6 seconds. With reduced integral gain, there is little effect on the control of the system, but when increased, torsional damping is reduced and the set point error is reduced. With reduced proportional gain, the vibration control increases and time to convergence decreases, while with increased proportional gain system instability ensues.

Excessive modification of any parameter can cause decreased performance and system instability. The final manually tuned proportional and integral gains obtained were Ki = 0.09, and Kp = 5.5. Using the manual tuning parameters, the stable speed output of the system was 1195 rpm. A set of plots were made for the constant 1200 rpm run with no PID control, with Ziegler-Nichols parameters, and with manual tuning parameters and are presented in figures 7-10 and 7-11. As can be seen in the plots, the output with manual tuning parameters exhibits the best torsional vibration control, the fastest convergence, and the smallest set point error making those tuning parameters the best selection. With the controller optimized for the system, one final simulation demonstrating the functionality of the model was conducted, with variation of the speed, pitch and yaw of the system operated in unison.

7.6 Demonstration of Vessel Maneuvering Simulation

A simple model was developed to operate the model under typical maneuvering operations to demonstrate the ability of the integrated model to be expanded and customized. To simulate the typical operation of a vessel accelerating, cruising, and turning, a signal builder block was set in Simulink and added to the simulation as a very simple maneuvering module. The maneuvering module consists of a set of signals that change in simulation time and a set of gains to convert the signals to set points in the propulsion plant model. The values of shaft speed set point, vessel speed, pitch, yaw, and immersion were varied simultaneously in a manner representative of ship acceleration as shown in figure 7-12. The simulation was operated for a total of 60 seconds and the system speed and torque outputs are displayed in figures 7-13 and 7-14, respectively. Under operation with the manually tuned PID gains, the system exhibited instabilities. By switching back to the Ziegler-Nichols PID gains, the system was able to operate with an initial transient during which the numerical oscillations cause poor model output.

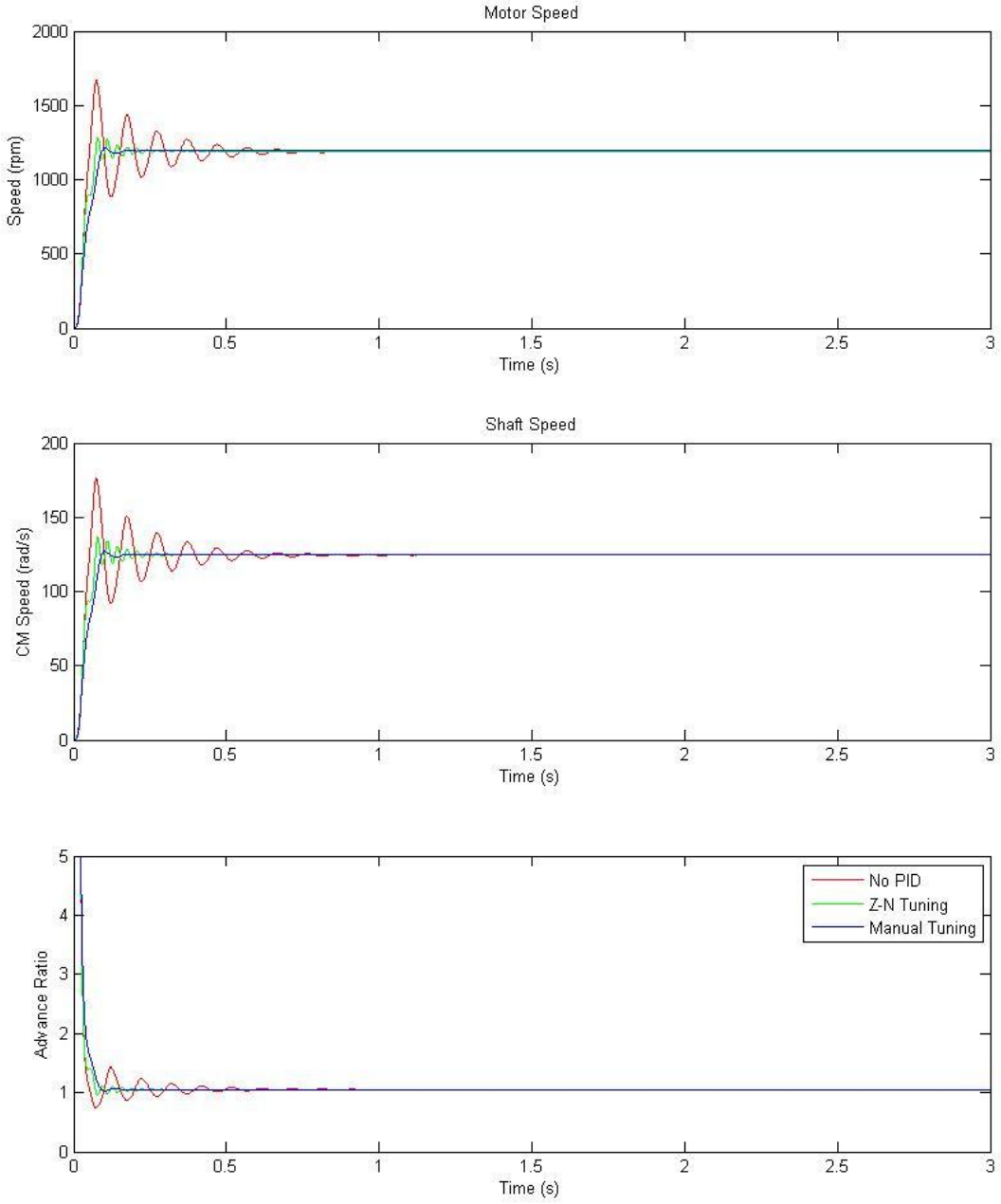

Figure 7-10: Speed output under various control tuning parameters

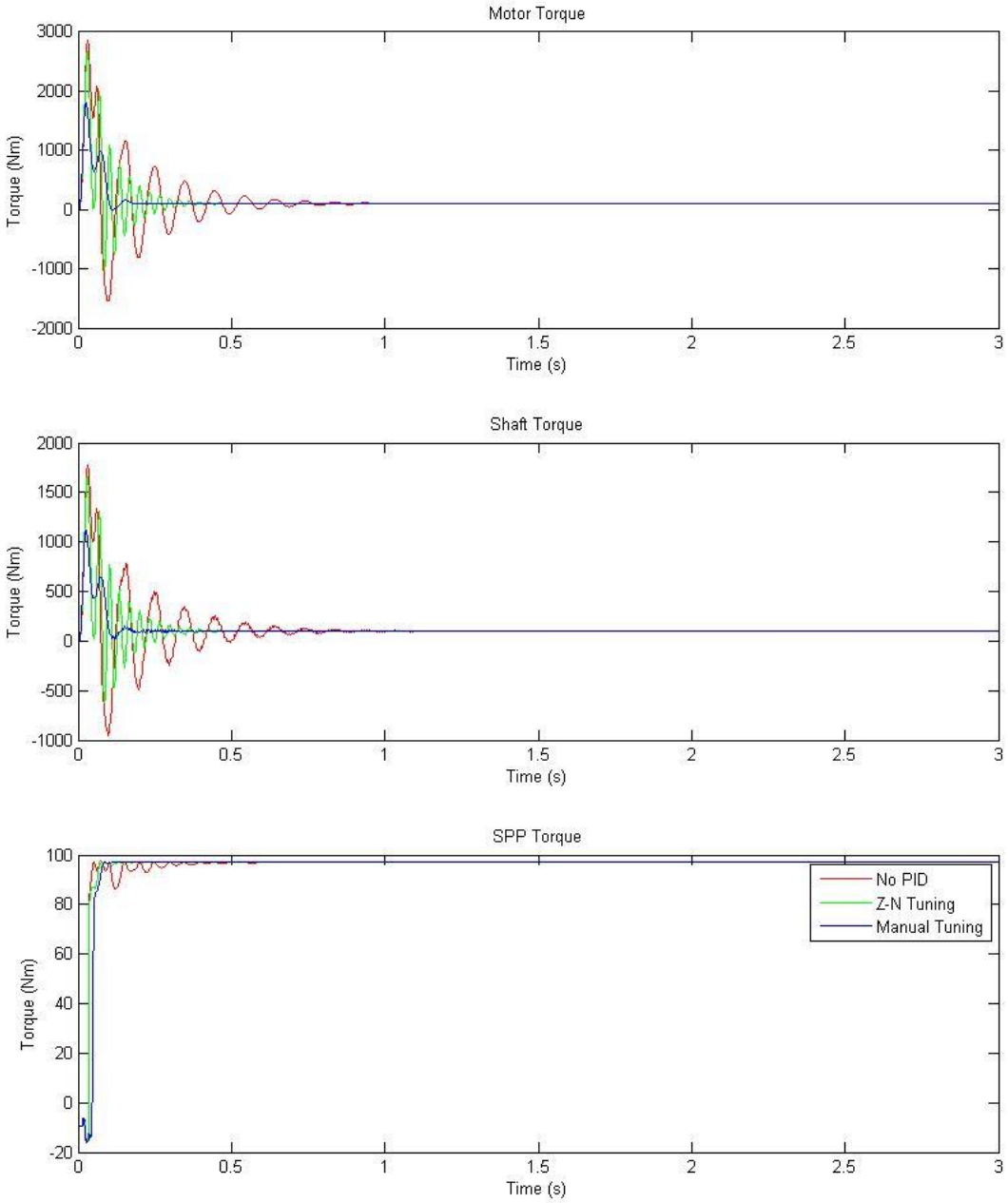

Figure 7-11: Torque output under various control tuning parameters

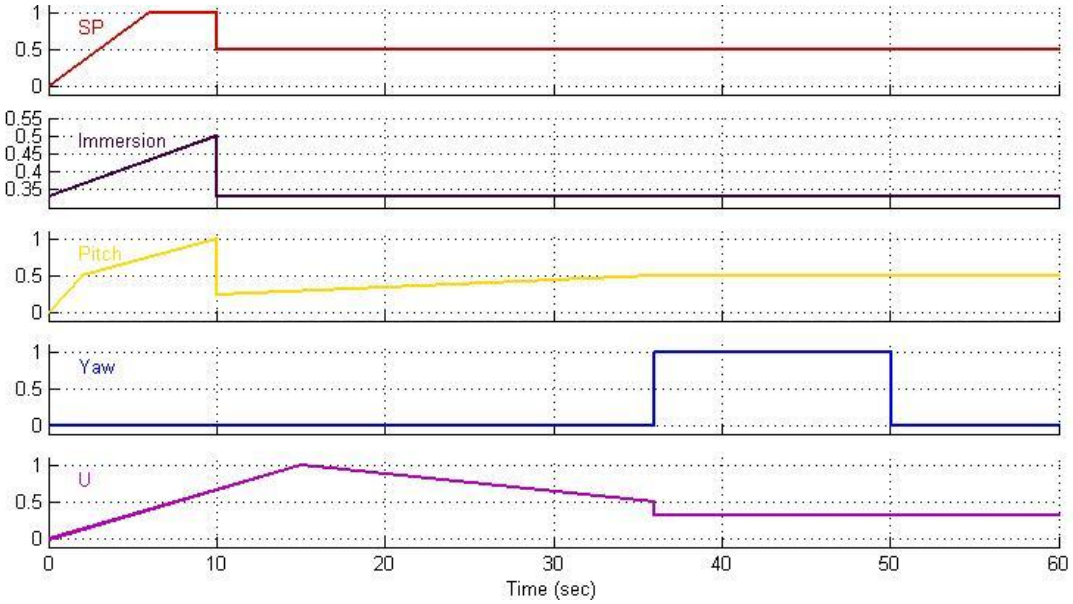

Figure 7-12: Maneuvering simulation parameter variation

After the initial transient, lasting the first 7 seconds of simulation time, the model become stable and produces very good output. This initial startup of the motor and acceleration of the vessel is an incredibly complex period for the propulsion plant and the investigation and more accurate modeling of such start up conditions should be considered for future research efforts. The output data clearly shows the step changes in yaw at 36 seconds, the deceleration of the vessel created, the large drop in torque, and even torque generation as well as the effects of shaft speed changes on the propulsor torque output. With a successful demonstration of the modular propulsion plant model utilizing the simple set of vessel behaviors depicted in figure 7-12, the research effort was concluded.

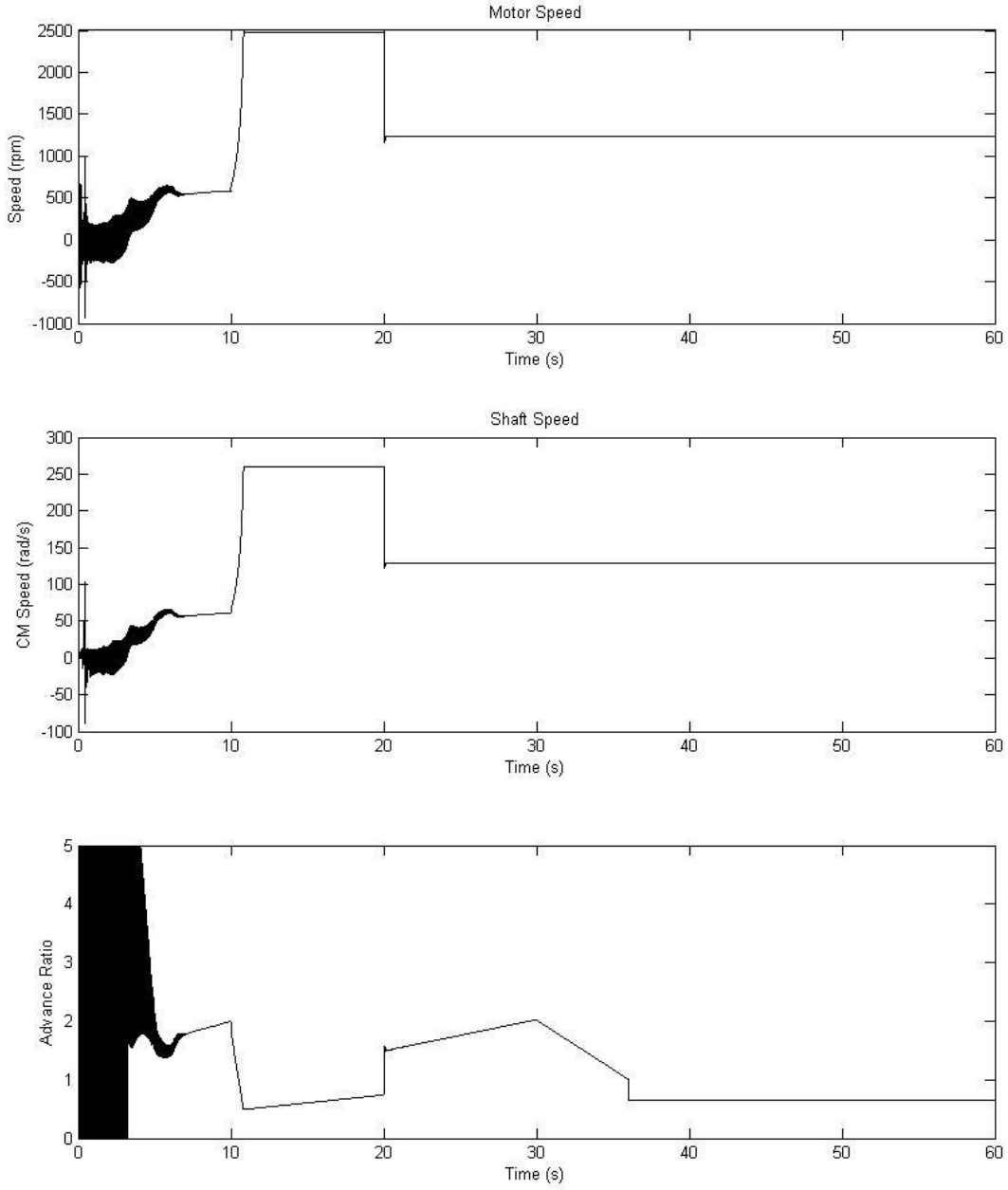

Figure 7-13: Maneuvering simulation model speed outputs

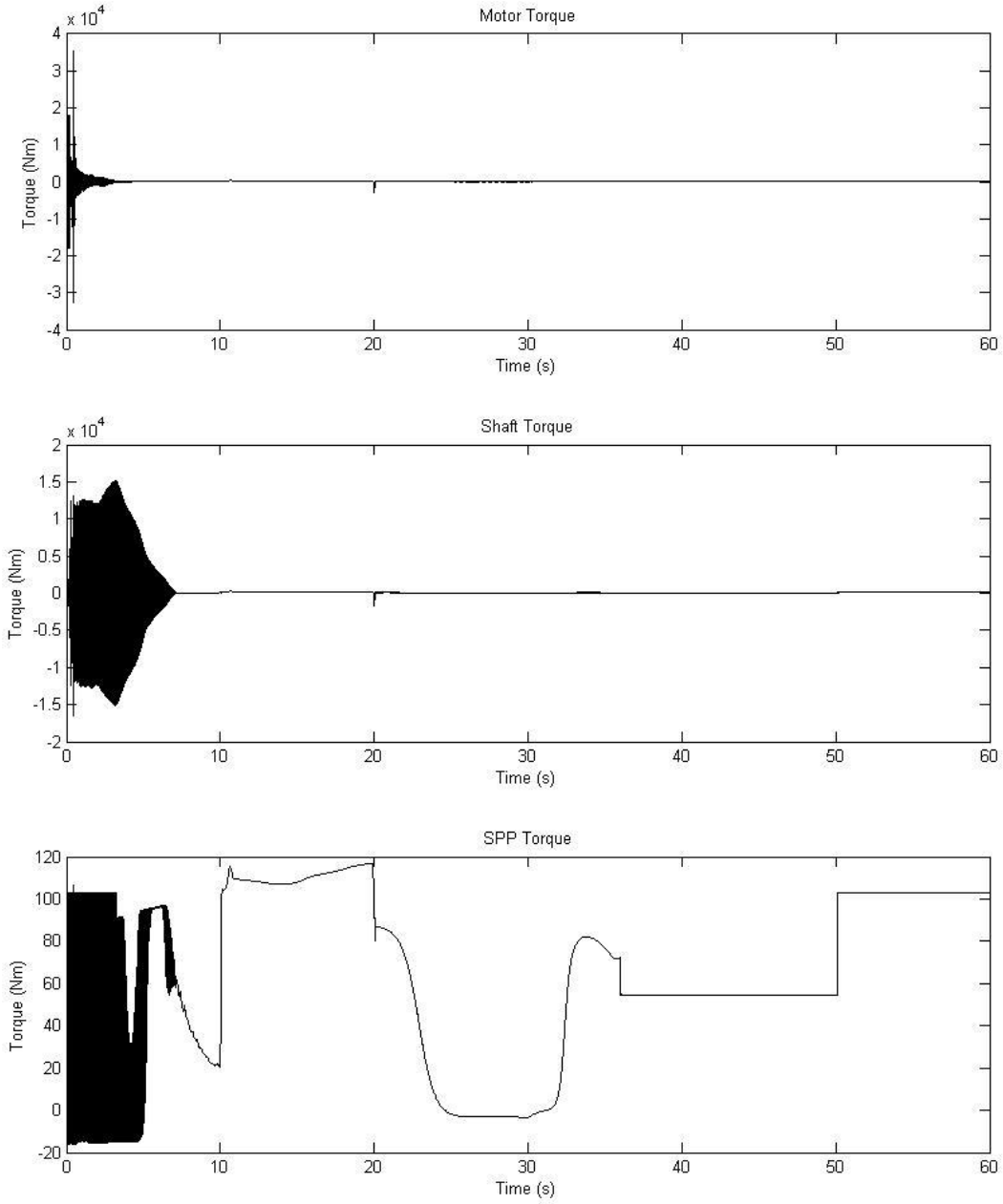

Figure 7-14: Maneuvering simulation model torque outputs

CHAPTER 8: Concluding Discussion

Through intensive literature search, evaluation of available models, reduction of available data, and generation of custom models, a marine propulsion plant simulation model was developed, validated, and operated for a small set of test cases. The model developed was intended to satisfy three research goals, reproduced below.

1. Develop a computational model of a marine propulsion plant that allows substitution of modular component sub-models.
2. Develop a methodology to translate measured propulsor data for non-conventional propulsors into a digital form that is continuous within the range of service of that propulsor, in order to build a propulsor sub-model.
3. Design and integrate an electric motor sub-model, a detailed propulsion shaft sub-model, and a surface piercing propeller sub-model to demonstrate via simulation the flexibility and utility of the simulation model.

The culmination of this work was gathered in the present text, revealing the details of how and why various aspects of the model development process were carried out.

8.1 Primary Accomplishments

The primary accomplishments of this research effort are the completion of a functional model of a marine propulsion plant, the development of a means to analyze experimental data and reduce it into an artificial neural network for application as a propulsor input-output model, and the integration of that model with a shafting model and a prime mover model, controlled by a simple PID controller. These accomplishments meet the goals of the research effort and provide a

functional design and analysis tool for the all electric drive marine propulsion plant. The model is a functional, validated, model of a small vessel plant consisting of a 50 HP electric induction motor, a 4 m long propulsion shaft, and a surface piercing propeller, applicable to a vessel on the order of 8 m long. A simple maneuvering module was established and connected to the model inputs. The model was operated under realistic vessel acceleration and steering maneuvers demonstrating its utility as both a design and a simulation tool.

By completing an in-depth literature search, coding numerous scripts, running many versions of models, and simulating a variety of operational situations, several important things were learned in the course of this research effort. Artificial neural networks are a powerful tool that can produce accurate models of systems for which the underlying physics is poorly understood. They are also a fickle tool that if programmed incorrectly can produce very poor results, demanding at least a basic understanding of what the result should be in order to select a good network. Active vibration control methods are vast when implemented via electric motor control. There are numerous different control schemes that can be implemented and the research carried out for this effort brought many to light. While advanced control schemes are possible, sometimes simple control, such as PID, can produce excellent results. It was further learned that initial transient behavior of numerical models, when combined with other models that provide data feedback, can increase the transient time by exciting the system of models with the initial transient. It was learned that because of this, one must take extreme care when combining models, as one model having a large amount of variation during a transient period can have large effects on the overall system of models. Of great importance to any research effort is learning what can be improved with future work.

8.2 Future Expansion of Model with Additional Modules

Future expansion of the model could add useful detail for sea trial type simulations. A human interface module could be developed that allows real-time modification of plant settings such as shaft speed, propeller yaw, propeller pitch, and propeller immersion ratio. Of interest would be a thrust output module running in parallel to the torque demand black box. It could be developed from the same data set with an artificial neural network tailored specifically to the thrust training data available. Additionally, a hull form module could be added that relates the thrust as a function of shaft speed, propeller yaw, propeller pitch, and immersion to the hull form drag, planing characteristics, and vessel speed. With these two additional modules, which interact with each other as well as the already established integrated propulsor plant model, the torque loading on the motor could be realistically modeled during vessel acceleration and deceleration. The modular nature of the modeling methodology allows for great flexibility in the expansion and customization of a model to a particular vessel. Future research efforts in continuation of the work presented herein could focus on design of these additional modules, investigation of the complex start up of the propulsion plant model, and improvement of the overall model control system to reduce error during system operational transients.

REFERENCES

1. Robinson, S. M., CDR, USN. Electric Ship Propulsion. New York: Simmons-Boardman Publishing Company, 1922.

2. Gottlieb, Irving. Electric Motors and Control Techniques. United States: TAB Books (McGraw-Hill), 1994.

3. Harrington, Roy L., ed. Marine Engineering. New Jersey: SNAME Press, 1992.

4. Sul, Seung-Ki. Control of Electric Machine Drive Systems. New Jersey: IEEE Press, 2011.

5. Lewis, Edward V. Principles of Naval Architecture Volume II: Resistance, Propulsion and Vibration. New Jersey: SNAME Press, 1988.

6. von Ellenrieder, Karl, et al. Open Water Towing Tank Testing of a Surface Piercing Propeller. The 29th American Towing Tank Conference, Annapolis, MD, August 2010.

7. Turing, Alan M. Intelligent Machinery. London: National Physical Laboratory, 1948.

8. Xiros, N. I. Robust Control of Diesel Ship Propulsion. London: Springer, 2002.

9. Copeland, B. Jack, and D Proudfoot. On Alan Turing's Anticipation of Connectionism.

10. Smith, Steven W. The Scientist and Engineer's Guide to Digital Signal Processing. Copyright 1997-1998. 10 April, 2011 <http://www.dspguide.com/ch26/2.htm>.

11. Fraser, Neil. Training Neural Networks. 21 September, 1998. 10 April, 2011. <http://vv.carleton.ca/~neil/neural/neuron-d.html>.

12. Pillay, P and V. Levin. Mathematical Models for Induction Machines. Thirtieth IAS Annual Meeting, Orlando, FL, October 1995.

13. The MathWorks, Inc. Matlab 2010a Help File. Natick, MA: The MathWorks, Inc., 2010

14. Krause, P. and C. Thomas. Simulation of Symmetrical Induction Machinery. IEEE Transactions PAS-84, 1965.

15. Marino, Riccardo, P. Tomei, and C. M. Verrelli. Induction Motor Control Design. London: Springer-Verlag, 2010.

16. WEG Electric Corp. WEG e-Technical Catalog. © 2011. 26 April 2011. <http://www.weg.com.br/catalog/index.asp?p_Categoria=28&p_Produto=394&p_Table=YES>

17. CATERPILLAR MARINE POWER SYSTEMS. 3056. June, 2008. 26 April, 2011. <http://marine.cat.com/cat-3056>

18. Roa, Camilo Carlos. Electric Motor Control with Application to Marine Propulsion. M.S. Thesis, Department of Ocean and Mechanical Engineering, Florida Atlantic University, Boca Raton, FL, 2010.

19. Ziegler, J. G. and N. B. Nichols. "Optimum Settings for Automatic Controllers." ASME, 1942.

20. ABB Oy. Technical Guide No.1 – Direct Torque Control. Helsinki: ABB Oy, 2002.

APPENDIX A: Source Code

AnalyzeDataZZ.m

AnalyzeDataZZ.m reads in the .txt files generated during data collection [6] and converts the data to Matlab .mat files for use in data processing. It also filters the data for cross talk elimination. The program was adapted from that used by J. Lorio in the data processing performed for reference [6].

```
clear
clc
%NUMPOINTS = 10e3;

% Physical Constants
g = 9.81;                       % gravity [m/s^2]

% Physical Properties
rho = 998.1;                    % Density Freshwater [kg/m^3]

% Multiplicative Conversion Factors
lbf2N = 4.4482216;
ft2m  = 0.30480;
in2m  = 0.0254;

% Prop Dimensions
D = 9.7*in2m;                   % Prop Diameter [m]

% Slip Ring Voltage Outputs
PropPosition = 1/0.27;          % [Deg./Volt] - may want to change mean
position (Unipolar +10V)
                                % so that 0 corresponds to top dead center
PropSpeed = 2000/10/60;         % [Hz/Volt] Prop speed voltage to rotation
rate (Bipolar +/- 10V)
CarriageSpeed = 3*ft2m;         % [m/sec/Volt] - May be revised based on
initial measurements

% Force Transducer Conversions
Vout = (rand(1,6)-0.5)*20;      % Random Test F/T Voltage Outputs [Volts]
Vexcite = ones(1,6)*10;         % Excitation Voltage 10 [Volts] (All
Channels)
Gain = ones(1,6)*1.0e3;         % Gains (1000 All Channels)
CF = Gain.*Vexcite*1e-6;

% Method 1 - single channel calibrations
% Sensitivities [microVolts/Voltexc-N]
S = [
    0.3556
    0.3563
    0.0880
    20.6308
```

```
    20.6893
    13.9781
    ]';
FM = (Vout./S./CF)';

% Method 2
% Inverse Sensitivity Matrix - for cancelling cross talk
B = [
     2.8201   0.0151 -0.0681   0.0076   0.0056   0.0035;
    -0.0363   2.8152 -0.0180 -0.0063   0.0121   0.0116;
     0.0409 -0.0468 11.3704   0.0065   0.0149 -0.0352;
    -0.0002   0.0017  0.0011   0.0484   0.0001 -0.0004;
     0.0018 -0.0007  0.0051 -0.0003   0.0483 -0.0006;
     0.0014 -0.0010  0.0004   0.0005   0.0005   0.0715;
    ];

% Coordinate Transformations (for calculating velocity component along shaft
axis)
dir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Text Files\';
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\MAT Files ZZ\'
files = dir(sprintf('%s*', dir_str));

%Use size of return to specify how many files will be processed.
num = size(files,1);

for i = 1:num-2,
    filename = files(i+2).name;
    fid = fopen([dir_str filename], 'r');
    filename = strrep(filename, '.', '');
    fdata = textscan(fid, '%f %f %f %f %f %f %f %f %f', 'headerlines', 7);

    fclose(fid);

    Phi = fdata{1};
    RPM = fdata{2};
    Fx = fdata{3};
    Fy = fdata{4};
    Thrust = fdata{5};
    Mx = fdata{6};
    My = fdata{7};
    Torque = fdata{8};
    Speed = fdata{9};

    save([sdir_str filename], 'Phi', 'RPM', 'Fx', 'Fy', 'Thrust', 'Mx', 'My',
'Torque', 'Speed')
end
```

Plot_Analysis.m

Plot_Analysis.m plots a set of all the data converted to .mat format so it may be reviewed to determine the best sections of the data. The script was adapted from one used by J. Lorio in the processing of data for reference [6]. The output is a set of .jpeg image files that may be reviewed and the steady state data extracted for conversion and eventual use in artificial neural network development.

```
clear
clc

dir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\MAT Files ZZ\';
files = dir(sprintf('%s*.mat', dir_str));

%Use size of return to specify how many files will be processed.
num = size(files,1);

for i = 1:num,

    load ([dir_str files(i).name])

    subplot(411)
    plot(RPM)
    title(files(i).name)
    ylabel('RPM')
    subplot(412)
    plot(Thrust)
    ylabel('Thrust')
    subplot(413)
    plot(Torque)
    ylabel('Torque')
    subplot(414)
    plot(Speed)
    ylabel('Speed')

    print  (['D:\Zack\SPP\SPP DATA\PlotsZZ\Plots\' strrep(files(i).name,
'.mat', '') '.jpg'], '-djpeg')

end
```

Volt_Conversion.m

Volt_Conversion.m was also adapted from the scripts generated by J. Lorio in support of reference [6] and reads in the Matlab .mat files, converts the voltage signals to measured values and saves the data as test sets in structure form for later use in artificial neural network development and training.

```
clear
clc

% Physical Constants
g = 9.81;                       % gravity [m/s^2]

% Physical Properties
rho = 998.1;                    % Density Freshwater [kg/m^3]

% Multiplicative Conversion Factors
lbf2N = 4.4482216;
ft2m  = 0.30480;
in2m  = 0.0254;

% Prop Dimensions
D = 9.7*in2m;                   % Prop Diameter [m]

dir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\MAT Files ZZ\';
files = dir(sprintf('%s*.mat', dir_str));
sdir1_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Converted Data ZZ\';
sdir2_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Converted Data Cross
ZZ\';

%Use size of return to specify how many files will be processed.
num = size(files,1);

for i = 7:num,

    load ([dir_str files(i).name]); %Bringing in the .mat files
    filename = files(i).name;

    fprintf('\t\tConverting file %d of %d\n', i, num)

    % Slip Ring Voltage Outputs
    % Convert Raw Voltage to RPM
    n = V2n(RPM);  %from function V2n
    % Convert Raw Voltage to Phi (Prop Position)
    Phi = V2Pos(Phi);
    % Convert Raw Voltage to Speed (U m/s)
    U = V2Speed(Speed);

    % Force Transducer Conversions
    Vout.Fx = Fx;
```

```
    Vout.Fy = Fy;
    Vout.Thrust = Thrust;
    Vout.Mx = Mx;
    Vout.My = My;
    Vout.Torque = Torque;           % Random Test F/T Voltage Outputs [Volts]


    %     ForceTorque = V2F(Vout);
    %     ForceTorque.Phi = Phi;
    %     ForceTorque.n = n;
    %     ForceTorque.U = U;
    %     save([sdir1_str filename], 'ForceTorque')

%     ForceTorque_Cross = V2F_Cross(Vout);
    ForceTorque_Cross = V2F(Vout);

    if(strcmp(filename,'H33P00Y00J0P8_Run7_BalancedCarriage.mat')),
        % Remove mean offsets from force and torque data
        Ln = length(n);
        n = n - mean(n(1:1e3));

        LThrust = length(ForceTorque_Cross.Thrust);
        ForceTorque_Cross.Thrust = ForceTorque_Cross.Thrust -
mean(ForceTorque_Cross.Thrust(1:1e3));

        LTorque = length(ForceTorque_Cross.Torque);
        ForceTorque_Cross.Torque = ForceTorque_Cross.Torque -
mean(ForceTorque_Cross.Torque(1:1e3));
    elseif(strcmp(filename, 'H33P00Y00J1P0_Run1.mat')),
        % Remove mean offsets from force and torque data
        Ln = length(n);
        n = n - mean(n(1:1e3));

        LThrust = length(ForceTorque_Cross.Thrust);
        ForceTorque_Cross.Thrust = ForceTorque_Cross.Thrust -
mean(ForceTorque_Cross.Thrust(1:1e3));

        LTorque = length(ForceTorque_Cross.Torque);
        ForceTorque_Cross.Torque = ForceTorque_Cross.Torque -
mean(ForceTorque_Cross.Torque(1:1e3));
    elseif(strcmp(filename, 'H33P00Y00J1P4_Run5.mat')),
        % Remove mean offsets from force and torque data
        Ln = length(n);
        n = n - mean(n(1:1e3));

        LThrust = length(ForceTorque_Cross.Thrust);
        ForceTorque_Cross.Thrust = ForceTorque_Cross.Thrust -
mean(ForceTorque_Cross.Thrust(1:1e3));

        LTorque = length(ForceTorque_Cross.Torque);
        ForceTorque_Cross.Torque = ForceTorque_Cross.Torque -
mean(ForceTorque_Cross.Torque(1:1e3));
    else
        % Remove mean offsets from force and torque data
        Ln = length(n);
        n = n - mean(n(1:1e3));
```

```
        LThrust = length(ForceTorque_Cross.Thrust);
        ForceTorque_Cross.Thrust = ForceTorque_Cross.Thrust -
mean(ForceTorque_Cross.Thrust(LThrust-1e3:LThrust));

        LTorque = length(ForceTorque_Cross.Torque);
        ForceTorque_Cross.Torque = ForceTorque_Cross.Torque -
mean(ForceTorque_Cross.Torque(LTorque-1e3:LTorque));
    end

    ForceTorque_Cross.Phi = Phi;
    ForceTorque_Cross.n = n;
    ForceTorque_Cross.U = U;
    save([sdir2_str filename], 'ForceTorque_Cross')

end

return
```

PlotVDR.m

PlotVDR.m was the last script adapted from J. Lorio's work in support of reference [6]. This script reads in the data structures created by Volt_Conversion.m and uses the file ValidDataRanges_reduced_ZZ.xls to extract only the steady-state data from each structure and re-write the data to new structures for use in the NeuralNetBuilderV5.m script.

```
%Plot valid data ranges only

clear all
clc

dir_str = 'D:\Zack\SPP\';
data_dir = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Converted Data Cross\';
[num, txt] = xlsread([dir_str 'ValidDataRanges_reduced_ZZ.xls']);
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Reduced Data MAT ZZ\';
spdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Reduced Data Plots
ZZ\';

NumFiles = length(num);
VDR = num;
DataRuns = txt;

for i=1:NumFiles
    filetoload = char(strcat(data_dir, DataRuns(i), '.mat'));
    filetosave = char(strcat(sdir_str, DataRuns(i), '.mat'));
    plottosave = char(strcat(spdir_str, DataRuns(i), '.jpg'));
    load(filetoload);

    start = VDR(i,1);
    stop = VDR(i,2);
    progU = VDR(i,3);
    clear REDUCED;

    for t=start:stop
        REDUCED.Fx(t-start+1,1) = ForceTorque_Cross.Fx(t);
        REDUCED.Fy(t-start+1,1) = ForceTorque_Cross.Fy(t);
        REDUCED.Mx(t-start+1,1) = ForceTorque_Cross.Mx(t);
        REDUCED.My(t-start+1,1) = ForceTorque_Cross.My(t);
        REDUCED.Phi(t-start+1,1) = ForceTorque_Cross.Phi(t);
        REDUCED.n(t-start+1,1) = ForceTorque_Cross.n(t);
        REDUCED.U(t-start+1,1) = ForceTorque_Cross.U(t);
        REDUCED.Thrust(t-start+1,1) = ForceTorque_Cross.Thrust(t);
        REDUCED.Torque(t-start+1,1) = ForceTorque_Cross.Torque(t);
    end

    %figure(i);
    subplot(411)
    plot(REDUCED.n)
    title(DataRuns(i))
```

```
    ylabel('RPM')
    subplot(412)
    plot(REDUCED.Thrust)
    ylabel('Thrust')
    subplot(413)
    plot(REDUCED.Torque)
    ylabel('Torque')
    subplot(414)
    plot(REDUCED.U)
    ylabel('Speed')

    print ([plottosave], '-djpeg')
    save ([filetosave], 'REDUCED')

end
```

NeuralNetBuilderV5.m

NeuralNetBuilderV5.m was developed specifically to generate artificial neural networks representing torque demand for a given set of inputs (vessel speed, shaft rotational speed, SPP pitch, SPP yaw, and SPP immersion). The script may be adapted to generate artificial neural networks for any desired output recorded during the data collection in support of the research effort of reference [6], however, as written, the script is tuned for the torque demand data only. Network specification parameters must be modified and tuned appropriately to generate good networks for other outputs.

```
clear all
clc

%Define file locations
dir_str = 'D:\Zack\SPP\';
data_dir = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Lookup Tables ZZ\';
train_data_dir = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Reduced Data MAT
ZZ\';
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Lookup Tables ZZ\';

Input_file = char(strcat(data_dir, 'BBLookupTableV1.mat'));
filetosave = char(strcat(sdir_str, 'BBNeuralNetV1.mat'));

%Generate training function data sets
[num, txt] = xlsread([dir_str 'ValidDataRanges_reduced_ZZ.xls']);
DataRuns = txt; %pull filenames from xls file

for i=1:length(DataRuns)

    filetoload = char(strcat(train_data_dir, DataRuns(i), '.mat'));
    load(filetoload);

    %Extract experiment property locations from filename
    H = strfind(DataRuns{i}, 'H');
    P = strfind(DataRuns{i}, 'P');
    Y = strfind(DataRuns{i}, 'Y');
    J = strfind(DataRuns{i}, 'J');
    JP = P(2);
    P = P(1);

    %Generate experiment properties structure
    filename = DataRuns{i};
    DataProps.RunName(i,1) = DataRuns(i);
    DataProps.Immersion(i,1) = str2num(strcat(filename(H+1),filename(H+2)));
    DataProps.Pitch(i,1) = str2num(strcat(filename(P+1),filename(P+2)));
    DataProps.Yaw(i,1) = str2num(strcat(filename(Y+1),filename(Y+2)));
```

```
    DataProps.J(i,1) = str2num(strcat(filename(J+1), '.', filename(JP+1)));

    %Generate experiment average results to strcuture
    DataAvgRes.avgFx(i,1) = mean(REDUCED.Fx(i));
    DataAvgRes.avgFy(i,1) = mean(REDUCED.Fy(i));
    DataAvgRes.avgMx(i,1) = mean(REDUCED.Mx(i));
    DataAvgRes.avgMy(i,1) = mean(REDUCED.My(i));
    DataAvgRes.avgn(i,1) = mean(REDUCED.n(i));
    DataAvgRes.avgU(i,1) = mean(REDUCED.U(i));
    DataAvgRes.avgTorque(i,1) = mean(REDUCED.Torque(i));
    DataAvgRes.avgThrust(i,1) = mean(REDUCED.Thrust(i));

end

%Load the input and target data for training the network
P1 = DataProps.Immersion(:,1);
P2 = DataProps.Pitch(:,1);
for c=1:length(P2)
    if P2(c,:) == 75
        P2(c,:) = 7.5;
    end
end
P3 = DataProps.Yaw(:,1);
P4 = DataProps.J(:,1);
P = [P1';P2';P3';P4'];

T1 = DataAvgRes.avgTorque(:,1)';
T2 = DataAvgRes.avgThrust(:,1)';
T3 = DataAvgRes.avgFx(:,1)';
T4 = DataAvgRes.avgFy(:,1)';
T5 = DataAvgRes.avgMx(:,1)';
T6 = DataAvgRes.avgMy(:,1)';
T7 = DataAvgRes.avgn(:,1)';
T8 = DataAvgRes.avgU(:,1)';
T = T1;


%Define a neural network for the SPP model
maxlr = maxlinlr(P, 'bias');
SPPnet =
newff(P,T1,[130,9,1],{'logsig','logsig','purelin'},'traincgp','learngdm','mse
');

SPPnet.numInputs = 1;
SPPnet.numLayers = 3;
SPPnet.biasConnect = [1;1;1];
SPPnet.inputConnect = [1;1;0];
SPPnet.outputConnect = [0 0 1];
SPPnet.layerWeights{:,:}.learn = [1];
SPPnet.inputWeights{:,:}.learn = [1];
SPPnet.biases{:}.learn = [1];

%Train the network
SPPnet = init(SPPnet);
SPPnet.trainParam.epochs = 100000;
SPPnet.trainParam.min_grad = 1e-20;
SPPnet.trainParam.max_fail = 100000;
```

```
SPPnet.trainParam.minstep = 1.0e-30;
SPPnet = train(SPPnet, P, T);

%Test trained network
a = sim(SPPnet, P);
b = T-a;
c = abs(T - a)./T*100;
figure(1)
clf
plot (P4, a, 'x')
title('Simulated Targets')
figure(2)
clf
plot (P4, T, 'o')
title('Actual Targets')
figure(3)
clf
hold on
plot (P4, a, 'x')
plot (P4, T, 'o')
title('Simulated and Actual Targets Overlayed')
figure(4)
clf
hold on
plot (P4, b, '*')
plot (P4, c, 'o')
title('Target Error and Percent Error')

RankVal = mean(abs(b))
sim(SPPnet,[33;0;0;1.4])
maxerror = max(abs(c))
figure(5)
clf
[m,b,r] = postreg(a,T);
```

SPPTorqueDemandGenV3.m

SPPTorqueDemandGenV3.m uses select artificial neural networks developed using NeuralNetBuilderV5.m to generate speed-torque demand curves for the surface piercing propeller tested in reference [6]. It outputs a Matlab plot that may be manually saved or exported to .jpeg format.

```
%Torque Demand Curve Generator
%Runs SPPModuleV1 repeatedly at varying speed N to generate a T-N curve
clear all
clc

%Recall Network
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Lookup Tables ZZ
Corrected\';
%SPPnet = 'SPPNETe208traincgplearngdm_13091_111_111';  %.4377
SPPnet = 'SPPNETe253traincgplearngdm_13091_111_100';  %.5765
nettoload = char(strcat(sdir_str, SPPnet, '.mat'));
load(nettoload);

%Parameters held constant
U = 5; %Vessel Speed in knots
kt2ips = 20.2537183; %Convert knots to inches per second
Dspp = 9.7; %SPP diameter in inches
Im = 33; %Immersion ratio supported by the majority of test results
Pi = 0; %Pitch of SPP in degrees
Y = 0; %Yaw of SPP in degrees
Tfactor = -10; %Gain factor to accont for bad amplifier/load sensor during
SPP tests
precision = 100; %Number of data points desired
Maxrpm = 2500; %Maximum desired SPP rpm

%Cycle SPP through JP from 2.0 to .200
JP = zeros(precision,1);
NT = zeros(precision,2);
for c = 1:precision
    NT(c,1) = c*(Maxrpm/precision);
    JP(c,1) = U*kt2ips/(NT(c,1)/60*Dspp);
    NT(c,2) = sim(SPPnet,[Im;Pi;Y;JP(c)])*Tfactor;
end

plot(NT(:,1),NT(:,2), 'r'); %Plots speed versus torque
Tfl = max(NT(:,2)); %Outputs the Full Load Torque
Nfl = NT(84,1); %Outputs a speed value to run in SPPModuleV1 for comparison

hold on

%10kt line
U = 10;
%Cycle SPP through JP from 2.0 to .200
```

```
JP = zeros(precision,1);
NT = zeros(precision,2);
for c = 1:precision
    NT(c,1) = c*(Maxrpm/precision);
    JP(c,1) = U*kt2ips/(NT(c,1)/60*Dspp);
    NT(c,2) = sim(SPPnet,[Im;Pi;Y;JP(c)])*Tfactor;
end

plot(NT(:,1),NT(:,2), 'g'); %Plots speed versus torque
Tfl = max(NT(:,2)); %Outputs the Full Load Torque
Nfl = NT(84,1); %Outputs a speed value to run in SPPModuleV1 for comparison

%15kt line
U=15;
%Cycle SPP through JP from 2.0 to .200
JP = zeros(precision,1);
NT = zeros(precision,2);
for c = 1:precision
    NT(c,1) = c*(Maxrpm/precision);
    JP(c,1) = U*kt2ips/(NT(c,1)/60*Dspp);
    NT(c,2) = sim(SPPnet,[Im;Pi;Y;JP(c)])*Tfactor;
end

plot(NT(:,1),NT(:,2), 'b'); %Plots speed versus torque
Tfl = max(NT(:,2)); %Outputs the Full Load Torque
Nfl = NT(84,1); %Outputs a speed value to run in SPPModuleV1 for comparison

%20kt line
U =10;
%Cycle SPP through JP from 2.0 to .200
JP = zeros(precision,1);
NT = zeros(precision,2);
for c = 1:precision
    NT(c,1) = c*(Maxrpm/precision);
    JP(c,1) = U*kt2ips/(NT(c,1)/60*Dspp);
    NT(c,2) = sim(SPPnet,[Im;Pi;Y;JP(c)])*Tfactor;
end

plot(NT(:,1),NT(:,2), 'y'); %Plots speed versus torque
Tfl = max(NT(:,2)); %Outputs the Full Load Torque
Nfl = NT(84,1); %Outputs a speed value to run in SPPModuleV1 for comparison
legend('5kt', '10kt', '15kt', '20kt', 4);
title('SPP Speed Torque Curve Set');
xlabel('Shaft Rate (rpm)');
ylabel('Torque Demand (Nm)');
```

PropulsionSystemConfigParamsV5.m

PropulsionSystemConfigParamsV5.m establishes the Simulink model parameters required and places the variables in the workspace for reference by the models. This script was generated for use in all simulation models and is required to activate them. It is also required for activation of some of the simulation analysis scripts.

```
%Section I etablishes simulation configuration parameters
%Section II establishes computed values required by the simulation

clear all
clc

%SECTION I: Input Values

%Recall Network
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Lookup Tables ZZ
Corrected\';
SPPnet = 'SPPNETe253traincgplearngdm_13091_111_100';  %.5765
nettoload = char(strcat(sdir_str, SPPnet, '.mat'));
load(nettoload);
%gensim(SPPnet, -1); %Used only to generate the SPP block in simulink
%in gensim -1 creates a net that samples continously.

%Define propeller speed
rpm = 1932; %Commanded Shaft Speed (rpm)
N = rpm;
SP = 2400;

%Set Shaft Model Simulation Parameters
n = 10; %Number of shaft elements
w0 = 0; %Shaft integrator initial conditions for velocity
Le = [.15 .6 .6 .6 .1 .6 .6 .6 .15 .1]; %Shaft element lengths (m)
Lt = sum(Le); %Total shaft length
nb = [1 0 0 0 1 0 0 0 1 0]; %Location of shaft bearings (1st and n-1 element
required)
Lb = 120/(2300*pi/30)*.05/length(find(nb)); %Tmax/wmax (Nm/rad/s)
rho = 7850; %Shaft material density (kg/m^3)
G = 8.2e10; %shear modulus (Pa/rad)
D1 = [.08 .08 .08 .08 .08 .08 .08 .08 .08 .2464]; %Shaft element outer
diameters (m)
D2 = [.05 .05 .05 .05 .05 .05 .05 .05 .05 .00]; %Shaft element inner
diameters (m)
H = [1 1 1 1 1 1 1 1 1 0]; %Shaft element hollow (1) or solid (0)
Lf = .05; %Motor-shaft flange length (m)
Df = .15; %Motor-shaft flange diameter (m)

%Set Motor Model Simulation Parameters
%Load lookup table
```

```
load 'D:\Zack\Independent Model
Modules\LookupTables\50HP10ktI33P0Y0p100tol5step05wShaftJPP.mat'
Kp = 0.6*0.3;%Proportional Gain Setting (Ku/2.2 for PI)
Ki = 0.5*0.3/0.035*0.55;%Integral Gain Setting (Ku/Tu*0.55 for PI)
Kd = 0.075*0.3*0.035; %Derivative Gain Setting
PIDdelay = 0; %PI Control Start Delay (s)

%Set SPP Simulation Parameters
Tfactor = -10; %Torque multiplication factor due to bad amplifier during test
%Tfactor is negative because data values were negative for torque loads
Dspp = 9.7; %SPP diameter (in)
U = 10; %Vessel speed in knots
kt2ips = 20.2537183; %Convert knots to inches per second
Im = 33; %SPP immersion ratio
Y = 0; %SPP yaw angle
Pi = 0; %SPP pitch angle

%SECTION II: Computed Values

%Compute masses and moments of inertia
m = zeros(1,n); %(kg)
j = zeros(1,n); %(kg*m^2)
jp = zeros(1,n); %(m^4)
for c = 1:n
    if H(c) == 0 %Establish mass and moment for solid elements
        m(1,c) = rho*Le(c)*pi*(D1(c)/2)^2;
        j(1,c) = 1/8*m(c)*(D1(c))^2;
        jp(1,c) = 1/32*pi*(D1(c))^4;
    end
    if H(c) == 1 %Establish mass and moment for hollow elements
        m(1,c) = rho*Le(c)*pi*((D1(c)/2)^2 - (D2(c)/2)^2);
        j(1,c) = 1/8*m(c)*((D1(c))^2 + (D2(c))^2);
        jp(1,c) = 1/32*pi*((D1(c))^4 - (D2(c))^4);
    end
end
mf = rho*Lf*pi*(Df/2)^2;
jf = 1/8*mf*(Df)^2;

%Generate g
jt = sum(j);
g = j/jt;

%Populate inertia (J) matrix and torque inverter matrix Jout
J = zeros(n,n);
for c = 1:n
    J(c,c) = j(1,c);
end

%Populate stiffness (Kt) matrix
kt0 = G/Lf*pi/32*Df^4;%Motor-shaft coupling stiffness coefficient
kt = zeros(n-1,1); %Shaft element torsional spring constants
for c = 1:length(kt)
    kt(c) = G*jp(1,c)/Le(c);
end
Kt = zeros(n,n);
Kt(1,1) = kt0;
Kt(1,2) = -kt(1);
```

```
for c = 2:n-1
    Kt(c,c) = kt(c-1);
    Kt(c,c+1) = -kt(c);
end
Kt(n,n) = kt(n-1);

%Generate torque output conversion vector
Kt0 = [kt0, zeros(1,n-1)]; %Motor model expects a positive torque load

%Populate loss (B) matrix
B = zeros(n,n);
for c = 1:n %Establish shaft tunnel bearing losses
    if nb(c) == 1
        B(c,c) = Lb; %Unweighted
    end
    if nb(c) == 0
        B(c,c) = 0;
    end
end

%Generate element-spring coordinate transformation matrix (L)
L = zeros(n,n);
L(1,1) = -1;
for c = 2:n
    L(c,c-1) = 1;
    L(c,c) = -1;
end

%Generate input vectorizer
Ln = [1; zeros(n-1,1)]*pi/30;%N converted to rad/s

%Generate Torque and Speed output vectorizers
TtoVec = [zeros(1,n-1),1]';
PropSpeed = [zeros(1,n-1),1];
```

MotorCapabilityPlotterV1.m

MotorCapabilityPlotterV1.m automatically operates the model AC50HP_TorqueTestV1.mdl for various input frequencies to generate a plot of torque capability curves for the AC induction motor modeled in AC50HP_TorqueTestV1.mdl. This plot is used to demonstrate the ability to vary the speed of an AC induction motor by frequency modulation.

```
%Generate data plot representing variable frequency torque output
%Run PropulsionSystemConfigParamsV5.m to initialize the propulsion plant
%and load the needed variables into the workspace

Freqset = [400, 350, 300, 250] %Select Frequencies to test
colorvec = ['r'; 'g'; 'b'; 'c'; 'm'; 'y'; 'k'; 'r'; 'g'; 'b'; 'c'; 'm'; 'y';
'k'];
colorcounter = 1;
figure(1)
clf

PIDdelay = 999; %Turn off PID controller

%Run and plot figures for no PID control
for c = Freqset
    fe = c;
    sim('AC50HP_TorqueTestV1');
    color = colorvec(colorcounter,1);
    figure(1)
    plot(Nout(:,1), Teout(:,1), color);
    title('Motor Speed');
    xlabel('Speed (rpm)');
    ylabel('Torque (Nm)');
    hold on
    legend('400Hz', '350Hz', '300Hz', '250Hz', '200Hz', '150Hz', '100Hz', 1);
    xlim([0 4000]);
    ylim([-50 650]);
    colorcounter = colorcounter + 1;
end
```

VFDLookupTableGenV3.m

VFDLookupTableGenV3.m was developed to establish a lookup table for use in the integrated propulsion plant model. This script iteratively runs the purpose-built model AC50HP_PIControlModuleV1.mdl with the PI control turned off at varying input frequencies to match the output speed to the corresponding speed for a given torque read from the propeller torque demand curve generated using SPPTorqueDemandGenV3.m script. It will read each data point of the table, use the torque value as the load on the AC induction motor, and vary the frequency input until the output speed of the AC induction motor model matches the speed on the propeller torque demand curve for that point. Once the frequency is established, the value is saved to a Matlab array with the corresponding speed. The process is repeated for each data point on the propeller torque demand curve until a complete speed-frequency conversion table is developed for the motor. The data recorded must be post-processed using LookupTablePostProcV1.m to put it in the correct form for application in the Simulink lookup table block.

```
%Variable Frequency Drive Lookup Table Generator
%Runs SPPTorqueDemandGenV1 code in SECTION I, and then runs each
%Torque value through AC50HP_PIControlModuleV1 (PI turned off)
%at various frequencies until Nout matches the corresponding shaft speed

clear all
clc

%SECTION I: SPPTorqueDemandGenV1 code

%Torque Demand Curve Generator
%Runs SPPModuleV1 repeatedly at varying speed N to generate a T-N curve
clear all
clc

%Define Save File
sdir = 'D:\Zack\Independent Model Modules\LookupTables\';
filename = '50HP10ktI33P0Y0p100tol5step05wShaftJ';
%Filename: [MotorHP knots Immersion Pitch Yaw Precision Tolerance Step Size]
filetosave = char(strcat(sdir, filename,'.mat'));
```

```
%Recall Network
sdir_str = 'D:\Zack\SPP\SPP DATA\StevensTestingFiles\Lookup Tables ZZ
Corrected\';
SPPnet = 'SPPNETe253traincgplearngdm_13091_111_100';  %.5765
nettoload = char(strcat(sdir_str, SPPnet, '.mat'));
load(nettoload);

%Parameters held constant
U = 10; %Vessel Speed in knots
kt2ips = 20.2537183; %Convert knots to inches per second
Dspp = 9.7; %SPP diameter in inches
Im = 33; %Immersion ratio supported by the majority of test results
Pi = 0; %Pitch of SPP in degrees
Y = 0; %Yaw of SPP in degrees
Tfactor = -10; %Gain factor to accont for bad amplifier/load sensor during
SPP tests
precision = 100; %Number of data points desired
Maxrpm = 2500; %Maximum desired SPP rpm

%Cycle SPP through N from 2300rpm to 230rpm
JP = zeros(precision,1);
NT = zeros(precision,2);
for c = 1:precision
    NT(c,1) = c*(Maxrpm/precision);
    JP(c,1) = U*kt2ips/(NT(c,1)/60*Dspp);
    NT(c,2) = sim(SPPnet,[Im;Pi;Y;JP(c)])*Tfactor;
end

plot(NT(:,1),NT(:,2)); %Plots speed versus torque

%SECTION II: Develop Lookup Table

%Set input torque to AC50HP_PIcontrolModuleV1 to the set of torques
%correpsonding to the desired operating shaft speeds (Limit to 2500 rpm)

%Define programming parameters
tol = 5; %Tolerance for matching rpm output of AC50HP_PIcontrolModuleV1
felast = 314; %318 w/ only motor J, 313 w/ motor + shaft J
Rowstop = 29;
for c = length(NT):1
    if NT(c,2) < 0
        Rowstop = c+1; %Determines last row with positive torque values
        break
    end
end

%Set Tm and cycle through frequencies to find Nout (from
%AC50HP_PIControlModuleV1phaseV) that matches NT(c,1) within +/-tol
for c = 1:(length(NT)-Rowstop)
    Tm = NT((length(NT)-c+1),2);
    Nm = NT((length(NT)-c+1),1);
    for fe = felast:-0.5:10
        sim('AC50HP_PIcontrolModuleV1phaseV');
        currentVal = round(Nout(length(Nout)))
        desiredVal = round(Nm)
        if currentVal < (desiredVal + tol) && currentVal > (desiredVal -
tol);
```

```
            LUT(c,1) = fe;
            LUT(c,2) = currentVal;
            save([filetosave], 'NT', 'LUT');
            break
        end
        if NT((length(NT)-c),2) < Tm
            felast = fe; %reduces iterations to get to lower torque loads
        else
            felast = 314; %318 w/ only motor J, 313 w/ motor + shaft J
        end
    end
end
```

LookupTablePostProcV1.m

LookupTablePostProcV1.m reads in the speed-frequency conversion table file generated by VFDLookupTableGenV3.m and cleans out any bad data points (zeroes), re-arranges the data in monotonically ascending order, and outputs a new, post-processed file that is needed for PropulsionSystemConfigParamsV5.m to operate.

```
%Lookup Table Post Processor
%Removes zero lines from lookup table and converts to a monotonically
%increasing set of input values
clear all
clc

%Load lookup table to post process
load 'D:\Zack\Independent Model
Modules\LookupTables\50HP10ktI33P0Y0p100tol5step05wShaftJ'

d=1;
for c = 1 : length(LUT)
    if LUT(c,2) == 0
        c=c+1;
    else
        LUTred(d,1) = LUT(c,2);
        LUTred(d,2) = LUT(c,1);
        d=d+1;
    end
end

for c = 1: length(LUTred)
    LUTm((length(LUTred)-c+1),1) = LUTred(c,1);
    LUTm((length(LUTred)-c+1),2) = LUTred(c,2);
end

save(['D:\Zack\Independent Model
Modules\LookupTables\50HP10ktI33P0Y0p100tol5step05wShaftJPP.mat'], 'LUTm')
```

ResonancePlotterV1.m

ResonancePlotterV1.m runs the complete propulsion plant model to demonstrate the existence of shaft resonance by operating the model at various constant speeds and plotting the motor, shaft, and propeller speeds and torques in a convenient pair of plot windows. Each window contains either the speeds or torques of the system and may be saved or exported as .jpeg images as desired.

```
%Generate data plots for multiple runs of constant speed:
%Run PropulsionSystemConfigParamsV5.m to initialize the propulsion plant
%and load the needed variables into the workspace

limH = 1300; %Upper limit of speed range (rpm)
limL = 1000; %Lowerlimit of speed range (rpm)
SpeedStep = 50; %Step size for speed range (rpm)
colorvec = ['r'; 'g'; 'b'; 'c'; 'm'; 'y'; 'k'; 'r'; 'g'; 'b'; 'c'; 'm'; 'y';
'k'];
colorcounter = 1;
figure(1)
clf
figure(2)
clf
for c = limL:SpeedStep:limH
    SP = c;
    sim('AC50HP_PI_Shaft_SPPV8');
    color = colorvec(colorcounter,1);
    figure(1)
    subplot(3,1,1);
    plot(Speeds.time(:,1), Speeds.signals(1,1).values(:,1), color);
    title('Motor Speed');
    xlabel('Time (s)');
    ylabel('Speed (rpm)');
    hold on
    subplot(3,1,2);
    plot(Speeds.time(:,1), Speeds.signals(1,2).values(:,1), color);
    title('Shaft Speed');
    xlabel('Time (s)');
    ylabel('CM Speed (rad/s)');
    hold on
    subplot(3,1,3);
    plot(Speeds.time(:,1), Speeds.signals(1,3).values(:,1), color);
    %legend('800rpm', '1000rpm', '1200rpm', '1400rpm', '1600rpm', '1800rpm',
'2000rpm', '2200rpm', '2400rpm', 1);
    legend('1000rpm', '1050rpm', '1100rpm', '1150rpm', '1200rpm', '1250rpm',
'1300', 1);
    title('SPP Speed');
    xlabel('Time (s)');
    ylabel('Advance Ratio');
    ylim([0 5]);
```

```
    hold on
    figure(2)
    subplot(3,1,1);
    plot(Torques.time(:,1), Torques.signals(1,1).values(:,1), color);
    title('Motor Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,2);
    plot(Torques.time(:,1), Torques.signals(1,2).values(:,1), color);
    title('Shaft Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,3);
    plot(Torques.time(:,1),Torques.signals(1,3).values(:,1), color);
    %legend('800rpm', '1000rpm', '1200rpm', '1400rpm', '1600rpm', '1800rpm',
'2000rpm', '2200rpm', '2400rpm', 1);
    legend('1000rpm', '1050rpm', '1100rpm', '1150rpm', '1200rpm', '1250rpm',
'1300rpm', 4);

    title('SPP Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    colorcounter = colorcounter + 1;
end
```

VibrationControlPlotterV1.m

VibrationControlPlotterV1.m is very similar to ResonancePlotterV1.m, except it runs the system with no motor control, and with PID motor control to demonstrate the ability of a PID controller to dampen the shaft resonance vibration. The script overlays the various speed plots with and without PID control for clear review and demonstration. Again the script outputs Matlab plots that may be saved or exported as desired.

```
%Generate data plots for multiple runs of constant speed with and without PID
control:
%Run PropulsionSystemConfigParamsV5.m to initialize the propulsion plant
%and load the needed variables into the workspace

Speedset = [800, 1200, 2400] %Select speeds to test
colorvec = ['r'; 'g'; 'b'; 'c'; 'm'; 'y'; 'k'; 'r'; 'g'; 'b'; 'c'; 'm'; 'y';
'k'];
colorcounter = 1;
figure(1)
clf
figure(2)
clf

PIDdelay = 999; %Turn off PID controller

%Run and plot figures for no PID control
for c = Speedset
    SP = c;
    sim('AC50HP_PI_Shaft_SPPV8');
    color = colorvec(colorcounter,1);
    figure(1)
    subplot(3,1,1);
    plot(Speeds.time(:,1), Speeds.signals(1,1).values(:,1), color);
    title('Motor Speed');
    xlabel('Time (s)');
    ylabel('Speed (rpm)');
    hold on
    subplot(3,1,2);
    plot(Speeds.time(:,1), Speeds.signals(1,2).values(:,1), color);
    title('Shaft Speed');
    xlabel('Time (s)');
    ylabel('CM Speed (rad/s)');
    hold on
    subplot(3,1,3);
    plot(Speeds.time(:,1), Speeds.signals(1,3).values(:,1), color);
    legend('800rpm', '1200rpm', '2400rpm', '800rpm PID', '1200rpm PID',
'2400rpm PID', 1);
    title('SPP Speed');
    xlabel('Time (s)');
    ylabel('Advance Ratio');
```

```
    ylim([0 5]);
    hold on
    figure(2)
    subplot(3,1,1);
    plot(Torques.time(:,1), Torques.signals(1,1).values(:,1), color);
    title('Motor Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,2);
    plot(Torques.time(:,1), Torques.signals(1,2).values(:,1), color);
    title('Shaft Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,3);
    plot(Torques.time(:,1),Torques.signals(1,3).values(:,1), color);
    legend('800rpm', '1200rpm', '2400rpm', '800rpm PID', '1200rpm PID',
'2400rpm PID', 4);
    title('SPP Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    colorcounter = colorcounter + 1;
end

PIDdelay = 0; %Turn on PID control

%Run and plot figures for with PID control
for c = Speedset
    SP = c;
    sim('AC50HP_PI_Shaft_SPPV8');
    color = colorvec(colorcounter,1);
    figure(1)
    subplot(3,1,1);
    plot(Speeds.time(:,1), Speeds.signals(1,1).values(:,1), color);
    title('Motor Speed');
    xlabel('Time (s)');
    ylabel('Speed (rpm)');
    hold on
    subplot(3,1,2);
    plot(Speeds.time(:,1), Speeds.signals(1,2).values(:,1), color);
    title('Shaft Speed');
    xlabel('Time (s)');
    ylabel('CM Speed (rad/s)');
    hold on
    subplot(3,1,3);
    plot(Speeds.time(:,1), Speeds.signals(1,3).values(:,1), color);
    legend('800rpm', '1200rpm', '2400rpm', '800rpm PID', '1200rpm PID',
'2400rpm PID', 1);
    title('SPP Speed');
    xlabel('Time (s)');
    ylabel('Advance Ratio');
    ylim([0 5]);
    hold on
    figure(2)
    subplot(3,1,1);
```

```
    plot(Torques.time(:,1), Torques.signals(1,1).values(:,1), color);
    title('Motor Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,2);
    plot(Torques.time(:,1), Torques.signals(1,2).values(:,1), color);
    title('Shaft Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    subplot(3,1,3);
    plot(Torques.time(:,1),Torques.signals(1,3).values(:,1), color);
    legend('800rpm', '1200rpm', '2400rpm', '800rpm PID', '1200rpm PID',
'2400rpm PID', 4);
    title('SPP Torque');
    xlabel('Time (s)');
    ylabel('Torque (Nm)');
    hold on
    colorcounter = colorcounter + 1;
end
```

TorqueVcurrentPlotterV1.m

TorqueVcurrentPlotterV1.m runs the integrated propulsion plant model AC50HP_PI_Shaft_SPPv6.mdl at three different speeds and for each speed, generates a plot of each of the three stator current draws as they relate to the electromagnetic torque. The output plots may be saved or exported as desired and represent the relationship between stator current and electromagnetic torque, allowing one to recognize patterns in the relationships depending on physical outputs of the AC induction motor.

```
%Generate data plots for torque vs speed at and away from shaft resonance
%Run PropulsionSystemConfigParamsV5.m to initialize the propulsion plant
%and load the needed variables into the workspace

Speedset = [800, 1200, 2400] %Select speeds to test
colorvec = ['r'; 'g'; 'b'; 'c'; 'm'; 'y'; 'k'; 'r'; 'g'; 'b'; 'c'; 'm'; 'y';
'k'];
figurecounter = 1;
figure(1)
clf
figure(2)
clf
figure(3)
clf

PIDdelay = 999; %Turn off PID controller

%Run and plot figures for no PID control
for c = Speedset
    colorcounter = 1;
    SP = c;
    sim('AC50HP_PI_Shaft_SPPV6');
    color = colorvec(colorcounter,1);
    figure(figurecounter)
    subplot(3,1,1);
    plot(Torques.signals(1,1).values(:,1),
StatorCurrents.signals(1,1).values(:,1), color);
    title(strcat('Electromagnetic Torque vs Stator Current a at
N=',num2str(c),'rpm'));
    xlabel('Torque (Nm)');
    ylabel('Current (A)');
    xlim([0 150]);
    ylim([-100 100]);
    colorcounter = colorcounter + 1;
    color = colorvec(colorcounter,1);
    subplot(3,1,2);
    plot(Torques.signals(1,1).values(:,1),
StatorCurrents.signals(1,2).values(:,1), color);
```

```
    title(strcat('Electromagnetic Torque vs Stator Current b at
N=',num2str(c),'rpm'));
    xlabel('Torque (Nm)');
    ylabel('Current (A)');
    xlim([0 150]);
    ylim([-100 100]);
    colorcounter = colorcounter + 1;
    color = colorvec(colorcounter,1);
    subplot(3,1,3);
    plot(Torques.signals(1,1).values(:,1),
StatorCurrents.signals(1,3).values(:,1), color);
    title(strcat('Electromagnetic Torque vs Stator Current c at
N=',num2str(c),'rpm'));
    xlabel('Torque (Nm)');
    ylabel('Current (A)');
    xlim([0 150]);
    ylim([-100 100]);
    figurecounter = figurecounter + 1;
end
```

APPENDIX B: MODEL COMPARISON TESTING

Test 1: Apply complete, detailed motor model to simple shaft and constant propeller torque

Setup: Integrated_Model_Niko_Test_1.mdl

shaft_config_Test_1.m

Parameters:

IC1 = acceleration integrator initial condition

IC2 = velocity integrator initial condition

IC3 = motor block initial conditions [slip, electrical angle, current magnitudes, current phases]

Prop Torque = 0 - 1000

Speed = 100 - 5000

Various combinations of the above parameters were set and runs completed with the following results:

1) Typical Results for SP = 1000rpm, Tm = 100Nm, IC1 = IC2 = 0, IC3 = [1,0 zeros] are as shown in figure B1-1 below.
2) For torque loads above the motor rated power output at a given speed, as would be expected, the motor speed does not reach full set point speed.
3) For variations of IC1, increases tend to drive the initial transient of the solution, but do not cause instability of the program other than a choppy start as seen in figure B1-2.
4) For variations of IC2, large increases tend to destabilize initial transient then the system recovers as shown below in figure B1-3.

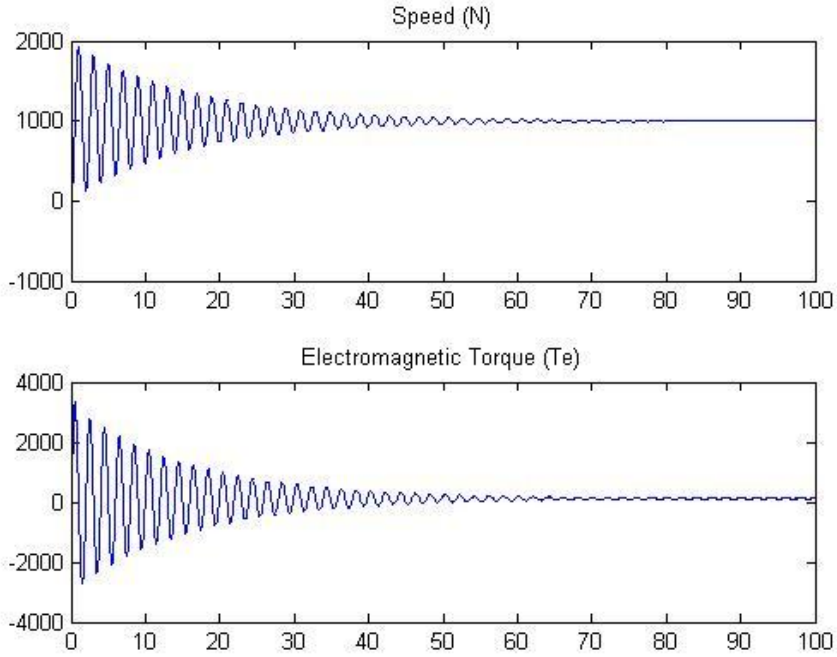

Figure B1-1: Typical results of motor output

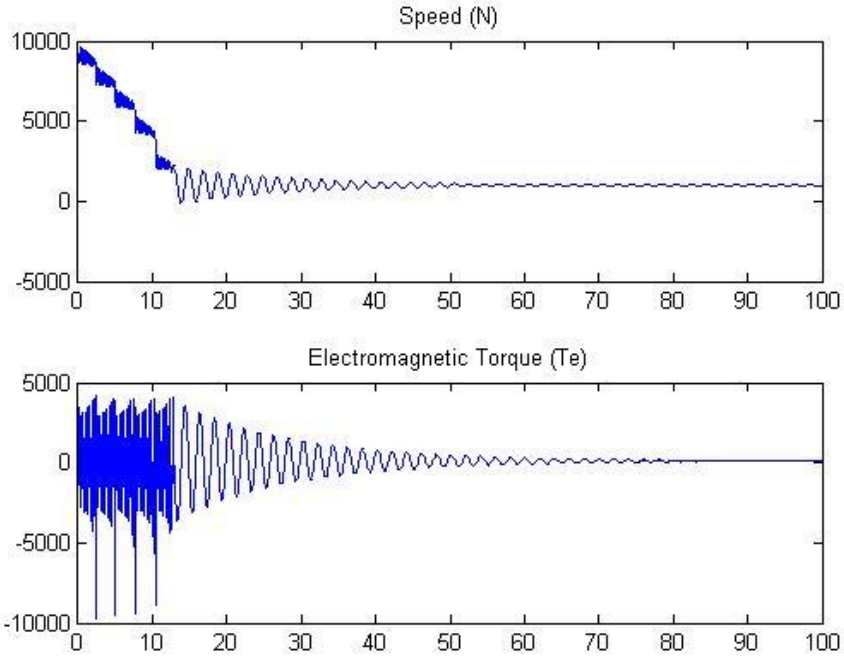

Figure B1-2: Unstable start for IC1 = 10,000

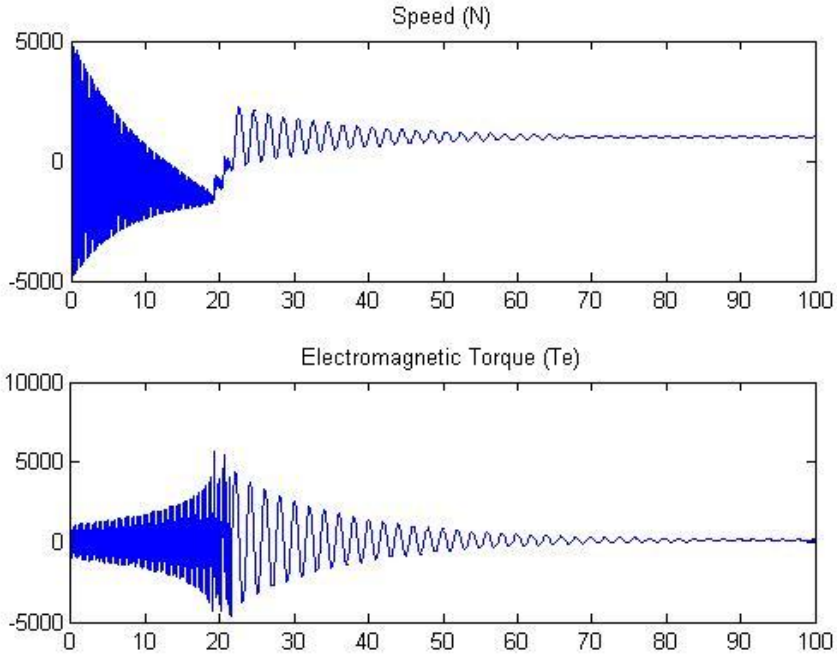

Figure B1-3: IC2 = 10,000 causes initial instability

5) Variations in speed can cause instability. If torque load is high and speed set point is high, divergence can occur as shown in figure B1-4 below. Other interesting results are shown in figure B1-5, when the set point was set at 1200 rpm and the torque load at 100 Nm. The motor achieved speed accurately, but the torque load out from the shaft and subsequently the electromagnetic torque of the motor show high amplitude excitation – resonance of the shaft. The resonance is exaggerated in this case due to small values for stiffness in the shaft simple model. These anomalies are thought to be due to the tuning of the speed-frequency lookup table used to drive the motor in the simple control system (lookup table) in use during these tests. The table was generated based on the torque-speed curve for the SPP propulsor. Since the testing was carried out using torque-speed combinations not necessarily lying on the torque-speed curve for the SPP, it should be expected that the motor will not correctly respond to all combinations.

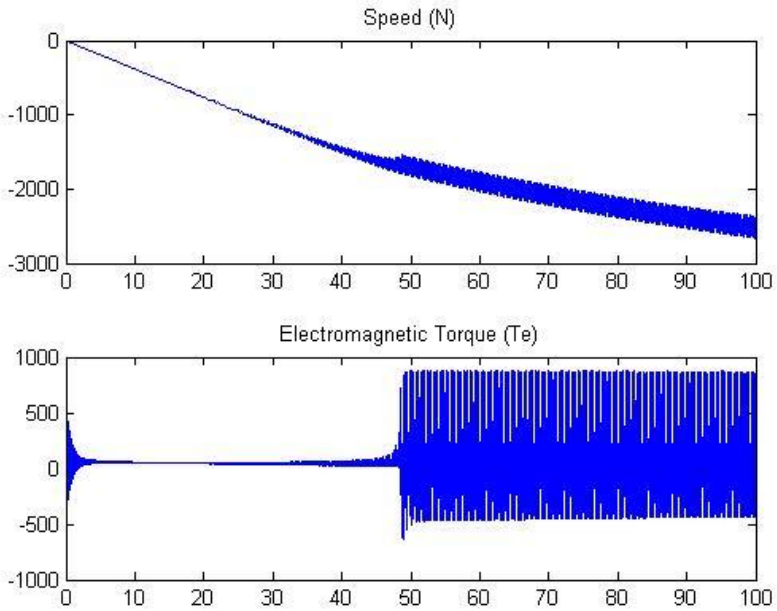

Figure B1-4: Divergence due to excessive torque load at specified SP: SP = 2000rpm, Tm = 100Nm

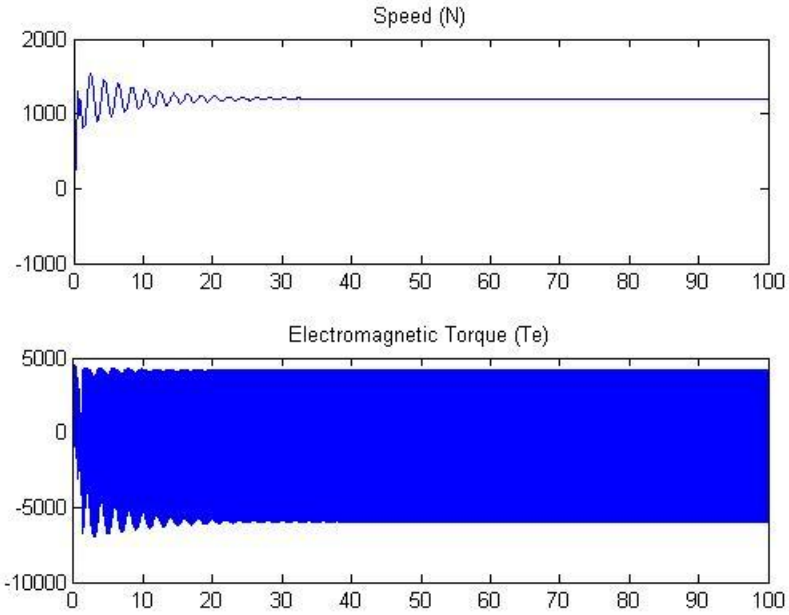

Figure B1-5: Excitation of the shaft at SP = 1200rpm and Tm = 100Nm

6) Modifying IC3 was carried out by setting the initial slip to 1 (the default value) and 10. With increase initial slip, increased transient instability resulted, but the model did recover and achieved the set point speed. Similarly setting non-zero initial electrical

angle causes initial deviations from set point, but eventually recovers. Increased initial current amplitude causes large initial torque oscillations early in the transient time, but recovers quickly. Modifying the initial phase angles has little effect on the output of the model. Figure B1-6 shows typical increases in transient instability and how it is recoverable when some motor parameters are modified.

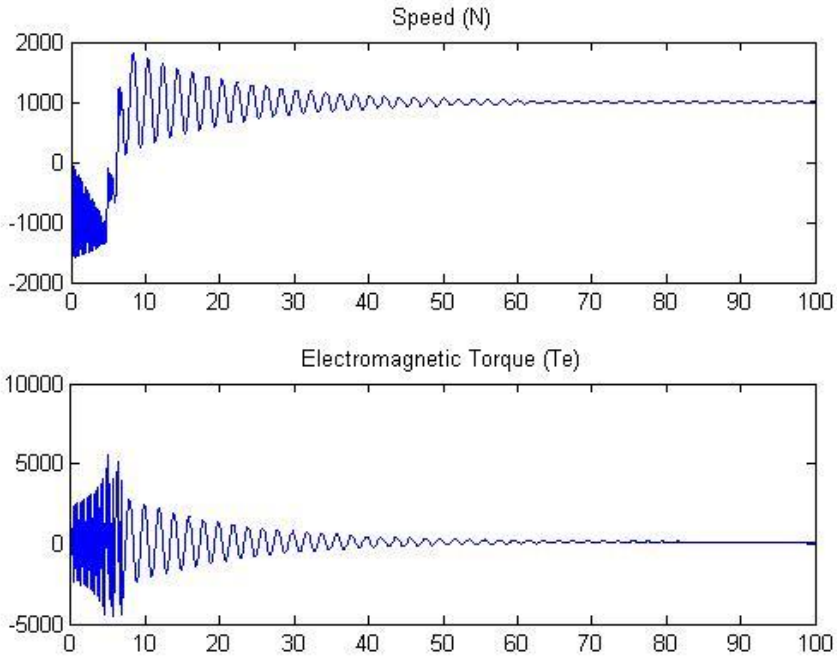

Figure B1-6: Increase in initial slip to 10 in motor initial conditions with SP = 1000rpm and Tm = 100Nm

Concluding from Test 1, the following initial conditions appear to be the best fit:

IC1: Set the acceleration integrator initial condition to 0.

IC2: Set the velocity integrator initial condition to 0.

IC3: Set the motor initial conditions to the Matlab defaults.

Deviating from speed set point and torque demand values laying on the Speed-Torque curve derived from the SPP can cause instability in the model. Further investigation of this issue will be conducted with the complete model to ensure that certain settings do not cause torque and speed demands on the motor that it cannot accurately model or is incapable of producing. This phenomena is explained by the fact that the frequency lookup table was generated based on the torque demand of the SPP and so the frequency settings to achieve a given speed may be inaccurate if the applied torque is not close to what is demanded by the SPP at that speed set point. A PI or PID control can help mitigate issues with this by correcting the lookup table for those torque demands that push the speed off the set point. The controller will be tuned, tested, and implemented for the full model simulations.

Test 2: Apply complete, detailed Neural Net Prop model to simple shaft and motor models

Setup: Integrated_Model_Niko_Test_2.mdl

shaft_config_Test_1.m

Parameters:

The SPP model does not have any parameters to modify. It is a simple input-output black box. The torque speed curve used in the simplified motor model is compared to that produced by the Neural Net SPP model.

The output of the Neural Net SPP and the simple quadratic models are shown below in figures B2-1 and B2-2, respectively. As can be seen, both models generate good data and they differ because they produce different torque requirements for the varying speed input. In figure B2-1, it can be noted that the torque goes below zero. This is due to the Neural Net SPP acting in generation mode, meaning the prop is being rotated by the forward motion of the ship due to the ship speed. The Neural Net SPP must accept a vessel speed, U, as an input in order to compute the advance ratio which is uses as a speed input. Figure B2-1 was generated using a forward vessel speed of 10 knots. By reducing the vessel speed to 5 knots while holding the same shaft rotational rate, figure B2-3 was be generated with the SPP demanding higher torque to rotate.

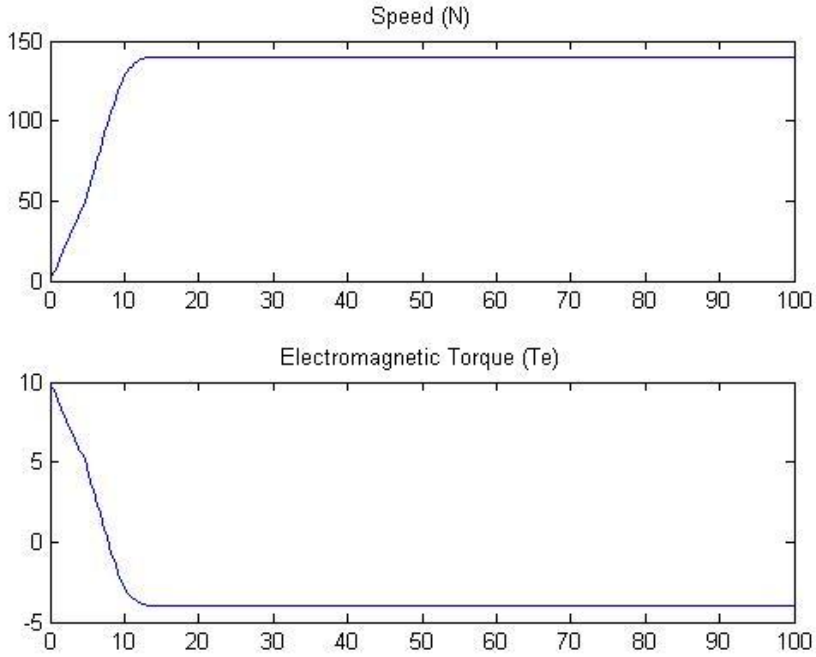

Figure B2-1: Neural Net SPP output

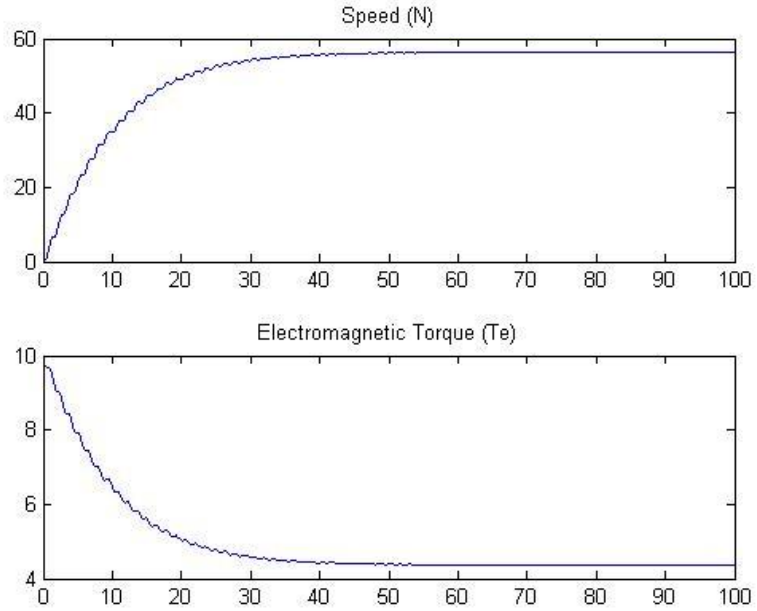

Figure B2-2: Quadratic prop output

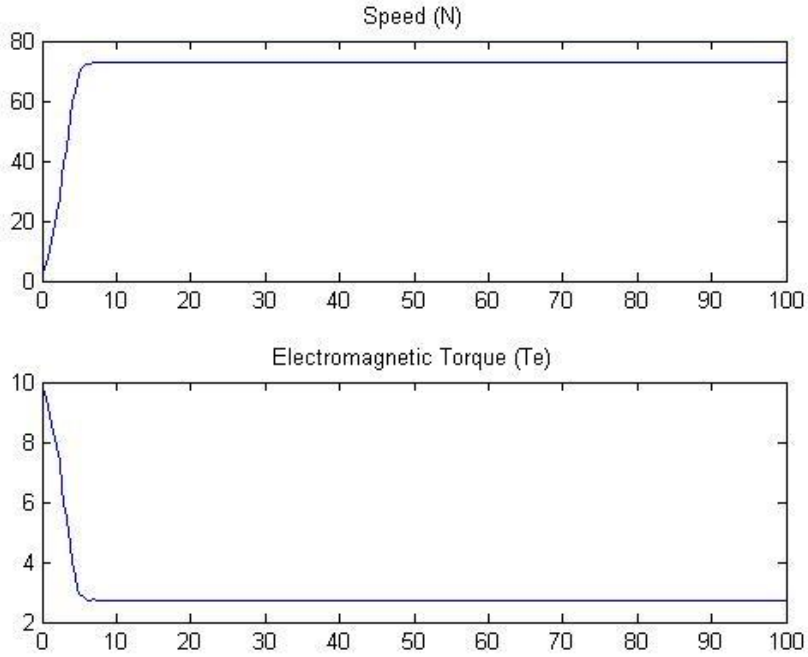

Figure B2-3: Neural Net SPP with U = 5kt

Test 3: Apply complete, detailed shaft model to simple quadratic prop and motor models. Compare to the simple, multi-element shaft model.

Setup: Integrated_Model_Niko_Test_2.mdl

Detailed_Shaft_Zack1.mdl

Propulsion_powertrain_Zack1.mdl

shaft_config_Test_1.m

Minor modification to the complete, detailed shaft model was completed to provide proper integration into the simple motor model and the simple propeller model. The detailed shaft model expects a negative value for torque load and so in testing, the summing block for the input torque, the friction loss, and the torsional loss was modified to multiply the input torque input by –1. Once this change was made, the simple, multi-element shaft and the complete, detailed shaft models produce the same results for the same parameters. The plots for each can be seen below in figures B3-1 and B3-2. The simpler multi-element shaft model simulates a shaft with set parameters for the torsional spring matrix, the inertia matrix, and the frictional loss matrix with the same values for all non-zero elements within those matrices. The complete, detailed shaft model allows for specific user inputs that control the inertia, stiffness, and frictional loss for each individual element as described in Chapter 3. For the comparison tests, the values were set the same for both models to compare the functionality of the two models.

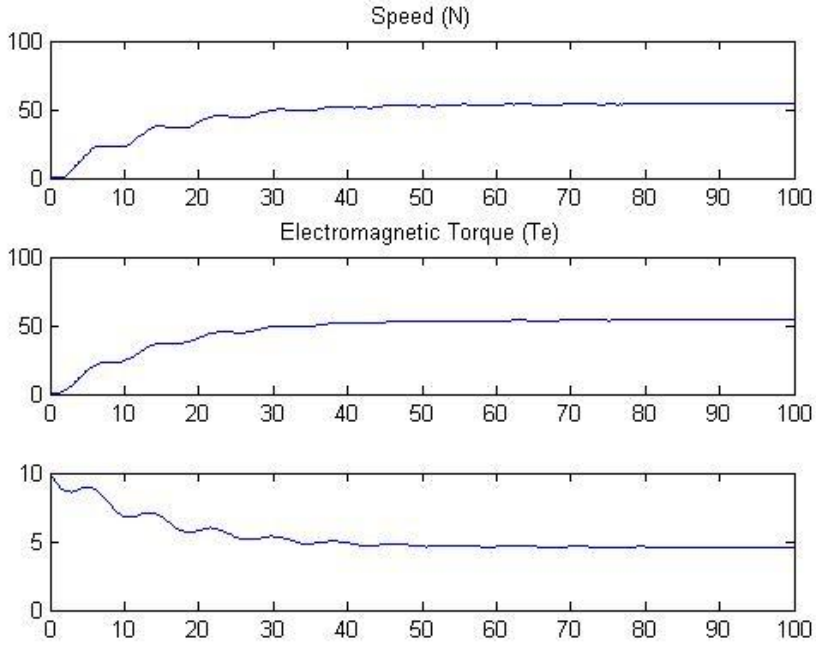

Figure B3-1: Simple, multi-element shaft output

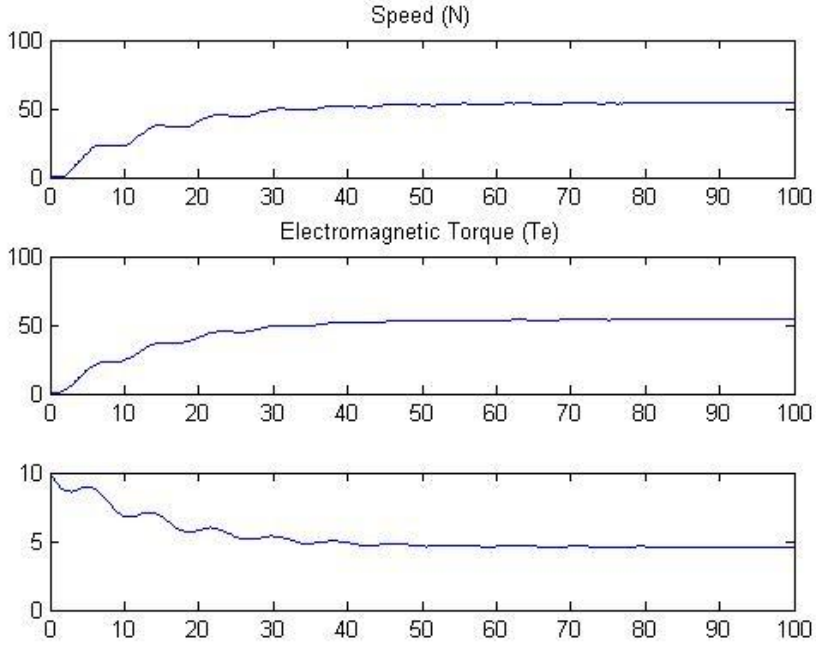

Figure B3-2: Complete, detailed shaft output

As shown below in figures B3-3 and B3-4, when overlaid, the outputs align perfectly indicating complete agreement between the simple, multi-element shaft model and the complete, detailed shaft model. Finally, the simple shaft model with a single element was compared. This model shows slight differences in shape due to its lack in the same degree of freedom that the multi-element models provide. Figure B3-5 shows the output result of the simple, single element model for comparison to the multi-elemen models. The agreement in shape is the final indicator that all the individual component models of the propulsion plant modeling effort are functional and provide useful results.

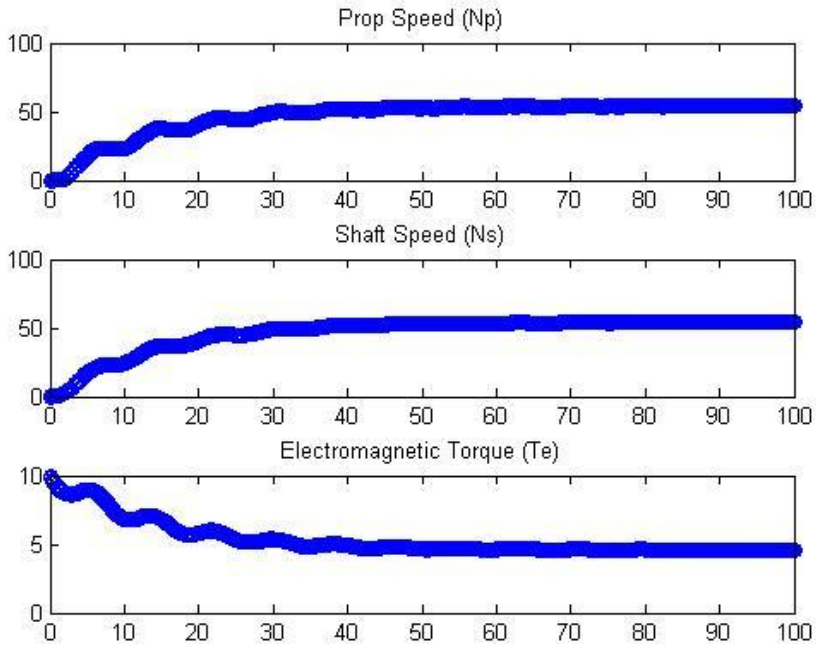

Figure B3-3: Overlay of Complete and Simple shaft model outputs

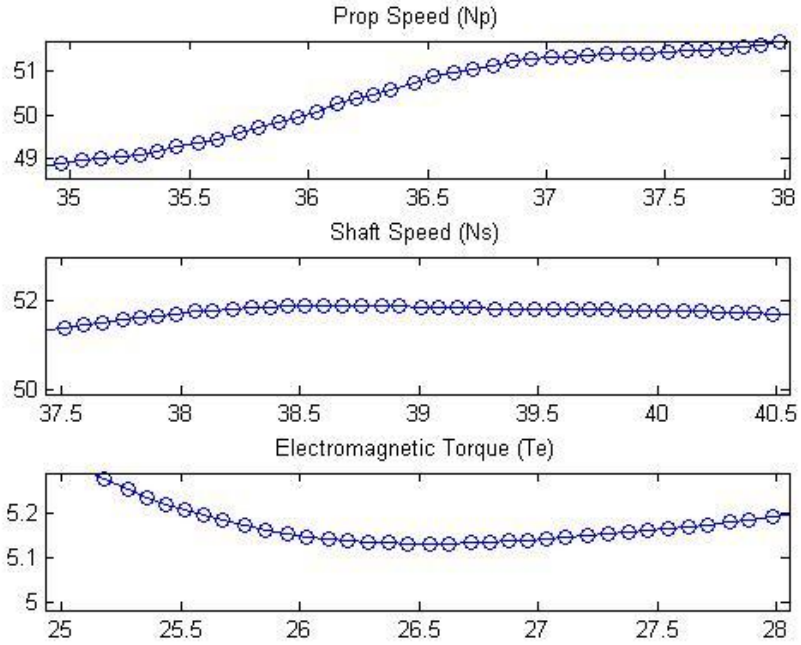

Figure B3-4: Close up showing agreement of two models

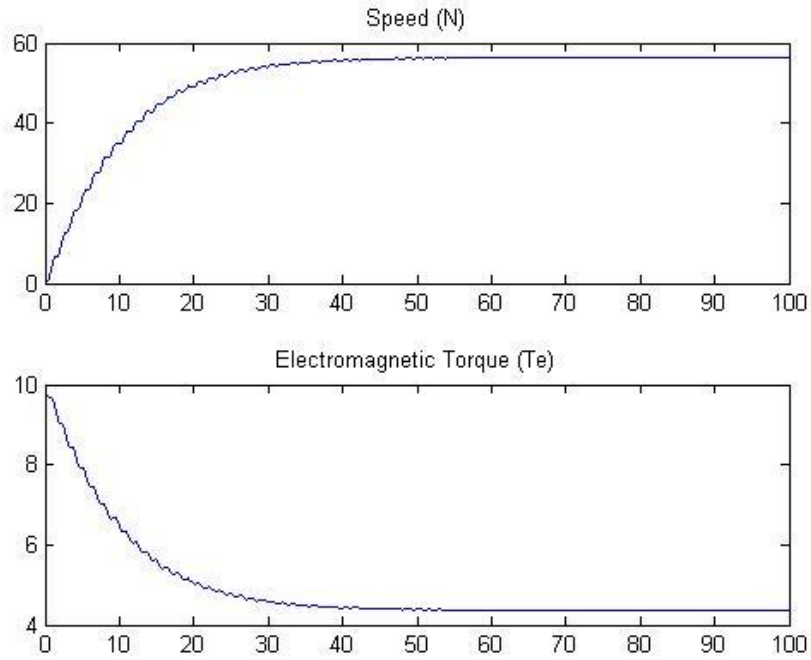

Figure B3-5: Output of the simple, single element shaft shows good agreement as well

APPENDIX C: ANNOTATED LIST OF FIGURES

Figure B3-2: Complete, detailed shaft output

Original image created by Zachary Samuel Zisman, 2011

Figure B3-3: Overlay of Complete and Simple shaft model outputs

Original image created by Zachary Samuel Zisman, 2011

Figure B3-4: Close up showing agreement of two models

Original image created by Zachary Samuel Zisman, 2011

Figure B3-5: Output of the simple, single element shaft shows good agreement as well

Original image created by Zachary Samuel Zisman, 2011