5-2015

A Neural Network Approach to Fault Detection in Spacecraft Attitude Determination and Control Systems

John N. Schreiner
Utah State University

A NEURAL NETWORK APPROACH TO FAULT DETECTION IN SPACECRAFT ATTITUDE DETERMINATION AND CONTROL SYSTEMS

by

John N. Schreiner

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Aerospace Engineering

Approved:

Dr. Rees Fullmer Major Professor	Dr. David Geller Committee Member
Dr. Charles Swenson Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

Abstract

A Neural Network Approach to Fault Detection in Spacecraft Attitude Determination and Control Systems

by

John N. Schreiner, Master of Science

Utah State University, 2015

Major Professor: Dr. Rees Fullmer
Department: Mechanical and Aerospace Engineering

This thesis proposes a method of performing fault detection and isolation in spacecraft attitude determination and control systems. The proposed method works by deploying a trained neural network to analyze a set of residuals that are defined such that they encompass the attitude control, guidance, and attitude determination subsystems. Eight neural networks were trained using either the resilient backpropagation, Levenberg-Marquardt, or Levenberg-Marquardt with Bayesian regularization training algorithms. The results of each of the neural networks were analyzed to determine the accuracy of the networks with respect to isolating the faulty component or faulty subsystem within the ADCS. The performance of the proposed neural network-based fault detection and isolation method was compared and contrasted with other ADCS FDI methods. The results obtained via simulation showed that the best neural networks employing this method successfully detected the presence of a fault 79% of the time. The faulty subsystem was successfully isolated 75% of the time and the faulty components within the faulty subsystem were isolated 37% of the time.

(140 pages)

Public Abstract

A Neural Network Approach to Fault Detection in Spacecraft Attitude Determination and Control Systems

by

John N. Schreiner, Master of Science

Utah State University, 2015

Major Professor: Dr. Rees Fullmer
Department: Mechanical and Aerospace Engineering

CubeSats are employed in a variety of missions as scientific platforms, low-cost technology demonstrators, and in the future, they will conduct service missions as part of larger satellite constellations. CubeSat ADCS designers today are being tasked with designing to ever increasing accuracy requirements. The ability to hold those requirements rests with the attitude control system being robust enough and able to sufficiently respond to changes in the control environment. The need for greater control autonomy is then evident in the need for these systems to be able to react independently to changes in the system dynamics and to identify and accommodate system faults. A key ability of a fault tolerant control system then is the capability to successfully detect and isolate faults as it provides a method for early detection and diagnosis of unforeseen faults, which in turn provides a spacecraft with a greater degree of autonomy.

The focus of this thesis was to determine whether there is enough information coming from a pattern of residuals defined by the attitude determination and control system to allow a neural network to discern whether or not a fault has occurred. To do this, a set of residuals was defined based on a comparison of a spacecraft's ADCS telemetry to estimated state values for both nominal and fault states. This set of residuals served as the

training set for a series of neural networks that were trained using either the resilient back-propagation, Levenberg-Marquardt, or Levenberg-Marquardt with Bayesian regularization training algorithms. After the networks were trained their outputs were calculated for both reapplication of the training data as well as for novel data of which they had no *a priori* knowledge.

The performance of the neural networks in detecting faults with this scheme leaves much to interpretation. Though all of the networks were trained from the same example set, significant differences exist in the ability of the networks to positively detect and isolate the faults with consistency. Where one network may excel in detecting the faults in a certain components, it may fare poorly at another. In general, the networks were better able to detect and isolate faults in the components of the attitude control and guidance subsystems, and with few exceptions less able to isolate faults of the attitude determination sensors.

To my wife, Jessica, who never stopped believing in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

It is well established that space-tolerant, off-the-shelf hardware is prone to failure [1]. In regards to a spacecraft's attitude determination and control system (ADCS) a fault in an actuator or failure of a sensor can lead to a loss of control authority, putting the spacecraft at risk and potentially jeopardizing mission objectives. In the past, this problem has been addressed by the addition of redundant actuators or sensors. But what if the ADCS system is space-limited, such as on a CubeSat? In such instances hardware redundancies to maintain control authority may simply not be an option and the need for fault detection is critical.

CubeSats are employed in a variety of missions as scientific platforms, low-cost technology demonstrators, and in the future, to conduct service missions as part of larger satellite constellations [1]. CubeSat ADCS designers today are being tasked with designing to ever increasing accuracy requirements. The ability to hold those requirements rests with the attitude control system being robust enough and able to sufficiently respond to changes in the control environment. The need for greater control autonomy is then evident in the need for these systems to be able to react independently to changes in the system dynamics and to identify and accommodate system faults [1, 2]. A key ability of a fault tolerant control system then is the capability to successfully detect and isolate faults as it can provide a method for early detection and diagnosis of unforeseen faults, which in turn provides a spacecraft with a greater degree of autonomy [3].

1.2 Previous Work

Much research has been done with respect to fault detection and isolation (FDI). Many

FDI methods methods work by defining a residual for a system and then thresholding that residual to determine the presence of a fault. In general the residuals are defined as the difference between a measured and state-estimated value. The environment in which the fault detection and isolation takes place in large part determines the type of state estimator employed. Kalman filters are employed as state estimators in noisy environments [3], whereas in more deterministic settings Luenberger observers have been utilized with success.

There are perhaps two main approaches to FDI: mechanical and analytical. Mechanical approaches to FDI rely on physical redundancies in order to fully isolate and recover from fault conditions. In many spacecraft applications, however, where extra mass in any system comes at a premium, it is not always possible to add redundant sensors or actuators [3]. In such cases analytical methods must be considered.

Analytical approaches to FDI can be further broken into two categories: observer-based and learning-based. Observer-based FDI approaches employ either Luenberger observers in deterministic environments, Kalman filters in noisy environments, or a combination of the two and are used as state estimators [4]. The estimation is compared to the system output, whether controller, actuator, or sensor, in order to generate a residual. During nominal operation these residuals are small and should be equal to about zero. Faults are indicated if the residuals experience a large swing away from the nominal value [3, 4].

Learning-based FDI approaches employ neural networks, often in conjunction with state estimators. These neural networks can be trained with knowledge of a spacecraft's operational states. They are able to "learn" the behavior of the plant dynamics and fault states such that they are able to discern, via the state estimators, when and where a fault has occurred [4].

1.3 Thesis Statement

Since attitude determination and control systems can be failure prone the need for greater fault identification and accommodation is critical. Many methods exist today to accomplish this. Offline techniques such at fault tree analysis and failure modes and effects

analysis give insight into ACS design by providing *a priori* knowledge of subsystem causal relationships and potential failure modes. Online methods of fault accommodation, such as fault tolerant control schemes, seek less to identify the root cause of a given failure and more to handle the fault to allow continued stable operation.

1.4 Problem Description

The focus of this thesis is to determine whether there is enough information coming from a pattern of residuals defined by the attitude determination and control system to discern whether or not a fault has occurred. This will be accomplished by deploying a neural network that will be trained to recognize nominal and non-standard behavior in the ADCS residuals. A description of the neural network is given in section 5.1. Such a learning-based approach to FDI will require the network to first be trained to differentiate between normal and abnormal behavior patterns. To do this, a set of residuals will be defined based on a comparison of a spacecraft's ADCS telemetry to estimated state values. These residuals will be generated for both nominal and fault states in order to generate a suitable set of training data from which the network can learn what constitutes normal and abnormal behavior.

1.5 Approach to Fault Detection and Isolation

As the attitude determination and control system is so closely coupled with the main sources of error, it will be examined directly in order to determine the spacecraft's fault state. As shown in figure 1.1, the attitude determination and control system has four main input categories. These are external inputs, sensor inputs, software inputs, and actuator inputs.

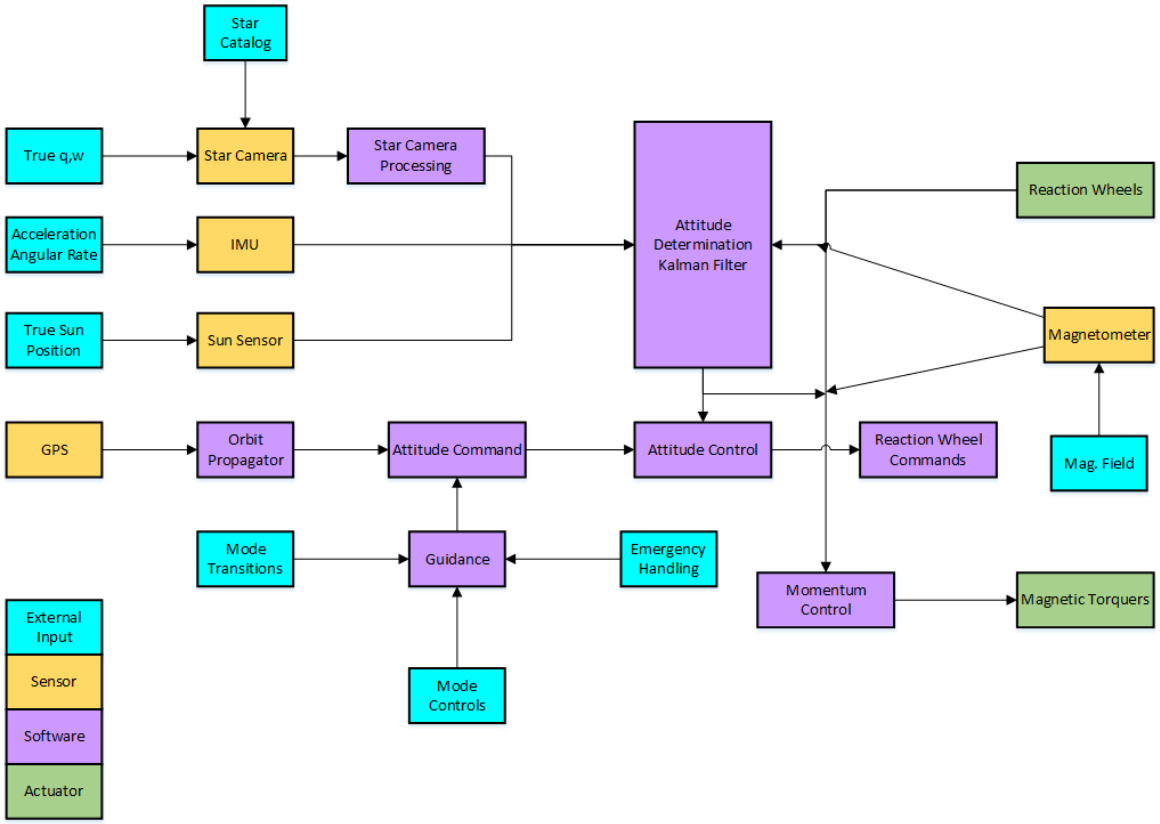

Fig. 1.1: ADCS process map showing the main input categories

External inputs originate externally to the ADCS. These could be "truth" quantities such as the position of the sun or the earth's magnetic field or they could be software commands initiated from the ground, such as ADCS mode transitions. Sensor inputs are the inputs to the ADCS as measured by its sensors, such as the measured position of the sun or spacecraft angular rotation rate. Software inputs are those generated onboard the ADCS, for example by the guidance flight software or the sensor noise filter. Lastly, the actuator inputs are generated specifically by the ADCS control actuators, for example the torque generated by a reaction wheel.

As the performance and output of the ADCS are so dependent on the operational state of the spacecraft, it will be used as the residual generator to train the neural network. The residuals themselves are generated based on the input categories of the ADCS. A detailed description of how they are defined follows in chapter 3.

When taken as a whole the set of residuals defined by the ADCS paints a picture of the spacecraft's operational state at any given moment. During normal operation, such as when the spacecraft is stationkeeping or maneuvering, the set of residuals will have a certain pattern to them, each remaining in the vicinity of a nominal value. When a fault has occurred, such as the failure of a control actuator, one or more of the residuals will display either a positive or negative bias relative to the nominal. It is the nominal-biased behavior of the residuals that differentiates between normal and abnormal states, and it is this behavior that the neural network will be trained to recognize. Section 5.1 provides a description of what a neural network is and how it will be used to recognize faults in the spacecraft.

For the purpose of this thesis, a fault is defined as any time the ADCS fails to do what it was commanded to do, or, having done what it was commanded to do, has failed to put the spacecraft into the desired orientation. Faults can arise in the ADCS from any of the input categories in figure 1.1. Faults pertaining to external inputs come in the form of user error in the definition of the guidance commands sent from the ground to the spacecraft. Sensor errors refer to malfunction of the onboard attitude determination and environmental sensors. Software errors relate specifically to errors in the onboard orbit propagator, its associated environmental models, as well as the reference time. Lastly, actuator errors can relate to the polarity of an actuator, an actuator's performance, or the amount of power supplied to it.

The approach to fault detection and isolation in this thesis is as follows:

1. Estimate the state of the spacecraft

2. Observe the state of the spacecraft

3. Compare the observed state of the spacecraft to the estimated state of the spacecraft

4. Use a trained neural network to detect and isolate the fault based on the comparison

Chapter 2

Literature Review

2.1 Failure Analysis

A Fault Tolerant Control (FTC) system is a control system that is able to maintain control authority in the presence of system faults or component failures. Such systems make use of a combination of fault analysis during the design phase as well as intelligent control schemes that are able to handle a variety of system faults. In such systems faults are identified during the design phase and those that can accommodated by design are, e.g. redundant sensors or actuators [1]. Faults that can occur online, such as reduced actuation or failure of a boom to properly deploy, are handled by the controller and require appropriate fault handling and control schemes [5].

In the past, faults have been handled in several ways, via redundancy in the control system, failure analysis, or if-then rules applied to the controller [6–8]. Redundancy in the controller provides for a backup system to be present in the instance that a component fails. If a sensor or actuator fails there is another that is able to come online to take its place, such as a second star tracker or a fourth reaction wheel in a 3-axis control system. Failure analysis has proven itself to be an important tool in the design of space missions as it allows the design engineer to identify and accommodate faults before the system is brought online.

Two popular methods of failure analysis are Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) [6]. These approaches differ somewhat in their scope and methodology. FMEA is a bottom-up failure analysis method. For a given component in a system, FMEA attempts to identify all the possible methods in which that component can fail and what the consequences of each failure are. As a design tool FMEA allows the designer to determine the severity of each individual failure mode such that it allows them to focus their design efforts on those modes deemed the most severe [5].

The disadvantage of using FMEA as a failure analysis tool is the potential for it to be exhaustive and to overwhelm the designer with the number of failure possibilities [6]. Fault Tree Analysis differs from FMEA in that rather than trying to determine all of the ways a component can fail, it begins by looking at a system-level fault and drilling down to determine the possible root causes. The major advantage of using this method over FMEA is that it is able to demonstrate the relationship between the various systems and subsystems and how they interact with each other [6].

An expansion of the fault tree analysis has been proposed by Barua, et al, in the form of a diagnostic tree (DX-tree) [9]. The goal of the DX-tree approach is to be able to determine faults and their causes by avoiding manual limit checking of telemetry data and is intended to be a semi-autonomous method of fault detection and identification by serving as a diagnostic tool for ACS system failure analysis. Although online applications of FDI exist in the form of observers and filters in the controller, their main limitations are the accuracy of the models, development costs, and an inability to determine the root cause of the system fault. With the DX-tree approach math models are not required and faults are identified via a combination of telemetry variables and expert operator knowledge. The main benefit of this approach is that is introduces a scheme to raise the level of autonomy in the decision making process.

A third method of fault handling is the application of if-then rules to the system controller. Such systems typically require long and costly development cycles. Due to the fixed nature of the rules, such systems demonstrate only a limited ability to handle dynamic system faults, such as actuator or sensor failures, and can therefore offer only limited assurances as to their robustness and stability [8].

These methods of fault handling provide the attitude control system designer a priori knowledge of potential system faults, giving them the ability to compensate for those faults during the design phase. On their own they provide the designer with valuable insight into the potential failure modes of the system; however, once passed the design phase they do little to compensate for unseen errors during online operation. A more complete approach

to the design of a fault tolerant system is given by Izadi-Zamanabadi and Larsen [5] where, along with the fault analysis methods outlined previously, they propose the addition of an online supervisory system. The purpose of this supervisory system is to act as a design analysis tool as well as providing system monitoring and error diagnosis possibilities. In the proposed supervisory system the individual components of the ADCS system are analyzed for failures via a method such as FTA or FMEA. After the system's failure modes are identified the severity of each mode is assessed to determine which are most severe. Those that found to be mission critical are simulated in software to determine the effects on the spacecraft should they actually occur. With the most severe faults identified the next step in developing the supervisory system is to determine, via a structural analysis [5, 10], the redundant information in the plant. This information is used in the identification and diagnosis of system faults. The next steps concern the accommodation and handling of detected faults via hardware or, preferably, software redundancies. Finally, the last step is to develop the decision logic of the system, which determines which control mode to use based on mission objectives and spacecraft health.

2.2 Fault-tolerant Control

Fault tree and failure modes and effects analyses are excellent tools for accommodating faults during the design phase of an ADCS system and a supervisory system provides online monitoring and fault detection capabilities. The greatest limitation these tools has, however, is that they are only capable of handling the faults that they are designed to handle, and in a control environment that demands ever more stringent accuracy requirements, may not provide adequate robustness in the face of all potential system faults. It is proposed by Narendra [11] that rather than attempt to identify and accommodate all possible system faults during the design phase, an adaptable control system should be introduced that is able to monitor and reconfigure itself in the presence of unknown errors. Traditional control design, shown in figure 2.1, is based on a single, fixed model of the system. Such control assumes that systems are time-invariant where in reality they are often required to operate in multiple environments. Narendra recommends the use of an intelligent controller that has

the ability to operate in multiple environments and to adapt itself to changing environments via a method known as multiple models, switching, and tuning (MMST) [11].

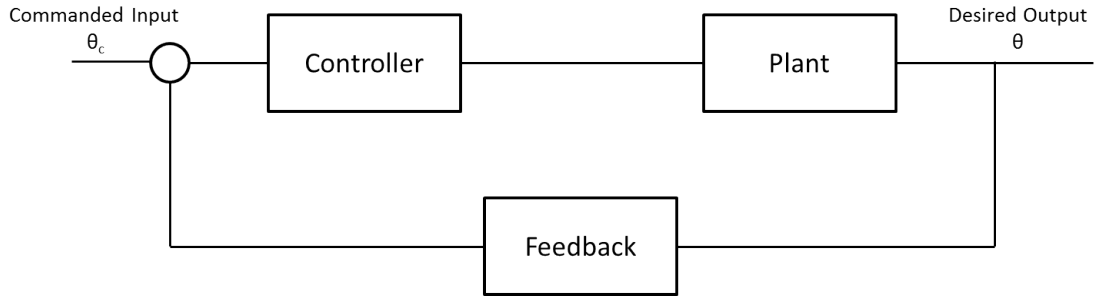

Fig. 2.1: A traditional closed-loop feedback controller

In an MMST control system, shown in figure 2.2, multiple controllers are designed for multiple operating environments. If one can determine which controller most closely matches the given plant dynamics at any given time, that controller can be used to stabilize the system. This is the aspect of "multiple models." Since the performance of a controller can only be measured after it is used, each controller also has an applicable system observer that estimates the current system dynamics. The observers are used to calculate an error function for each given controller and are able to switch to whichever controller currently has the smallest value of that error function, and therefore, most closely matches the current system dynamics. Immediately after the system has switched controllers it begins to adjust the control parameters of the new controller to improve accuracy. This is what is meant by "switching and tuning."

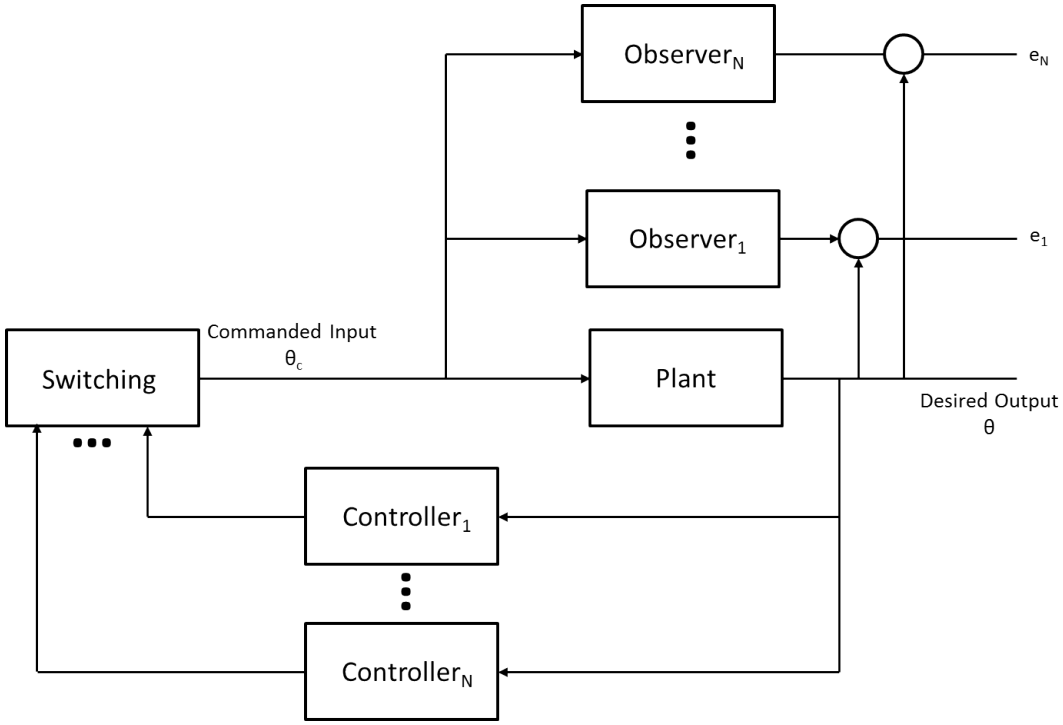

Fig. 2.2: A multiple models, switching and tuning (MMST) control scheme

Chapter 3

Residual Definitions

Before a neural network can be deployed for FDI relative to a spacecraft's ADCS, a set of training data must first be defined. This training data, or sample set, consists of two parts: the inputs to the system and the desired outputs of the system. As stated previously, the behavior of the residuals of the ADCS are indicative of the state of that system and will therefore serve as the input vector to train the network. The residuals are defined for scalar quantities as

$$Residual = ObservedValue - ExpectedValue$$

and for vector quantities as

$$Residual = ||ObservedValue - ExpectedValue||$$

To generate the residuals, two software models are constructed. One model is meant to represent the actual spacecraft. The model outputs are in effect simulated telemetry from a spacecraft's attitude determination and control system. This model will hereafter be referred to as the Fault model. It's purpose is to provide the observed telemetry needed to form the residuals. The second model is the model of the spacecraft, or in this case, the model of the modeled spacecraft. Referred to as the Reference model, its purpose is to provide a basis for comparison to the Fault model. A detailed description of the software models is presented in chapter 3.

One aspect of this thesis is that the fault detection scheme will not be limited to detecting only faults in the actuators or sensors, but that it will be able to provide a general sense as to whether the spacecraft is properly oriented, and if not, what the probable cause of the misorientation is. In this thesis, "properly oriented" means that the spacecraft is in the *desired* orientation, and not necessarily the *commanded* orientation. The state of the

spacecraft, therefore, is expanded beyond the actuators and sensors and includes attitude data as well. To fully define the state of the spacecraft, the ADCS is subdivided into five systems: Attitude and Rate, Guidance, Actuators, Orbit Propagator/Time, and Sensors.

A note on notation is required before proceeding. The attitude determination and control system generally has two kinds of outputs: those that are measured, and those that are estimated. Quantities that are estimated, such as the attitude quaternion from the Kalman filter, are denoted as $\hat{\cdot}$. Quantities that are measured, such as the magnetic field vector, are likewise denoted as $\tilde{\cdot}$.

3.1 Attitude and Rate

Observation of the spacecraft's attitude and rate provide the first indications that a fault has occurred. The attitude and rate residuals are defined as the difference between the observed attitude, and rate and the expected attitude and rate. The spacecraft's primary attitude determination sensors provide the source of the observed attitude and rate. These could be a star camera, a sun sensor, an IMU or rate gyro for example.

The attitude and rate as recorded by the Fault model, i.e. the spacecraft's telemetry, are compared against the attitude and rate of the Reference model. If the spacecraft is doing what it should, then the attitude parameters should have a small residual and is indicative of an "all is well" state.

The attitude and rate residuals, shown in equations (3.1) and (3.2), are defined as the norm of the difference between the attitude and rate as reported by the Fault model's Kalman-filtered sensor outputs and the equivalent Reference model parameters.

$$q_{res} = \|\hat{q}_{sc} - \hat{q}_{ref}\| \tag{3.1}$$

$$\omega_{res} = \|\hat{\omega}_{sc} - \hat{\omega}_{ref}\| \tag{3.2}$$

3.2 Guidance

If the spacecraft is out of orientation, one possible source of error is the commands received from the guidance system. Latch-up or single event errors could interrupt the guidance commands as they are received by the spacecraft. Processor or clock malfunctions could also cause the received guidance commands to either not execute on time or not execute at all. Finally, if the spacecraft is receiving guidance commands directly from the ground, there is always the possibility of user error that the commands were not defined properly. By checking the echoed attitude and rate commands from the spacecraft versus those estimated by the Reference model, one can determine that if the spacecraft is out of orientation is it simply due to it receiving the wrong commands, e.g. by user error, or is another source to blame, such as a clock reset.

The guidance command residuals defined the norm of the difference between the spacecraft's echoed attitude and rate commands and those as relayed by the Reference model as

$$q^*_{res} = \left\| q^*_{sc} - q^*_{ref} \right\| \tag{3.3}$$

$$\omega^*_{res} = \left\| \omega^*_{sc} - \omega^*_{ref} \right\| \tag{3.4}$$

3.3 Actuators

The actuators are not checked against the Reference model. If the attitude and guidance command residuals are small, then the actuator residuals will be small also. If the attitude and guidance command residuals are biased, such as during a fault event, then it goes without saying that the residuals of spacecraft actuators and Reference model actuators will be correspondingly large; however, this is not indicative of a fault in the actuators. For example, if the spacecraft is desired to have a final attitude quaternion of $q = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$ and due to user error the command uploaded to the spacecraft's

guidance system was $q = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$, a residual defined based on the output of the spacecraft's actuators and the Reference model's actuators would be quite large, as the reference model was commanded to have a final orientation of $q = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$. Though the residual is large, the actuation system is not experiencing a fault event. This would cause confusion and lead to difficulties when training a neural network. For this reason the actuator residuals are defined based on the commanded actuator output and the measured actuator output as reported by the spacecraft.

As will be described in section 4.1 the spacecraft is assumed to be 3-axis controlled and its ADCS consists of a reaction wheel assembly and magnetic torque rods for momentum management. Each reaction wheel has measurable inputs of wheel speed and supply voltage. The magnetic torque rods are assumed to be either "all on" or "all off" and as such their polarity is used to define the basis of a residual.

The actuator residuals are defined as

$$w_{res_i} = \tilde{w}_i - w_i^*, i = 1, 2, 3 \tag{3.5}$$

$$v_{res_i} = \tilde{v}_i - v_i^*, i = 1, 2, 3 \tag{3.6}$$

$$P_{res} = P_i - P_i^*, i = 1, 2, 3 \tag{3.7}$$

where $\tilde{w}_i$, $\tilde{v}_i$, and P_i refer to the measured wheel speed (RPM), wheel voltage (volts) and magnetic torquer polarity for each of the three actuators and w_i^*, v_i^*, and P_i^* refer to the commanded wheel speeds, voltages and torquer polarities.

The measured wheel speeds and voltages are defined further as

$$\tilde{w}_i = w_i + \eta_{w,i}, i = 1, 2, 3 \tag{3.8}$$

$$\tilde{v}_i = v_i + \eta_{v,i}, i = 1, 2, 3 \tag{3.9}$$

The noise terms are defined as having zero mean and being normally distributed as

$$\eta \sim N\left(0, \sigma^2\right) \tag{3.10}$$

with the measurement of the wheel velocity being assumed known to within a 3σ value of 5 RPM and the wheel supply voltage to within 0.1 volts 3σ:

$$\sigma_{wheel} = \frac{5}{3} \tag{3.11}$$

$$\sigma_{V_\omega} = \frac{0.1}{3} \tag{3.12}$$

3.4 Orbit Propagator and Time

The purpose of the propagator checks is to determine whether or not the attitude is out of place due to an error in the propagator. By comparing the propagator's output to the reference model the spacecraft's position in space and time can be confirmed. If it is determined that the propagator is out of sync with the reference model, it would be an likely source of attitude error. Note that propagator errors can be due, for example, to a system reset resulting in a clock or orbit ephemeris errors.

There are a total of four residuals defined based on the spacecraft's orbit propagator. The first residual is defined as the difference between the spacecraft's onboard clock, kept as the Julian date, and the Julian date as kept by the reference model. The time residual is use as differences in the clocks could cause errors with command timing or could be indicative of a reset and is defined formally as

$$JD_{res} = JD_{sc} - JD_{ref} \tag{3.13}$$

The second residual is the difference in the spacecraft's orbital position as reported

by the propagator and the orbital position as estimated by the reference model as seen in equation (3.14).

$$r_{res} = \|\mathbf{r}_{sc} - \mathbf{r}_{ref}\| \tag{3.14}$$

The position of the spacecraft as reported by the propagator is useful in a number of ways. It is used to determine the nadir vector for spacecraft with nadir pointing payloads that do not have other means of determining it, such as a horizon crossing indicator, or it can also be used to model the location of the sun for solar pointing in lieu of a sun sensor. The orbital velocity and orbital elements need not be considered as they so closely coupled with the orbital position vector that if the position is incorrect, the velocity vector and, correspondingly, the orbital elements will be similarly mismatched.

The third and fourth residuals, defined in equations (3.15) and (3.16), are the earth's magnetic field and the unit vector to the sun, both in inertial coordinates, and are used similarly to the position residual as checks on the spacecraft's orbit.

$$B_{res} = \left\|{}^{I}\mathbf{B}_{sc} - {}^{I}\mathbf{B}_{ref}\right\| \tag{3.15}$$

$$S_{res} = \left\|{}^{I}\mathbf{S}_{sc} - {}^{I}\mathbf{S}_{ref}\right\| \tag{3.16}$$

3.5 Sensors

The purpose of checking the sensors is to ascertain if any of the sensors is off in some way. Each of the sensors, the star camera, IMU, magnetometer, and sun sensor, will be checked and verified by comparing the sensor solution to a secondary solution obtained from an alternate sensor. Comparing the two solutions by forming a residual determines whether or not the sensor solution is faulty.

3.5.1 Star Camera

The star camera provides the primary attitude solution as a quaternion, $\hat{\mathbf{q}}_{sc}$. A secondary solution is obtained by employing the TRIAD attitude determination algorithm, which is described by Schuster in [12]. Using the measured solar and magnetic field unit vectors in the body-fixed coordinate system, denoted as ${}^{B}\hat{\mathbf{i}}_S$ and ${}^{B}\hat{\mathbf{i}}_B$, as well as the inertial solar and magnetic field unit vectors as estimated by the onboard propagator, denoted as ${}^{I}\hat{\mathbf{i}}_S$ and ${}^{I}\hat{\mathbf{i}}_B$, the "best" rotation matrix that transforms from one coordinate system to another can be calculated by constructing orthonormal matrices as follows:

First, construct the orthonormal inertial matrix as

$$R = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix} \tag{3.17}$$

where

$$\mathbf{r}_1 = {}^{I}\hat{\mathbf{i}}_S \tag{3.18}$$

$$\mathbf{r}_2 = \frac{{}^{I}\hat{\mathbf{i}}_S \times {}^{I}\hat{\mathbf{i}}_B}{\left\| {}^{I}\hat{\mathbf{i}}_S \times {}^{I}\hat{\mathbf{i}}_B \right\|} \tag{3.19}$$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2 \tag{3.20}$$

Next, construct the orthonormal matrix out of the body-fixed vectors as

$$S = \begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & \mathbf{s}_3 \end{bmatrix} \tag{3.21}$$

where

$$\mathbf{s}_1 = {}^{B}\hat{\mathbf{i}}_S \tag{3.22}$$

$$\mathbf{s}_2 = \frac{{}^B\hat{\mathbf{i}}_S \times {}^B\hat{\mathbf{i}}_B}{\left\| {}^B\hat{\mathbf{i}}_S \times {}^B\hat{\mathbf{i}}_B \right\|} \tag{3.23}$$

$$\mathbf{s}_3 = \mathbf{s}_1 \times \mathbf{s}_2 \tag{3.24}$$

The rotation matrix to transform from the inertial to the body-fixed coordinate system is then simply

$$\begin{matrix} B \\ I \end{matrix} A = SR^T \tag{3.25}$$

Finally, the attitude quaternion can be derived directly from the rotation matrix using Sidi equation A.4.16 [13]:

$$\hat{\mathbf{q}}_{TRIAD} = \begin{bmatrix} q_4 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \pm 0.5\sqrt{1 + a_{11} - a_{22} - a_{33}} \\ {}^{0.25(a_{23}+a_{32})}\!/_{q_4} \\ {}^{0.25(a_{31}+a_{13})}\!/_{q_4} \\ {}^{0.25(a_{12}+a_{21})}\!/_{q_4} \end{bmatrix} \tag{3.26}$$

The residual that describes the validity of the star camera attitude solution is defined as the difference of the TRIAD-derived attitude solution and the star camera's attitude solution:

$$q_{TRIAD,res} = \|\hat{\mathbf{q}}_{TRIAD} - \hat{\mathbf{q}}_{sc}\| \tag{3.27}$$

3.5.2 Rate Sensor

The rate sensor determines the angular rate of the spacecraft. As described by Sidi in [13] it is possible to estimate a spacecraft's angular body rates without the use of rate sensors by using the star camera's attitude quaternion solution. Noting the differentiation of the quaternion elements, the body rates are estimated via the following relation:

$$\begin{bmatrix} \hat{p} \\ \hat{q} \\ \hat{r} \end{bmatrix} = \begin{bmatrix} \hat{q}_{sc,4} & \hat{q}_{sc,3} & -\hat{q}_{sc,2} & -\hat{q}_{sc,1} \\ -\hat{q}_{sc,3} & \hat{q}_{sc,4} & \hat{q}_{sc,1} & -\hat{q}_{sc,2} \\ \hat{q}_{sc,2} & -\hat{q}_{sc,1} & \hat{q}_{sc,4} & -\hat{q}_{sc,3} \end{bmatrix} \begin{bmatrix} \dot{\hat{q}}_{sc,1} \\ \dot{\hat{q}}_{sc,2} \\ \dot{\hat{q}}_{sc,3} \\ \dot{\hat{q}}_{sc,4} \end{bmatrix} \tag{3.28}$$

The residual defined to check the validity of the rate sensor is defined as the norm of the difference between estimated rates from equation (3.28) and the measured body rates from the rate sensor.

$$IMU_{res} = \left\| \begin{bmatrix} \hat{p} & \hat{q} & \hat{r} \end{bmatrix}^T - \begin{bmatrix} \tilde{p} & \tilde{q} & \tilde{r} \end{bmatrix}^T \right\| \tag{3.29}$$

3.5.3 Magnetometer and Sun Sensor

The magnetometer and sun sensor measure the earth's magnetic field and the position of the sun in the body-fixed coordinate system. The sensor measurements are compared to the magnetic field unit vector and the unit vector to the sun as estimated by the propagator model. Noting that the propagator-estimated vectors are first transformed from the inertial to the body-fixed coordinate system, the residuals are defined as

$$Magnetometer_{res} = \left\| {}^{B}\tilde{\mathbf{i}}_B - {}^{B}\hat{\mathbf{i}}_B \right\| \tag{3.30}$$

$$SunSensor_{res} = \left\| {}^{B}\tilde{\mathbf{i}}_S - {}^{B}\hat{\mathbf{i}}_S \right\| \tag{3.31}$$

Chapter 4

Software Model Description

4.1 Attitude Determination and Control System Overview

The functional basis of the attitude determination and control system examined in this research is the Hyperangular Rainbow Polarimeter (HARP) mission conducted by the University of Maryland, Baltimore County, of which the Utah State University Space Dynamics Laboratory is a subcontractor. HARP is a 3-axis stabilized, 3U CubeSat whose primary mission is to keep its imaging platform pointed nadir during science maneuvers. The ADCS consists of a reaction wheel assembly with magnetic torque rods for momentum management, orbit propagator, star tracker, IMU, sun sensor, and magnetometer. The system software model was built using MATLAB/Simulink. Its major components are the attitude control system and spacecraft dynamics, sensor models, orbit propagator, environmental truth models, and disturbance torque models.

4.2 Attitude Control System

The major components of the ACS are the guidance trajectory generator, controller, actuator models, and spacecraft dynamics. Descriptions for each of these components is provided in the following sections.

4.2.1 Guidance Trajectory Generator

The spacecraft's attitude commands are generated via a a guidance trajectory generator based on the eigenaxis slew given in [13]. The purpose of the trajectory generator is determine the time-varying attitude, and angular velocity and acceleration commands in the body-fixed coordinate system.

The first step of the process is to determine the total rotation angle, α, and the eigenvector of rotation, $\mathbf{e}$. The spacecraft is assumed to be in an initial orientation with quaternion $\mathbf{q}_0$ and it is desired to slew to a final orientation, $\mathbf{q}_f$, via a 3-2-1 Euler rotation sequence with Euler angles ϕ,θ, and ψ. Since the rotation matrix can be expressed either in terms of the Euler angles or the attitude quaternion, $\mathbf{q}_f$ can be determined by taking advantage of this equivalency. Abbreviating $\cos(-)$ as c and $\sin(-)$ as s, the rotation matrix corresponding to the Euler rotation sequence can be obtained as

$$A_{321} = \begin{bmatrix} c\phi c\psi & c\theta s\psi & -s\theta \\ -c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi + s\phi s\theta s\psi & s\phi c\theta \\ s\phi s\psi + c\phi s\theta c\psi & -s\psi c\psi + c\phi s\theta s\psi & c\phi c\theta \end{bmatrix} \tag{4.1}$$

The quaternion corresponding to the final desired orientation can then be found from equation (3.26) using the elements of A_{321} .

In order to determine the rotation angle and eigenvector of rotation for the desired slew maneuver, the total attitude transformation must first be calculated as

$$A_{slew} = A(\mathbf{q}_f) A^T(\mathbf{q}_0) \tag{4.2}$$

where $A(\mathbf{q})$ is

$$A(\mathbf{q}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{4.3}$$

With the total attitude rotation calculated, the rotation angle can be found as a function of the trace of the attitude rotation matrix.

$$tr[A_{slew}] = 1 + 2\cos(\alpha) \tag{4.4}$$

Finally, since the eigenvector of rotation is a direct function of the total attitude rotation, it can be calculated directly from (4.2) as

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} (a_{23}-a_{32})/2\sin\alpha \\ (a_{31}-a_{13})/2\sin\alpha \\ (a_{12}-a_{21})/2\sin\alpha \end{bmatrix} \tag{4.5}$$

The next step of the process is to generate a single degree of freedom trapezoidal trajectory. The purpose of the trapezoidal trajectory is to determine the time-varying rotation angle$\alpha(t)$, rotational velocity$\dot{\alpha}(t)$, and rotational acceleration$\ddot{\alpha}(t)$, such that the time-varying quaternion can be computed as a function of the eigenvector of rotation, shown in (4.6).

$$\mathbf{q}(t) = \begin{bmatrix} e_1 \sin\left(\alpha(t)/2\right) \\ e_2 \sin\left(\alpha(t)/2\right) \\ e_3 \sin\left(\alpha(t)/2\right) \\ \cos\left(\alpha(t)/2\right) \end{bmatrix} \tag{4.6}$$

The attitude command, $\mathbf{q}^*$, is then generated as a function of the time-varying quaternion,$\mathbf{q}(t)$, and the spacecraft's initial quaternion, $\mathbf{q_0}$, as shown in equation (4.7)

$$\mathbf{q}^* = Q\mathbf{q}(t) \tag{4.7}$$

where

$$Q = \begin{bmatrix} q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \tag{4.8}$$

Finally, the spacecraft's rotational velocity and acceleration commands, $\omega^*$and α^* respectively, are computed as functions of $\dot{\alpha}(t)$, $\ddot{\alpha}(t)$, and $\mathbf{e}$ as

$$\omega^* = \dot{\alpha}(\mathbf{t})\mathbf{e} \tag{4.9}$$

$$\alpha^* = \ddot{\alpha}\,(t)\,\mathbf{e} \tag{4.10}$$

4.2.2 Plant Dynamics

The basic equation of motion governing the vehicle's dynamics is given by Euler's equation

$$\mathbf{T} = \dot{\mathbf{h}} + \omega \times \mathbf{h} \tag{4.11}$$

Here, $\dot{\mathbf{h}}$ represents the vehicle's total torques in the body frame and is defined as,$\mathbf{h}$ is the vehicle's total momentum in the body frame, including that imparted by the ACS, $\omega$is the vehicle's angular rate in the body frame, and $\mathbf{T}$, the total external torques experienced by the vehicle, can be written

$$\mathbf{T} = \mathbf{T}_c + \mathbf{T}_d \tag{4.12}$$

where $\mathbf{T}_c$ represents the vehicle's attitude control torques and $\mathbf{T}_d$ are the environmental disturbing torques experienced by the vehicle. The spacecraft's plant dynamics can be represented as

$$\mathbf{T} = I\dot{\omega} \tag{4.13}$$

where I is the spacecraft's inertia tensor and$\dot{\omega}$refers to its rotational acceleration. Using estimates for the disturbing torques, Euler's equation can be solved to determine the angular acceleration as

$$\dot{\omega} = I^{-1}\left[\mathbf{T}_d - \dot{\mathbf{h}} - \omega_{\otimes}\,(I\omega + h)\right] \tag{4.14}$$

Noting that $\omega_{\otimes}$ is the cross product matrix of the angular rate vector and is defined as

$$\omega_{\otimes} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{4.15}$$

(4.14) can then be numerically integrated to obtain the vehicle's angular rate. As described in [13] the attitude quaternion can be written as a differential equation in the form of

$$\frac{d}{dt}\mathbf{q} = \dot{\mathbf{q}} = \frac{1}{2}\Omega\mathbf{q} \tag{4.16}$$

where $\mathbf{q}$ is the current attitude quaternion at a given time and

$$\Omega = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \tag{4.17}$$

Integrating equation (4.16) yields the spacecraft's current orientation, $\mathbf{q}(t)$.

4.2.3 Controller Selection

The purpose of the controller is to stabilize the vehicle's plant dynamics. It does this by commanding the actuators to generate a control torque such that it zeros the acceleration of the spacecraft caused by the disturbing torques. A proportional-integral-derivative (PID) control law, shown in equation (4.18), was chosen for the HARP spacecraft due to its robustness and the ability of the added integral mode to reduce the disturbance error.

$$T_c = K_d(\omega^* - \omega) + 2K_p\mathbf{q}_e q_{e_4} + K_i \int 2\mathbf{q}_e q_{e_4} dt + I\alpha^* + \omega_{\otimes}(I\omega + \mathbf{h}) + \mathbf{T}_d \tag{4.18}$$

The error, $\mathbf{q}_e$, is defined as

$$\mathbf{q}_e = Q\mathbf{q_s} \tag{4.19}$$

where Q is the same as (4.8) and $\mathbf{q}_s$ is

$$\mathbf{q}_s = \begin{bmatrix} -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix}^T \tag{4.20}$$

The proportional, integral, and derivative gains, K_p, K_i, and K_d, respectively, were chosen based on the deadbeat design of the system defined as

$$Is^3 + K_d s^2 + K_p s + K_i = 0 \tag{4.21}$$

According to Dorf and Bishop in [14] this third order system can be represented as

$$s^3 + 1.9\omega_n s^2 + 2.2\omega_n^2 s + \omega_n^3 = 0 \tag{4.22}$$

where ω_n refers to the natural frequency of the system. Isolating the highest order term of (4.21) and equating with (4.22) yields

$$s + \frac{K_d}{I}s^2 + \frac{K_p}{I}s + \frac{K_i}{I} = s^3 + 1.9\omega_n s^2 + 2.2\omega_n^2 s + \omega_n^3 \tag{4.23}$$

At this point the selection of the controller gains is dependent only upon the choice of the natural frequency and the spacecraft's inertia tensor. For a deadbeat design with $\omega_n = 1$ the gains become

$$K_p = 2.2I \tag{4.24}$$

$$K_i = I \tag{4.25}$$

$$K_d = 1.9I \tag{4.26}$$

4.2.4 Actuator Models

The model's actuation system is based on HARP's ADCS, which uses the off-the-shelf XACT system developed by Blue Canyon Technologies. It consists of a trio of reaction wheels mounted along each of the body axes as well as magnetic torque rods (MTR) for momentum management. Per Blue Canyon Technologies' specifications, the momentum wheels have a maximum momentum capability of 18 Nms and torque generation of 0.6 mNm. They are modeled in the Laplace domain as

$$\frac{h_{wheel}(s)}{h_{wheelcmd}(s)} = \frac{1}{\tau_w s + 1} \tag{4.27}$$

HARP's reaction wheels have a maximum wheel momentum of approximately 18 mNms. This corresponds to a maximum wheel speed of 6,000 RPM at 12 volts. The commanded wheel speed is estimated based on the wheel's moment of inertia as

$$\omega_{wheelcmd} = \frac{h_{wheelcmd}(s)}{I_w} \tag{4.28}$$

The actual measured wheel speed of the each of the reaction wheels is assumed to be known only to within ±5 RPM and is modeled as

$$\omega_{wheel} = \frac{h_{wheel}(s)}{I_w} + \eta_{wheel} \tag{4.29}$$

where η_{wheel} is given in equation (3.11).

The voltage draw of the reaction wheels was estimated based on the quadratic function shown in equation (4.30) to simulate increasing power requirements as they spun faster, up to a maximum of 12 volts at 6,000 RPM. The measured voltage was assumed known to within ±0.1 volts.

$$V_\omega = 2.22E-7\omega_{wheel} + 6.67E-4\omega_{wheel} + \eta_{V_\omega} \tag{4.30}$$

with η_{V_ω} given in equation (3.12).

4.3 Attitude Determination System

The XACT attitude determination and control system used on HARP comes with a suite of attitude determination sensors including a star tracker, inertial measurement unit, magnetometer, and sun sensor, as well as a Kalman filter to process sensor noise. All sensors have a sampling rate of 50 Hz with the exception of the star tracker, which is capable of a 5 Hz attitude solution.

4.3.1 Star Camera

A star camera is used as the primary means of attitude determination on HARP. The spacecraft's attitude quaternion can be determined via comparison of vectors measured in the body-fixed coordinate system with known vectors in the inertial coordinate system. The basic model is given as

$$\widetilde{\mathbf{b}}_i = A\mathbf{r}_i + \nu_i \tag{4.31}$$

where the unit vectors measured in the body-fixed coordinate system are given by $\widetilde{\mathbf{b}}_i$, the known unit vectors in the inertial system given by $\mathbf{r}_i$, and the errors represented by ν_i. The rotation between the inertial and body-fixed coordinate systems is given by the matrix, A, and is determined by integrating equation (4.16) and applying the result to equation (4.3). The noise is modeled as a random variable with variance

$$\sigma^2 = 0.01\,\mathrm{deg}^2 \tag{4.32}$$

4.3.2 Inertial Measurement Unit

The spacecraft's angular rate is measured using an inertial measurement unit (IMU) and is modeled as

$$\tilde{\omega} = \omega + \beta + \eta_{\mathbf{u}} + \eta_{\nu} \tag{4.33}$$

where $\widetilde{\omega}$ denotes the measured angular rate, β, η_u and η_ν represent the sensor bias, bias noise, and sensor noise respectively and have values of

$$\beta = 0.1\,{}^{\circ}\!/_{\mathrm{hr}} \tag{4.34}$$

$$\eta_u = \sqrt{10}E - 10\,{}^{\mathrm{rad}}\!/_{\mathrm{s}^{3/2}} \tag{4.35}$$

$$\eta_\nu = \sqrt{10}E - 7\,{}^{\mathrm{rad}}\!/_{\sqrt{\mathrm{s}}} \tag{4.36}$$

4.3.3 Kalman Filter

Since the measurements made by the star tracker and IMU are corrupted by noise and bias, they must first be filtered in order to obtain estimates of the true attitude and angular rate. This is accomplished via a discrete time extended Kalman filter (EKF) as outlined in [15]. The discrete time EKF is the preferred filtering method as the discrete time propagation of the state and covariance can accommodate the difference in sampling rates between the star tracker (5 Hz) and IMU (50 Hz).

4.3.4 Magnetometer and Sun Sensor Models

The magnetometer and sun sensor models follow the same general format as (4.33), but is modified as

$$\tilde{\mathbf{x}} = \mu\mathbf{x} + \beta + \eta \tag{4.37}$$

$\widetilde{\mathbf{x}}$ represents the measured quantity, which is either the earth's magnetic field vector or the unit vector to the sun in the spacecraft's body-fixed coordinate system. $\mathbf{x}$ is the truth vector for either the magnetic field or the unit vector to the sun. $\mu$is an uncertainty parameter. Since the environmental truth models of both the reference and spacecraft models are identical, and the spacecraft model is meant to simulate actual telemetry, this parameter is added in order to introduce the uncertainty that would exist between the reference model and actual spacecraft telemetry. The uncertainty parameter is modeled as

$$\mu = \begin{bmatrix} \cos \sigma n_x \\ \cos \sigma n_y \\ \cos \sigma n_z \end{bmatrix} \tag{4.38}$$

where σ is the assumed maximum uncertainty in the quantity being measured and n_i is a random number between zero and one. The maximum uncertainty of the magnetometer was assumed to be $\sigma = 10^\circ$while the sun sensor was assumed to have maximum uncertainty of $\sigma = 5^\circ$.

β represents the bias in the measurements. This is mainly a concern with the magnetometer. It is assumed that the spacecraft is not magnetically clean and that there exists a residual magnetic field. In lieu of available estimates or measurements of the HARP spacecraft, the residual magnetic field was assumed to be

$$\beta = \begin{bmatrix} 100 \\ 50 \\ 75 \end{bmatrix} nT \tag{4.39}$$

η refers to the noise present in the measured unit vectors and is modeled as per equation (3.10) with

$$\sigma_{mag} = 0.01 \tag{4.40}$$

$$\sigma_{sun} = 0.0005 \tag{4.41}$$

4.4 Truth Models

In order to accurately characterize the spacecraft's operating environment, models simulating the orbital dynamics and earth's magnetic field have been included. The earth's heliocentric planetary state vector is also calculated in order to determine when the spacecraft is in eclipse and is based on the method outlined in [16].

4.4.1 Orbital Dynamics

The spacecraft's orbit mechanics include the effects of the earth's oblateness and are modeled according to Curtis in [16]. The basic equation of two-body motion is given in (4.42). Note that $\hat{\mathbf{u}}_r$ refers to the spacecraft's radial unit vector.

$$\ddot{\mathbf{r}} = \frac{-\mu}{r^2}\hat{\mathbf{u}}_r \tag{4.42}$$

In order to adequately capture the earth's oblateness effects, 4.42 is modified by adding the disturbing acceleration due to the second zonal harmonic, J_2, as

$$\ddot{\mathbf{r}} = \frac{-\mu}{r^2}\hat{\mathbf{u}}_r + \mathbf{p} \tag{4.43}$$

The disturbing acceleration can be written in terms of the spacecraft's radial, traverse, and normal unit vectos as

$$\mathbf{p} = p_r\hat{\mathbf{u}}_r + p_\perp\hat{\mathbf{u}}_\perp + p_h\hat{\mathbf{h}} \tag{4.44}$$

The components of $\mathbf{p}$ themselves are functions of the second zonal harmonic, J_2, the spacecraft's radial position vector, r, the earth's radius, R, as well as the argument of perigee, ω, the inclination, i, and true anomaly, θ, as shown below.

$$p_r = -\frac{\mu}{r^2}\frac{3}{2}J_s\left(\frac{R}{r}\right)^2\left[1 - 3\sin^2 i \sin^2(\omega+\theta)\right] \tag{4.45}$$

$$p_\perp = -\frac{\mu}{r^2}\frac{3}{2}J_s\left(\frac{R}{r}\right)^2 \sin^2 i \sin 2(\omega+\theta) \tag{4.46}$$

$$p_h = -\frac{\mu}{r^2}\frac{3}{2}J_s\left(\frac{R}{r}\right)^2 \sin 2i \sin(\omega+\theta) \tag{4.47}$$

Integrating 4.43 once yields the spacecraft's velocity vector. Integrating once more yields position.

4.4.2 Magnetic Field Model

The International Association of Geomagnetism and Aeronomy (IAGA) provides a model of the earth's magnetic field known as the International Geomagnetic Reference Field (IGRF). The earth's magnetic field can be modeled as a spherical harmonic expansion, the coefficients of which are provided by the IAGA.

4.5 Environmental Disturbance Models

4.5.1 Gravity Gradient

The largest external disturbing torque experienced by the spacecraft is the gravitational moment due to the gravity gradient. Within a magnetic field, an asymmetric body will tend to align the axis with the smallest moment of inertia to the direction of the field [13]. As such, for satellites in low earth orbit, this disturbance cannot be neglected. The torque due to the gravity gradient for each of the spacecraft's body axes is a function of the spacecraft's orbital radius, moments of inertias and Euler angles as

$$\mathbf{T}_{GG} = \frac{3\mu}{2R_0^3} \begin{bmatrix} (I_{zz} - I_{yy}) \sin 2\phi \cos^2 \theta \\ (I_{zz} - I_{xx}) \sin 2\theta \cos \phi \\ (I_{xx} - I_{yy}) \sin 2\theta \sin \phi \end{bmatrix} \tag{4.48}$$

where R_0 is the orbital radius and ϕ and θ refer to the roll and pitch angles respectively.

4.5.2 Magnetic Field

Due to the inability of spacecraft designers to design a spacecraft that is absolutely magnetically neutral, there is usually a residual dipole moment that is created by the spacecraft's internal components. The interaction of this dipole moment with the earth's magnetic field imparts a torque on the spacecraft. This torque can be modeled as

$$T_m = D\mathbf{B} \tag{4.49}$$

where D is the spacecraft's residual dipole moment in units of $amps \cdot turns \cdot m^2$and $\mathbf{B}$ is the earth's magnetic field vector in $Tesla$ [17].

4.5.3 Aerodynamic Drag

Due to HARP's low orbital altitude of 600 km, the spacecraft will experience an external torque due to atmospheric drag. The torque due to drag was modeled per [17] as

$$T_a = F\left(c_{pa} - c_g\right) \tag{4.50}$$

The center of pressure and center of gravity are denoted as c_{pa} and c_g respectively and are estimated from CAD models. F is the force term and is defined further as

$$F = \frac{\rho C_D A V^2}{2} \tag{4.51}$$

The atmospheric density, ρ, is estimated based on the MSISE-90 model of earth's upper atmosphere. The drag coefficient, C_D, was assumed to be 2.5. The spacecraft's surface area,

A, was estimated based on CAD models. Finally, V refers to the spacecraft's orbital velocity and is found by integrating equation (4.43).

Chapter 5

Training the Neural Network

5.1 What is a Neural Network?

According to Haykin [18] a neural network can be described as "a massively parallel distributed processor made up of simple processing units ['neurons'], which has a natural propensity for storing experiential knowledge and making it available for use." It is able to gain knowledge by acquiring it through its environment via a learning process and able to store that knowledge using synaptic weights. The synaptic weights are analogous to the gains of a linear adaptive filter and are modified during the learning process, much as filter gains are, until the network is able to achieve its desired performance.

One of the great advantages of using a neural network in FDI is its ability to perform input-output mapping. Using input-output mapping a neural network is able to modify its synaptic weights by applying training samples. The training samples consist of an input signal and a desired response. During training the synaptic weights are modified in order to minimize the error between the desired response and actual response of the network. The input-output training samples act as a "teacher" for the neural network and they tell the network how it should respond to a given example from the training set [18].

5.2 Neural Network Training Set

To remind the reader, the purpose of the residuals is to provide the learning space from which a neural network can be trained to detect and isolate fault events. A neural network is trained by being given examples from a training set of data, each example consisting of a set of inputs with a corresponding desired output. It is desired that the residuals defined by the attitude determination and control system form the input portion of the training set, and therefore a vector of residuals is defined as

$$Inputs = \Big[\; q_{res} \quad \omega_{res} \quad q^*_{res} \quad \omega^*_{res} \quad w_{res_{3\times1}} \quad v_{res_{3\times1}} \quad P_{res_{3\times1}} \quad JD_{res} \quad r_{res} \quad B_{res}$$
$$S_{res} \quad q_{TRIAD,res} \quad IMU_{res} \quad Magnetometer_{res} \quad SunSensor_{res} \; \Big]^T_{21\times n} \tag{5.1}$$

The input vector has dimensions 21 x n, where n is the number of examples in the training set. A Matlab script was used to generate this vector and is given in appendix B.1.

Similarly, a target vector must also be defined to complete the training set. The purpose of the target vector is to tell the neural network how it should respond to a given input. As the inputs are defined as the residuals of the attitude determination and control system, the outputs correspond to the possible faults that could be present in the ADCS. Each element of the target vector corresponds to a single source of error and has a default value of 0. When a fault is detected by the neural network, it changes the element corresponding to the fault to a 1.

The input categories of figure 1.1 are consulted once again to determine the sources of error in the ADCS to define the target vector. The external inputs to the ADCS are comprised of such quantities as the true position of the sun, the earth's magnetic field, the spacecraft's true attitude and angular rate as well as guidance commands from the ground. As the latter is the item that is most controllable, a fault state is defined that corresponds to either improperly defined ADCS guidance commands, e.g. by user error, or guidance commands that are received improperly due to another unknown source.

The ADCS sensors are responsible for taking the measurements that allow for accurate attitude determination and are critical to its operations. Fault states are therefore defined that correspond to each of the attitude determination sensors. Such sensors can include star trackers and horizon crossing indicators. This thesis assumes an ADCS with four such sensors: star tracker, IMU, magnetometer, and sun sensor with fault states reserved for each.

Like the sensors the ADCS software is equally critical to the operation of the system.

The software is responsible for issuing attitude commands, reaction wheel commands, or managing the momentum storage of the control system. It is also comprised of the orbit propagator, which is responsible for propagating the spacecraft's orbital position and estimating the position of the sun and earth's magnetic field in lieu of GPS or sensor measurements. As seen in figure 1.1 most of the software is situated downstream from two main inputs: the orbit propagator and the guidance commands. A fault in either of these two components would manifest itself, for instance, in the controller commands to the reaction wheels, or other downstream component. As the guidance commanding is handled elsewhere, an element in the fault vector is then reserved based on the performance of the onboard propagator. Also, as the spacecraft's position and targetting capabilities are also time-dependent, a second fault corresponding to the spacecraft's onboard clock in the form of the Julian date, is reserved as well.

The final source of inputs in the control actuators: the reaction wheels and magnetic torque rods. Each reaction wheel has two elements of the target vector. One element corresponds to wheel speed, the other to the power being supplied to the wheel. The reaction wheels have the separate elements defined to better differentiate the cause of error, whether it is the wheel itself malfunctioning or if the cause is due to a lack of nominal power. The magnetic torque rods each have a single element corresponding to an on/off state and whether the polarity is correct.

Sixteen sources of error have been defined: 9 corresponding to the reaction wheels and torquers, 2 corresponding to the propagator and clock, 1 corresponding to the commanding of the ADCS, and 4 for the attitude determination sensors. A 16 element vector corresponding to each of these errors is defined as

$$Targets = \Big[\; Speed - Whl_i \quad Power - Whl_i \quad Polarity - MTR_i \quad Cmds \quad Clock \quad Speed - Whl_i \quad Power - Whl_i \quad Polarity - MTR_i \quad Cmds \quad Clock \; \Big]^T_{16\times n} \tag{5.2}$$

with each element having a value of 0 for nominal state or 1 when a fault has been detected.

The target vector has dimensions 16 x n, where n is the number of examples in the training set. A Matlab script was used to generate this vector and is given in appendix B.2.

5.3 Proof of Concept

Prior to training a neural network using the input and target vectors defined in equations (5.1) and (5.2), a reduced set of inputs and targets was defined in order to demonstrate proof of concept. The purpose of performing a proof of concept exercise was to demonstrate that a neural network could indeed discern a pattern within a set of residuals defined by an ADCS and determine a fault. A simple reaction wheel-controlled, single degree of freedom spacecraft was considered. Residuals were defined to compare the spacecraft's attitude to that estimated by a reference model, and the measured wheel speed and voltage were compared to the commanded wheel speed and voltage. Two possible sources of error were assumed for the system –wheel speed, to check for mechanical failure, and wheel voltage to assure nominal power was available. The associated input and target vectors used to train the neural network are given in equations (5.3) and (5.4).

$$\mathbf{X} = \begin{bmatrix} \theta_{res} & w_{res} & v_{res} \end{bmatrix}^T \tag{5.3}$$

$$\mathbf{Y} = \begin{bmatrix} WheelSpeed & WheelVoltage \end{bmatrix}^T \tag{5.4}$$

A pattern recognition neural network was constructed using the Matlab neural net toolbox. Pattern recognition is described by Haykin as "the process whereby a received pattern or signal is assigned to one of a prescribed number of classes" [18]. The length of the target vector, $\mathbf{Y}$, represents the number of classes that the input vector, $\mathbf{X}$, will be divided into, which in this instance is two. An example set was constructed by commanding the spacecraft to perform a 180°slew maneuver at $t = 60$ seconds. At $t = 145$ seconds the output of the wheel was limited to 25% for10 seconds in order to simulate a wheel malfunction before being allowed to return to normal operation. During the maneuver the residuals defined by (5.3) were monitored and are presented in figure 5.1. It was assumed

that the wheel speed was known to within three standard deviations of $\pm 5RPM$. When the measured wheel speed exceeded this threshold, the element of $\mathbf{Y}$ corresponding to the speed of the reaction wheel was switched from 0 to 1 to indicate a fault state.

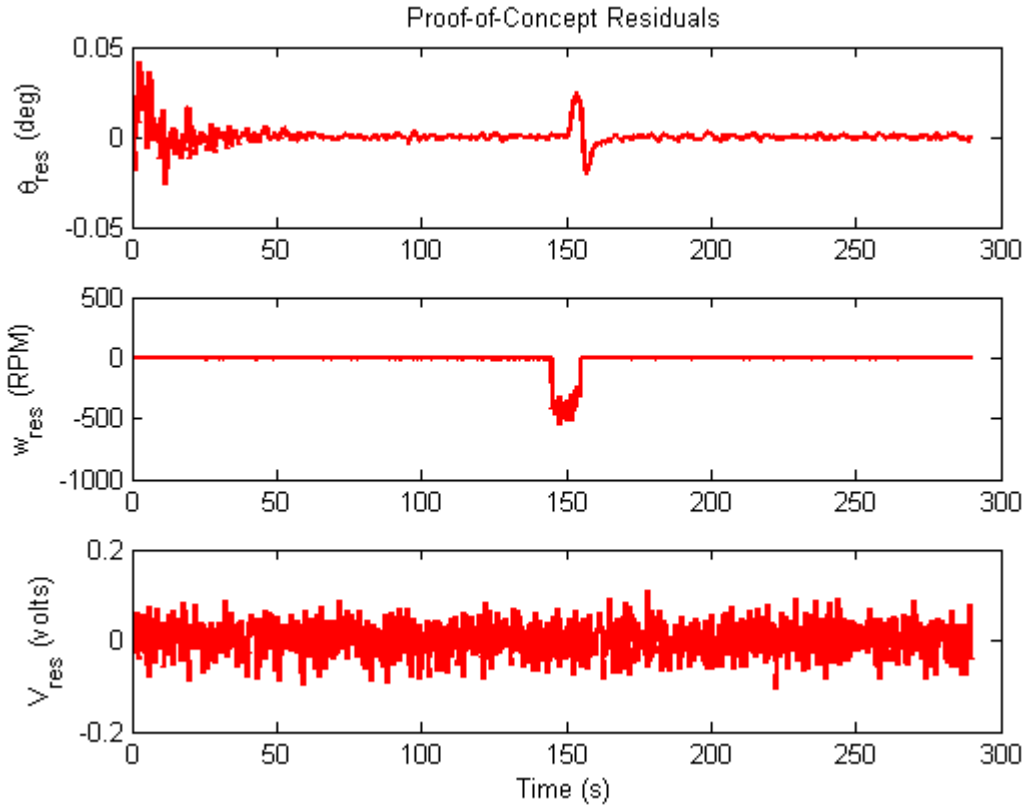

Fig. 5.1: Attitude, wheel speed, and wheel supply voltage training residuals

As can be seen from the residuals, at 145 seconds there is a sudden bias in the wheel speed as the wheel's torque authority is limited. There is a small aberration in the attitude residual as the wheel is unable to provide the necessary output during the fault event. The bias in the attitude and wheel speed residuals persist until 155 seconds, at which time the wheel returns to normal operation and the attitude is able to readjust. The reaction wheel supply voltage residual remains unchanged from a nominal value during the fault event, which indicates that the ADCS had no trouble supplying the power required by the wheel.

Applying this data to the neural network yields the results shown in figure 5.2. From the figure it is easily seen that the neural network is reporting normal wheel operation

and nominal power available until $t = 145s$ at which point it successfully detects the fault induced in the reaction wheel. It is worth noting that based on the results shown in figures 5.1 and 5.2 that although a fault was detected in the reaction wheel, the residuals and the neural network output all returned to nominal values and is indicative of the spacecraft successfully completing the desired maneuver.

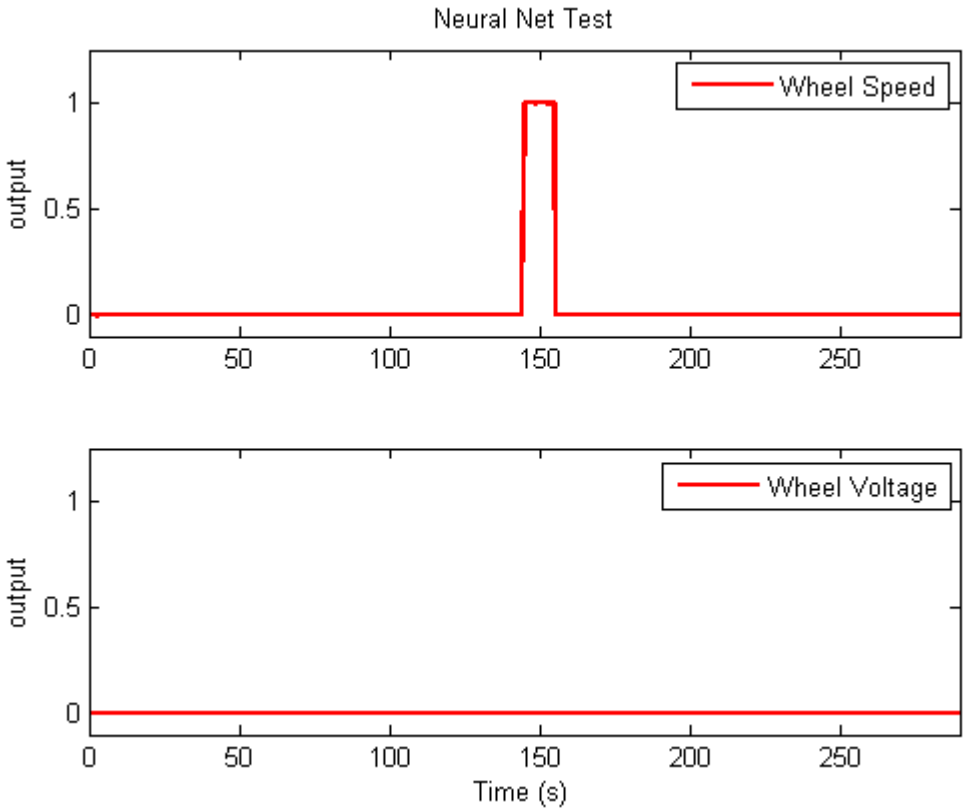

Fig. 5.2: Fault detected at $t = 145s$

5.4 Generating the Training Data

Based on the example given in section 5.3, a pattern recognition neural network is shown to be able to discern a fault within a reduced set of residuals for the specific case of a reaction wheel failure. To detect faults relative to the entire ADCS the training space is expanded to include all of the residuals described in chapter 3 and is given by equation (5.1). The target vector, which tells the neural network what pattern of residuals corresponds to

which failure, is given by equation (5.2).

For the neural network to be able to recognize failures in the ADCS it must first be trained to recognize what patterns of residuals correspond to which component failures, and what patterns of residuals correspond to normal behavior. As such the neural network will require an extensive number of example sets that are comprised of residuals generated for both fault and nominal operation scenarios that encompass the entire ADCS, that is, all of the components of the target vector describe in equation (5.2). The training data were generated in the following manner:

First, determine which component the fault will be injected into, for example, the amount of power available to the reaction wheels. Next, determine a set of maneuvers for the spacecraft to perform that will generate residuals over a wide range. The intent of this is to attempt to bound the problem and provide the neural network with training examples that cover as wide a range as possible. After defining the maneuvers, the next step is to generate simulated telemetry using both the reference and fault software models and to use that telemetry to form the residual vector. Note that the telemetry should be sampled at an appropriate rate to mimic as closely as possible the data that would be received from the spacecraft. In the case of HARP, a full attitude solution is available at a rate of 5 Hz. Finally, the appropriate element of the residual vector is then thresholded such that when the threshold is exceeded the corresponding element of the target vector is incremented from 0 to 1. The values of the thresholds are determined either empirically or based on the known standard deviation of a measured quantity, such as sensor noise. Following this general procedure, a library of training examples is generated for the ADCS on a component-by-component basis.

5.4.1 Nominal Training Examples

A nominal training set of examples, corresponding to normal operation of the ADCS, was generated for four maneuvers: a stationkeeping maneuver where the ADCS was commanded to keep the spacecraft nadir pointing, and 180° maneuvers for roll, pitch, and yaw. Since no fault was present during any of these cases, no thresholding of the residuals was

required. The residuals for the nominal stationkeeping maneuver are shown in figure 5.4. Quantities that would normally be thresholded are shown with their thresholds given as hashed lines.

The target vector, which describes to the neural network the output desired for a given input, is shown graphically in figure 5.3. Since no faults were defined for either the nominal stationkeeping or the roll, pitch, or yaw maneuvers, the desired output of each component is 0.

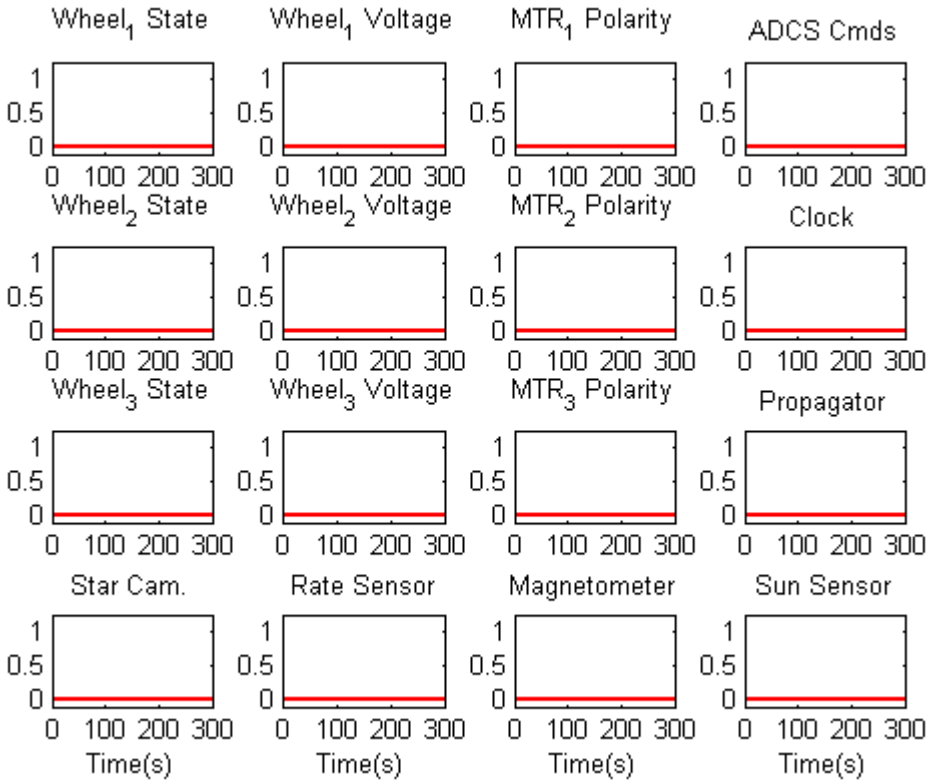

Fig. 5.3: Neural network output for a nominal hold maneuver

5.4.2 Training for Reaction Wheel Faults

Training examples were generated for each of the reaction wheels that covered two different faults: a sudden limitation of available torque authority and a limiting of the available power. 180° maneuvers for roll, pitch, and yaw, as well as stationkeeping, were defined in order to generate as large a residual as possible for each of the reaction wheels.

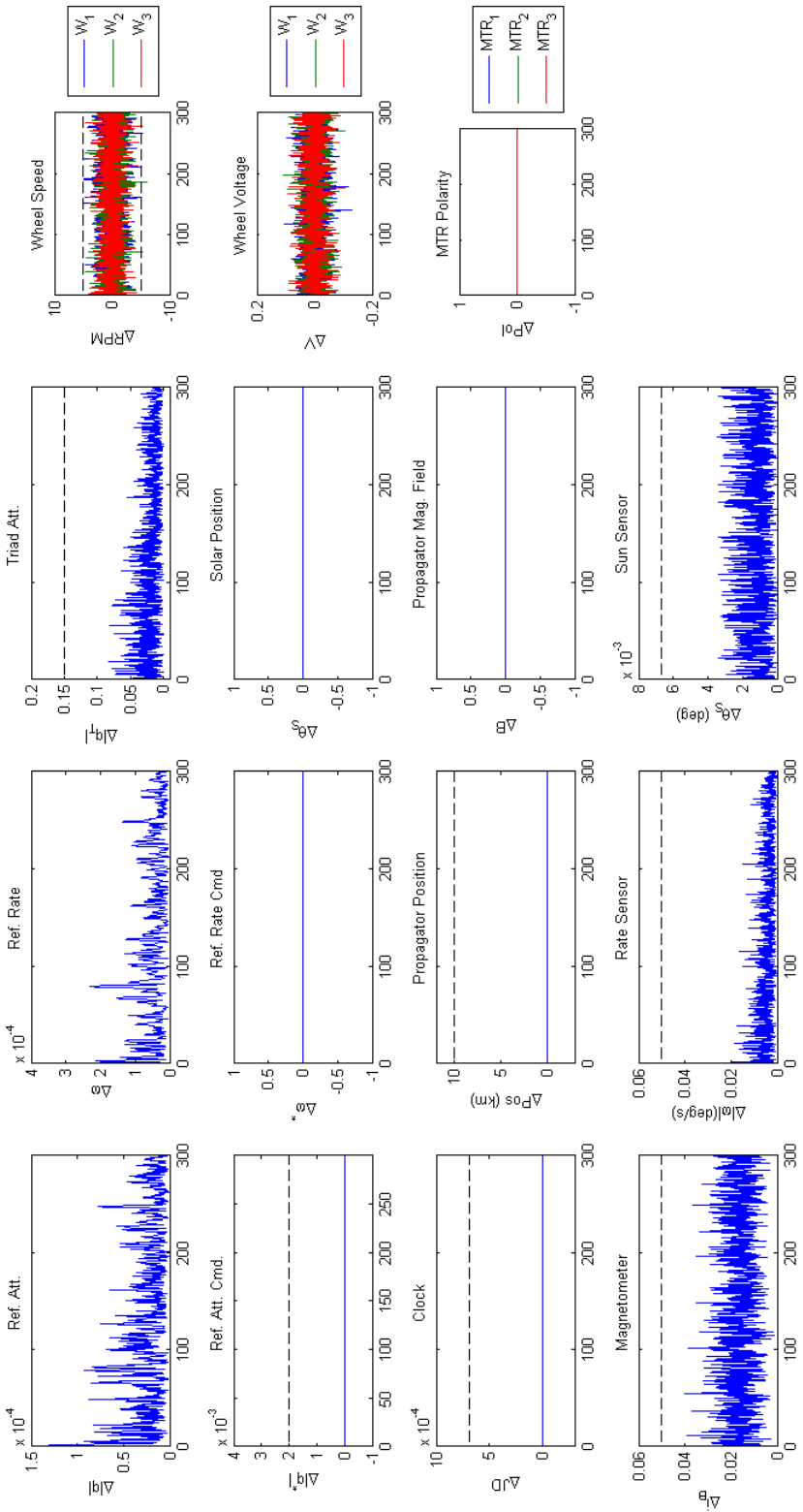

Fig. 5.4: The residuals for a nominal stationkeeping maneuver

For each of the faults and maneuvers listed previously, the available torque authority or power were limited to 75%, 50%, 25%, and 0% of that which was requested by the system. In the case of limiting the torque authority, it was assumed that a mechanical malfunction was to blame. As such, a ten second duration was assumed for each fault event, during which an extra noise term with power equal to 1% of the total torque capability of the wheel was added to the output. This extra noise was omitted during the simulated power failures. The lack of available power was not assumed to be a singular event and therefore had no fixed duration and was allowed to persist for the entirety of the simulated fault event. For each failure mode it was assumed that the speed of the wheel was known to within three standard deviations of 5 RPM and each wheel residual was thresholded at five standard deviations. Figure 5.5 shows residuals that are typical for a mechanical failure of a reaction wheel.

The maneuver associated with these residuals is a 180° maneuver about the pitch axis. Between $t = 145$ seconds and $t = 155$ seconds the output of reaction wheel 2 was limited to 25% of what was requested by the controller. During this time the residual of reaction wheel 2 experiences a very large, sudden bias as it is no longer capable of providing the desired torque commanded by the system. The reference attitude and reference angular rate residuals also experience biases during this time. After $t = 155$ seconds the wheel is allowed to return to normal operation and the spacecraft is able to complete the maneuver as is denoted by the residuals returning to nominal values. The residual of reaction wheel 2 was thresholded to generate the desired neural network output shown in figure 5.6.

Limiting the power available to the reaction wheels was conducted in a manner similar to the mechanical malfunction described above, with the exception being that the noise term associated with the mechanical malfunction was omitted and that once the fault was injected it was allowed to persist indefinitely. Figures 5.7 and 5.8 show the residuals and desired network response of a typical power failure event.

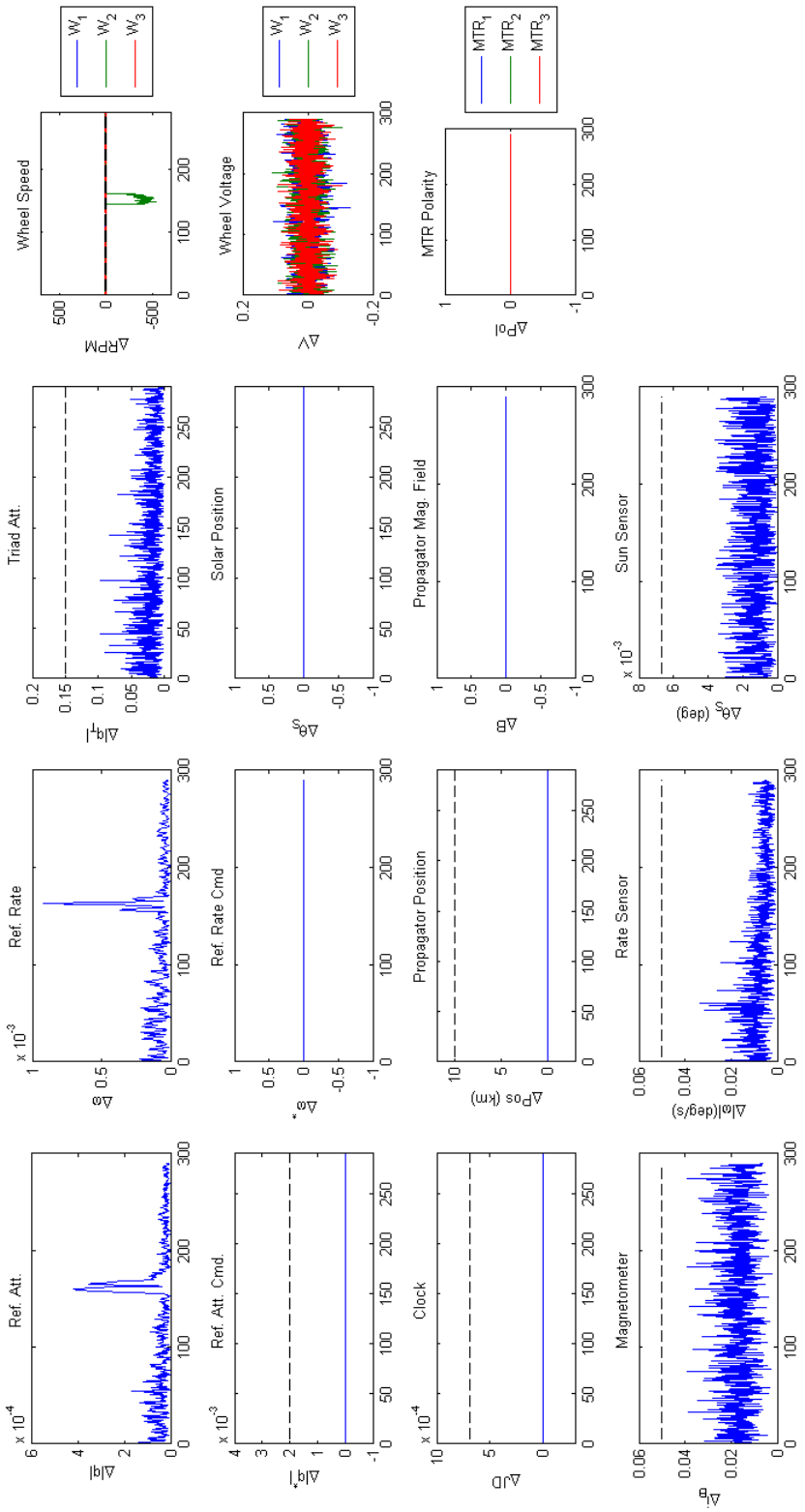

Fig. 5.5: The residual biases associated with a typical reaction wheel mechanical failure

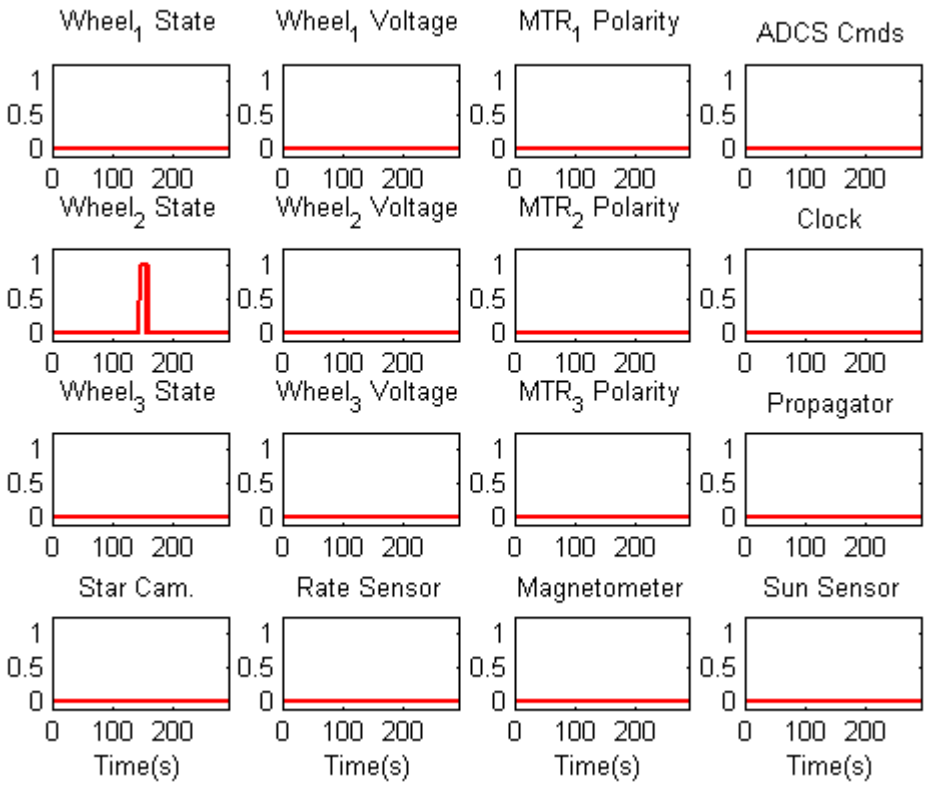

Fig. 5.6: Neural network output associated with a failure in a reaction wheel

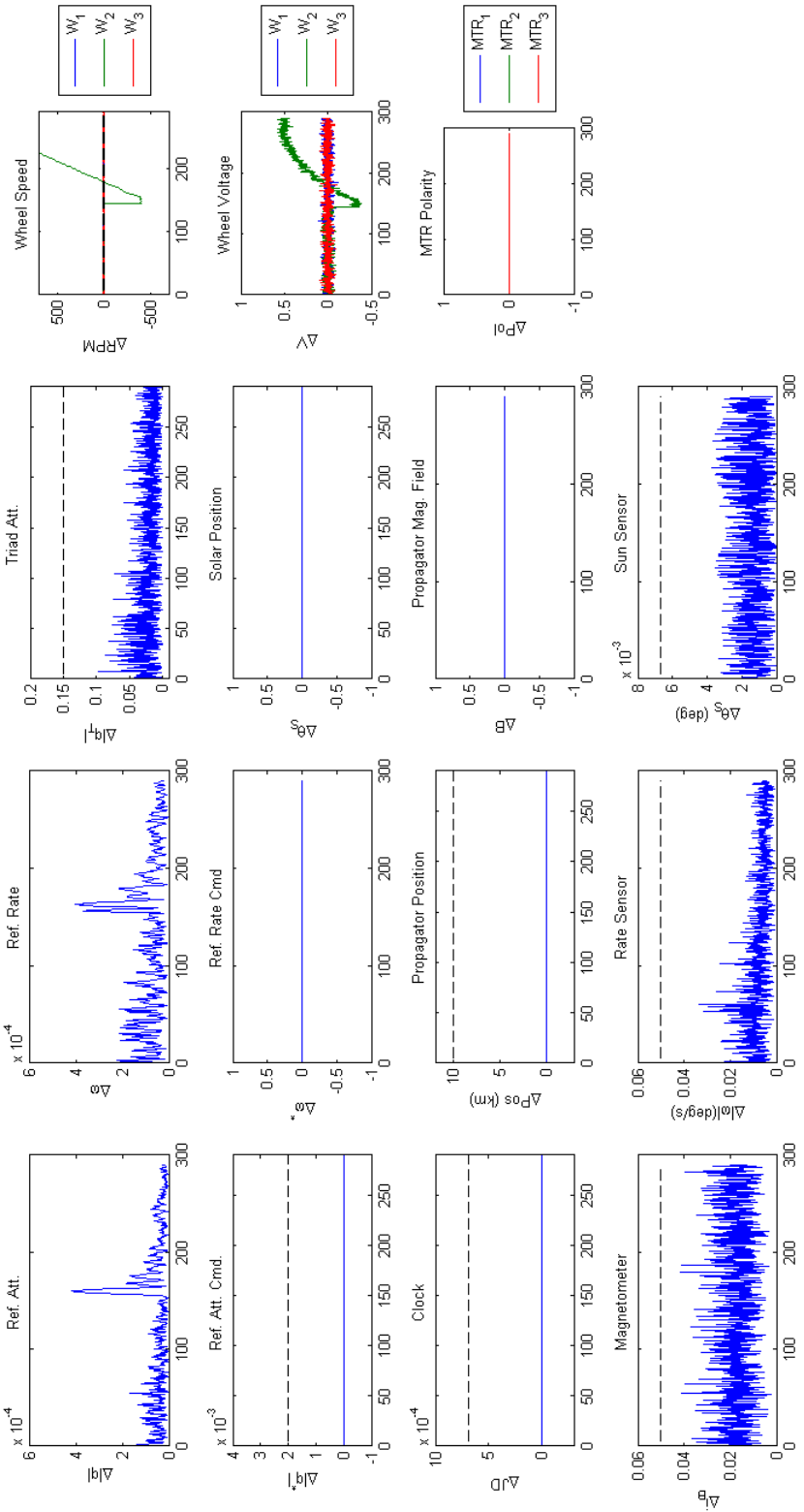

Fig. 5.7: Residuals typical of a wheel power failure

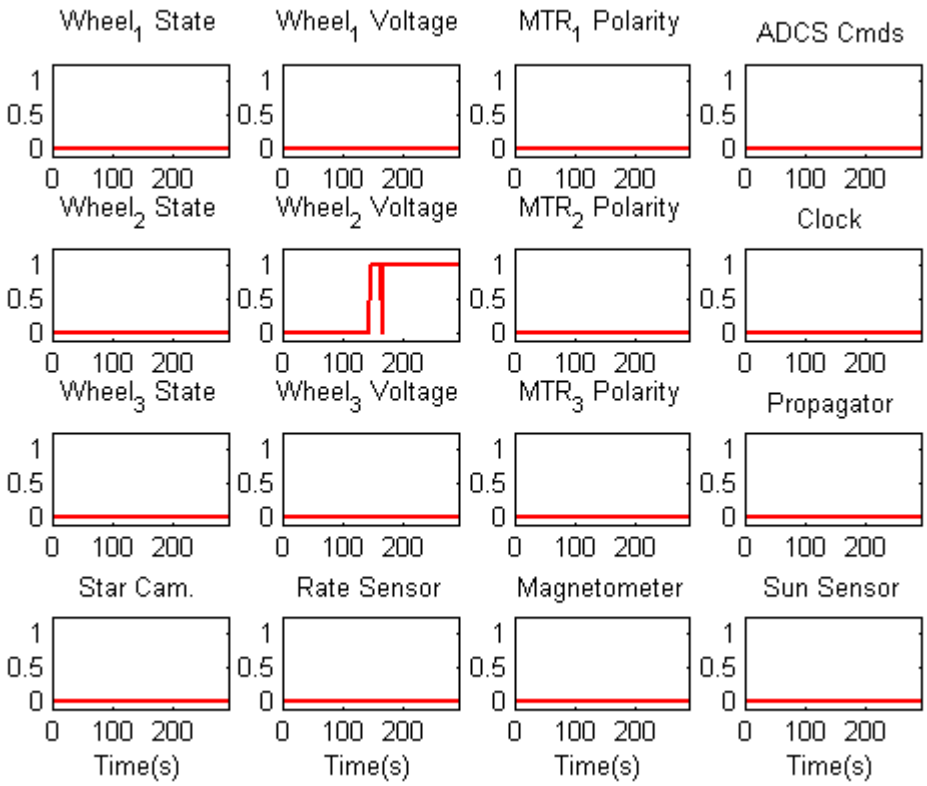

Fig. 5.8: Desired neural network output for a power failure in reaction wheel 2

5.4.3 Training for the Magnetic Torque Rods

Figures 5.9 and 5.10 give the residuals and output associated with the training of the magnetic torque rods. Training examples for the magnetic torque rods were limited to determining whether or not they fired as commanded with the proper polarity. For each of the three magnetic torque rods 180° maneuvers for roll, pitch, and yaw were simulated. To simulate constant desaturation of the reaction wheels the MTRs were commanded to fire when the wheels reached 8% of their momentum storage capability. When the torque rods were commanded to fire their polarity was reversed in order to generate the residual. Since the residual of the MTRs is based only upon whether or not the polarity is correct, it was thresholded such that any non-zero value would be indicative of a fault.

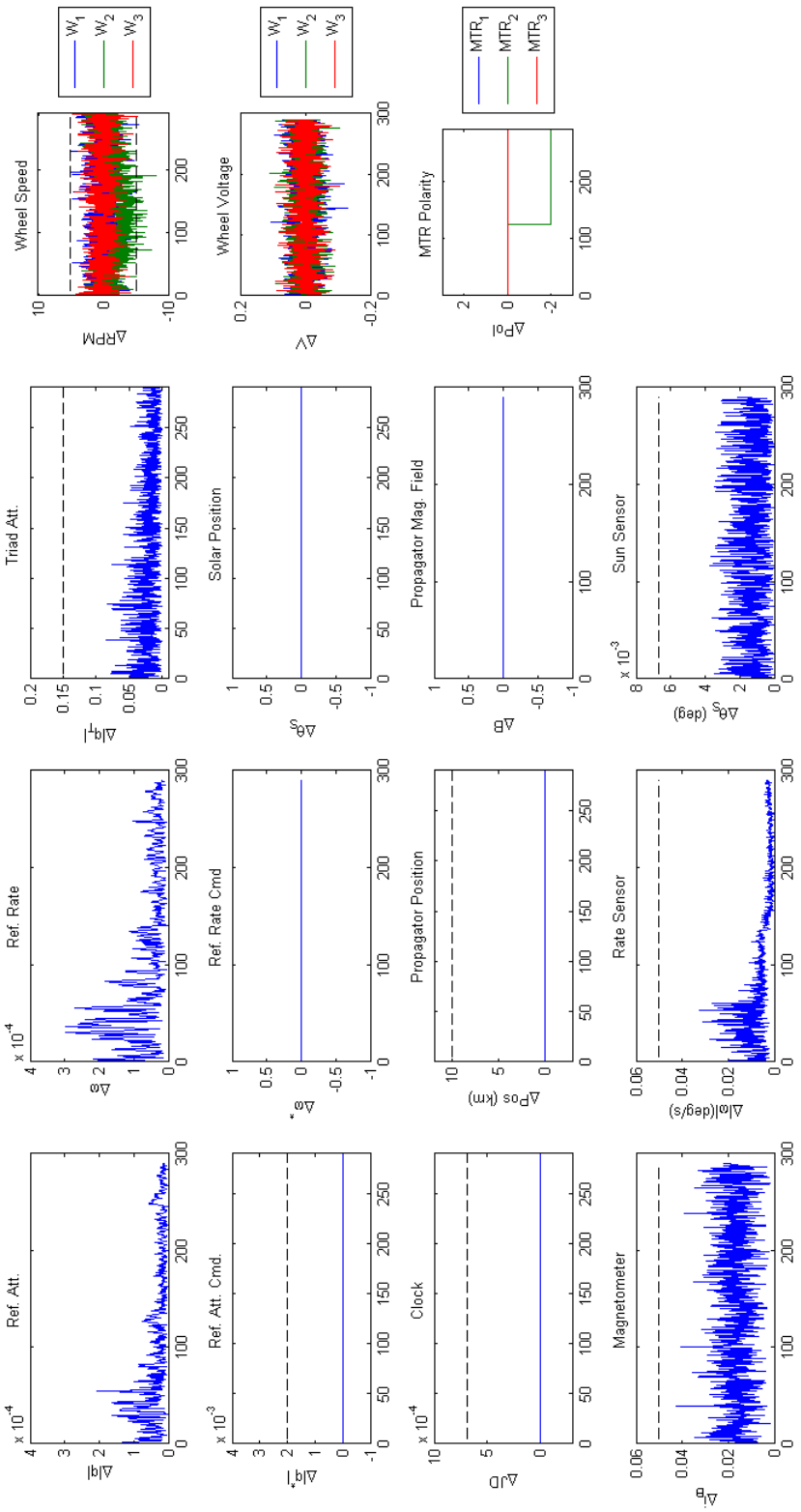

Fig. 5.9: The residuals associated with reversed firing of a magnetic torque rod

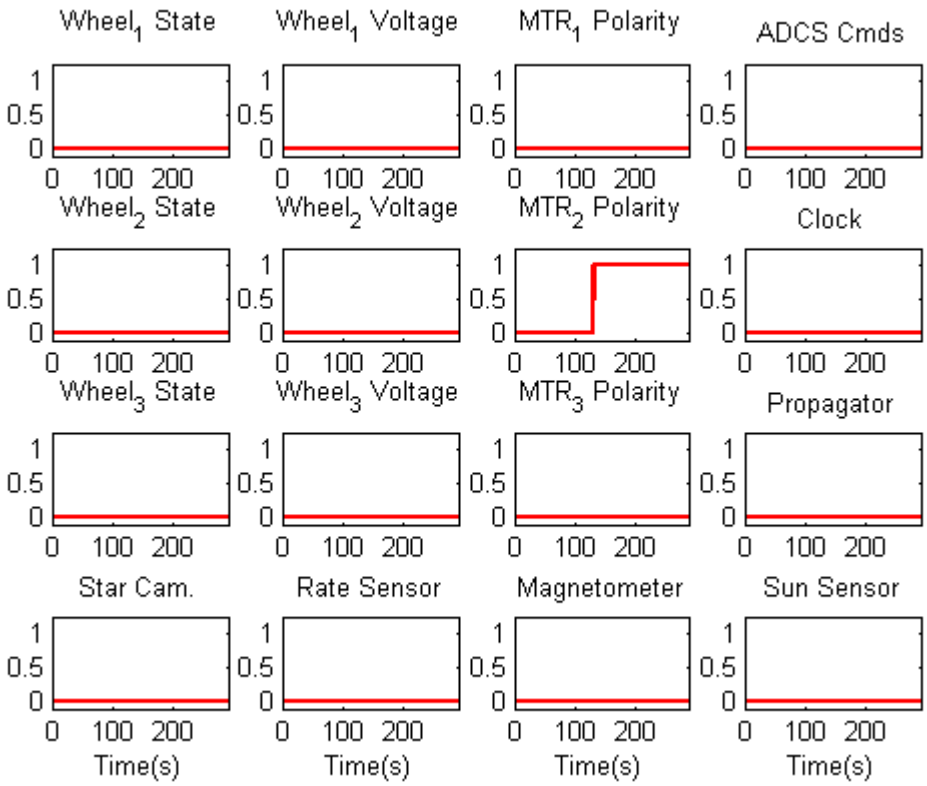

Fig. 5.10: The desired output associated with a fault in the magnetic torque rods

5.4.4 Training for Guidance Command Errors

A probable source of error when commanding a CubeSat to point towards a target is that the commands are somehow received improperly by the ADCS. This error could take one of several forms. For example, there is the possibility that the user working the ground console has programmed the pointing commands improperly. This is especially true when pointing towards a target of opportunity that has not been previously hard coded into the ADCS flight software. Another form that this error could take would be either a command parameter became corrupted during transmission, or there was an intermittent delay that caused the commands to be processed sporadically [19].

Two tests were defined to train for this error. The first tests for guidance commands to the ADCS that are either defined improperly or are otherwise corrupted prior to execution. In this instance two sets of maneuvers were defined: a $\pm 180^{\circ}$ pitch for the reference model

which represents the desired performance of the spacecraft, and a $\pm 180^{\circ}$ roll for the fault model which represents its actual performance. The assumption in this instance is that either the user programming the ADCS commands was in error or some unknown software glitch has garbled the original command. Figure 5.11 shows the residuals which are typical for this error. Note that since the final attitude and rate residuals did not return to a nominal value that this indicates that the spacecraft is not in the desired orientation.

When the guidance commands have been set improperly several biases present themselves in the residuals relative to the reference model. This is because the reference model is meant to represent the *desired* response of the ADCS compared to the *actual* response based on the spacecraft's telemetry. In both cases the spacecraft was commanded to turn through the same rotation angle with the same angular rate, but due to an unknown error, whether user or otherwise, the commands that were executed were not those that were intended.

The second test assumed a transient malfunction such that the guidance commands were received intermittently either due to processor malfunction or other unknown error source. These tests were defined such that the guidance commands were not received at $time = t$, but rather at $time = t + \Delta t$ where Δt is an unknown delay and is defined formally as

$$\Delta t = \mu_t + \nu_t \tag{5.5}$$

In equation (5.5) μ_t is the mean delay time and is allowed to vary between zero and $10,000\,\text{ms}$. The mean delay time was also allowed to vary by an amount, ν_t, which is a normally distributed parameter with standard deviation, σ_t defined as

$$\sigma_t = \frac{\mu_t}{10} \tag{5.6}$$

Residuals typical of the second scenario are given in figure 5.12.

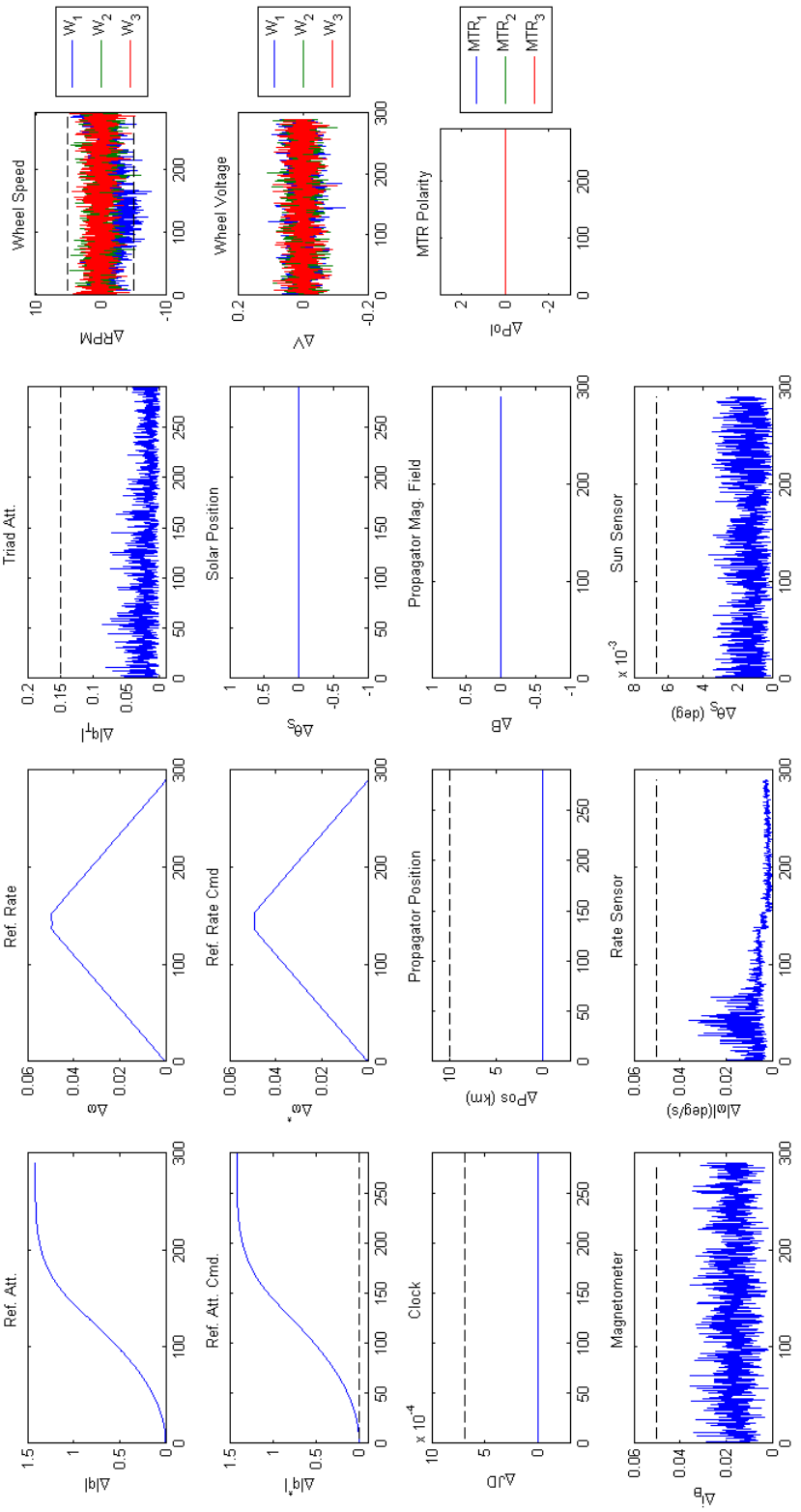

Fig. 5.11: The residuals associated with ADCS command errors

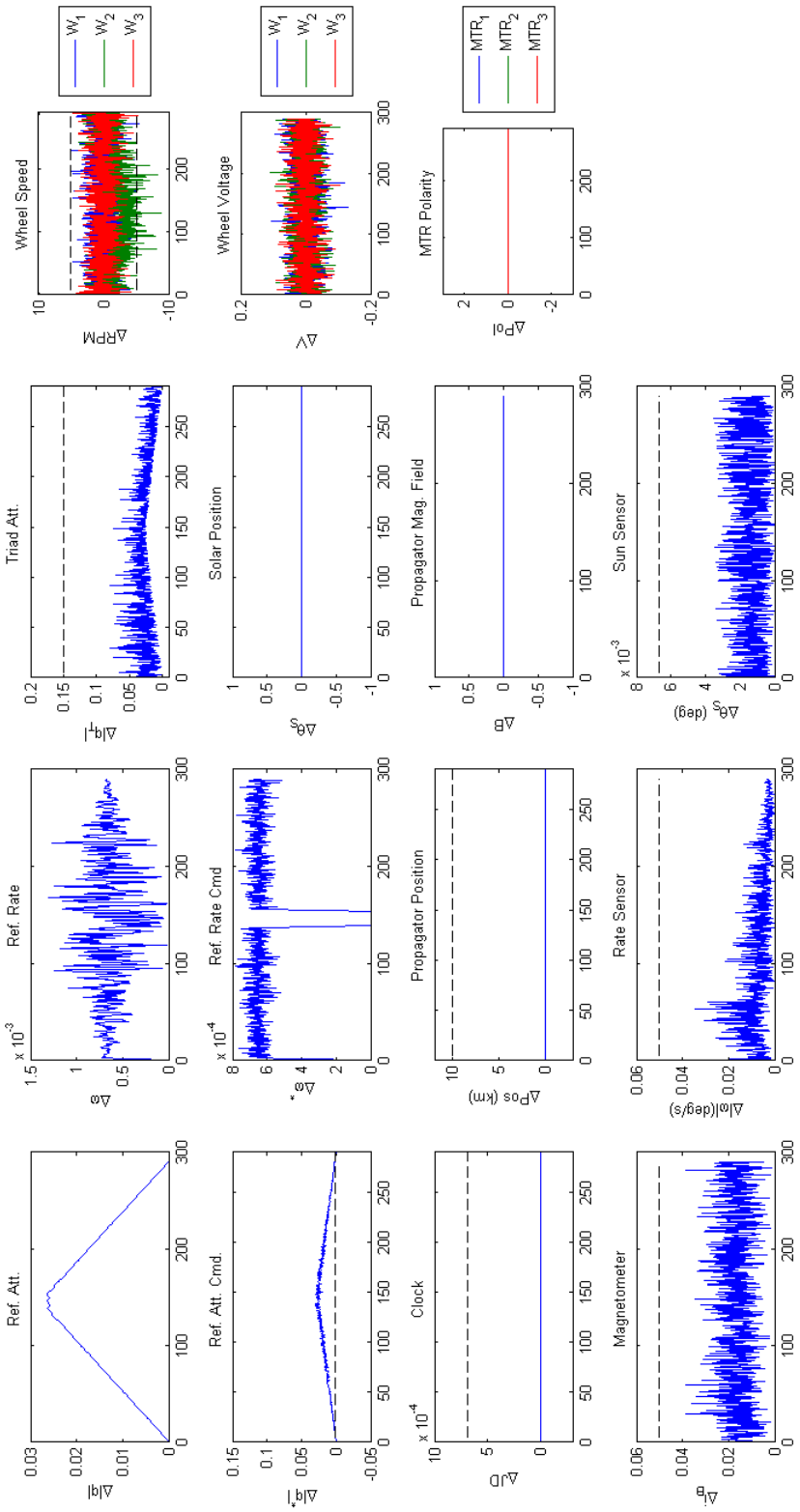

Fig. 5.12: Residual behavior indicative of improperly timed ADCS guidance commands

Figure 5.13 shows the response desired of the neural network for errors relative to the guidance commands received by the ADCS. Since the command residual is defined as the difference between the intended commands and those echoed by the spacecraft, the residual threshold was set empirically so as to be a small value, 0.002.

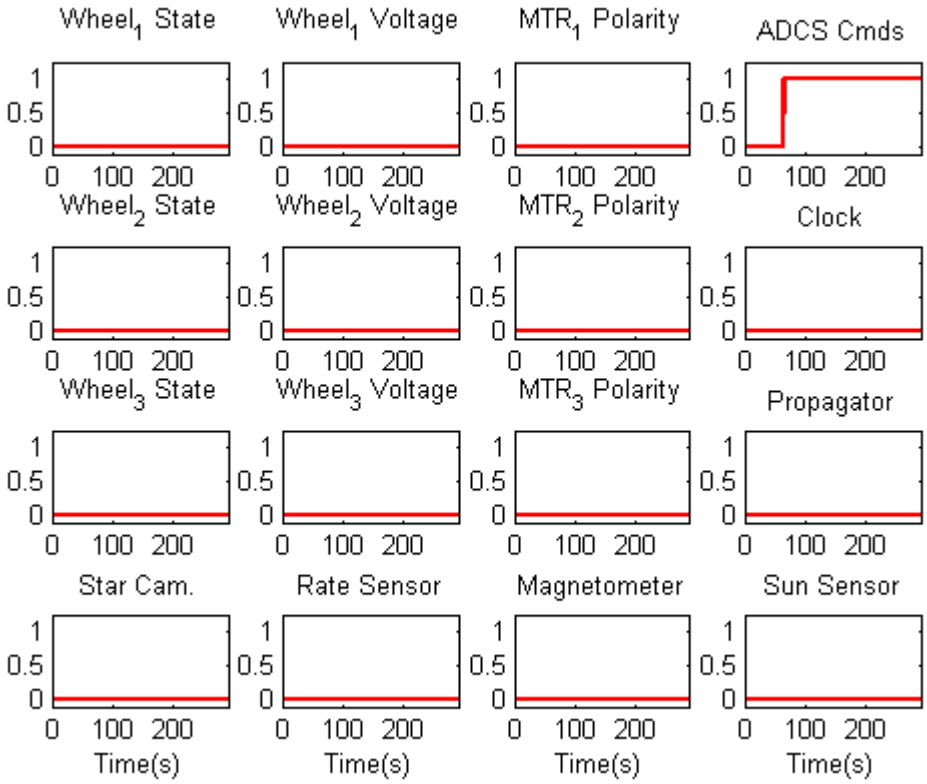

Fig. 5.13: The output desired of the neural network for errors relative to the ADCS guidance commands

5.4.5 Training for Clock and Propagator Errors

Training for errors related to the orbit propagator and the onboard clock encompassed whether or not the orbit propagator had drifted too far between updates and whether the clock was adequately synchronized with the ground. For both errors stationkeeping and slew maneuvers were defined for the ADCS. Figure 5.14 shows residuals typical of a propagator error.

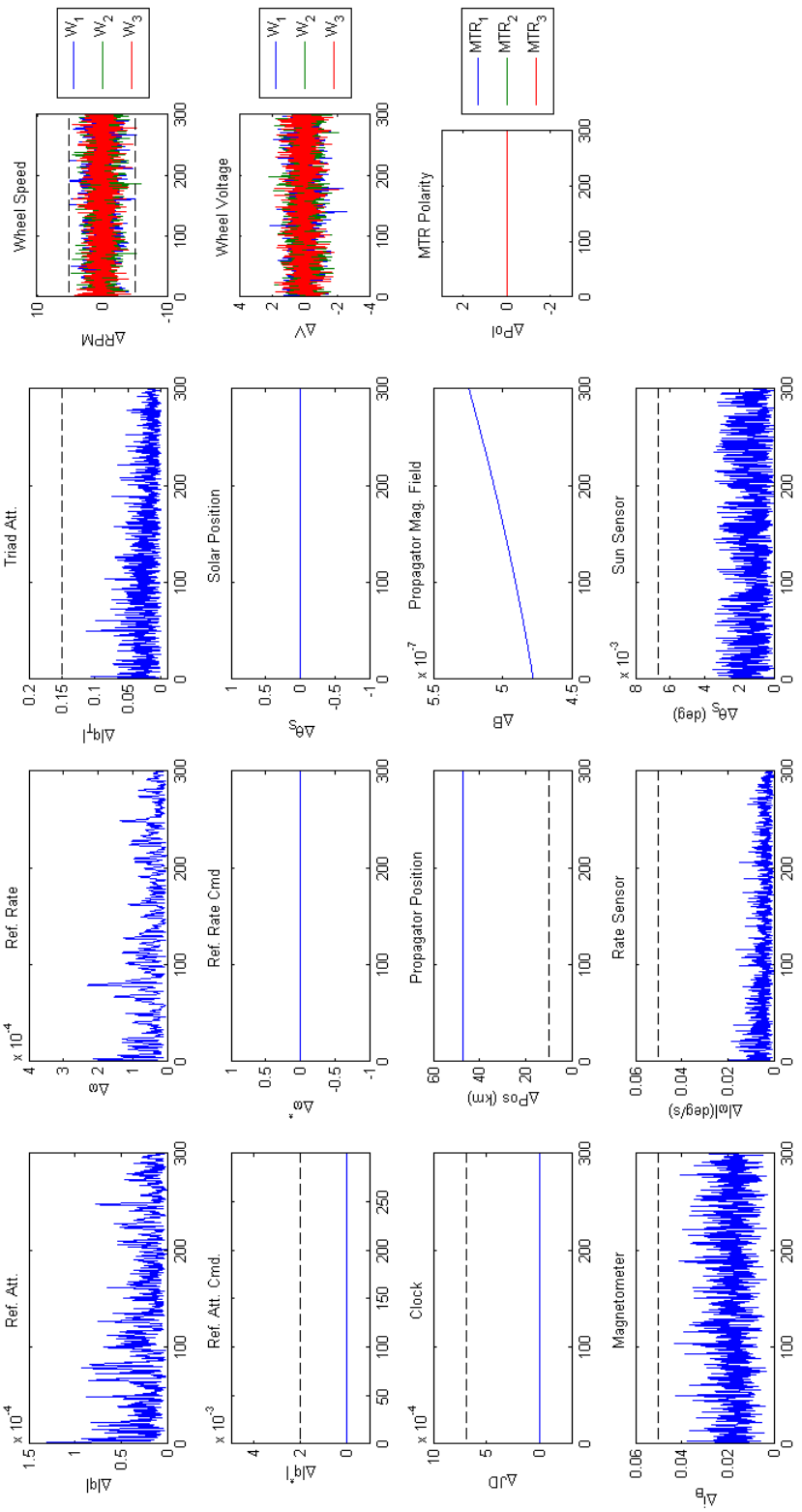

Fig. 5.14: The residuals for propagator related errors

When the propagator is in error several of the residuals that are dependent upon propagator inputs will experience a bias. The propagator position and magnetic field residuals, and to a lesser extent, the solar position residual, will all experience a bias when the propagator is in error. For the stationkeeping maneuver highlighted in figure 5.14 the propagator's orbit elements were allowed to drift from those that were assumed in the reference model such that were was an error of approximately 50 km between the position as estimated by the reference model and that reported by the onboard propagator. The propagator's magnetic field residual is also non-zero. These two non-zero residuals are excellent indications that the spacecraft may not necessarily be in the position it was assumed. It is worth noting also that the although the solar position residual is very small in this instance, its magnitude is dependent upon the size of the error in the propagator.

Training examples were also generated to allow the neural network to recognize when the time as reported by the spacecraft differed from that of the reference model. The timing of commands is critical to the spacecraft's pointing accuracy, especially when pointing towards the earth. To train for this error two example cases were considered. The first assumed that the onboard clock had reset to 1 January, 2000 00:00:00.000 UTC and had remained in this reset state prior to being updated either manually or by synchronizing with a GPS. The second case assumed a relatively small difference in time of one minute. Figure 5.15 shows the residuals typical for a clock error.

Upon inspection of the residuals when the onboard clock is not timed exactly with the reference, biases appear in any system that has residuals defined based on either the clock or the propagator. Note also the appearance of biases in the residuals of the reference attitude and angular rate and their commands. The appearance of residuals here is due to the fact that the ADCS was commanded to perform a maneuver at a specified time. As the onboard clock was not properly keeping time the ADCS failed to execute the maneuver as scheduled. Figures 5.16 and 5.17 show the desired network response to propagator and clock errors. Note that the propagator and clock residuals were thresholded at 10 km and 1 Julian minute (6.9e-4) respectively to generate the desired outputs.

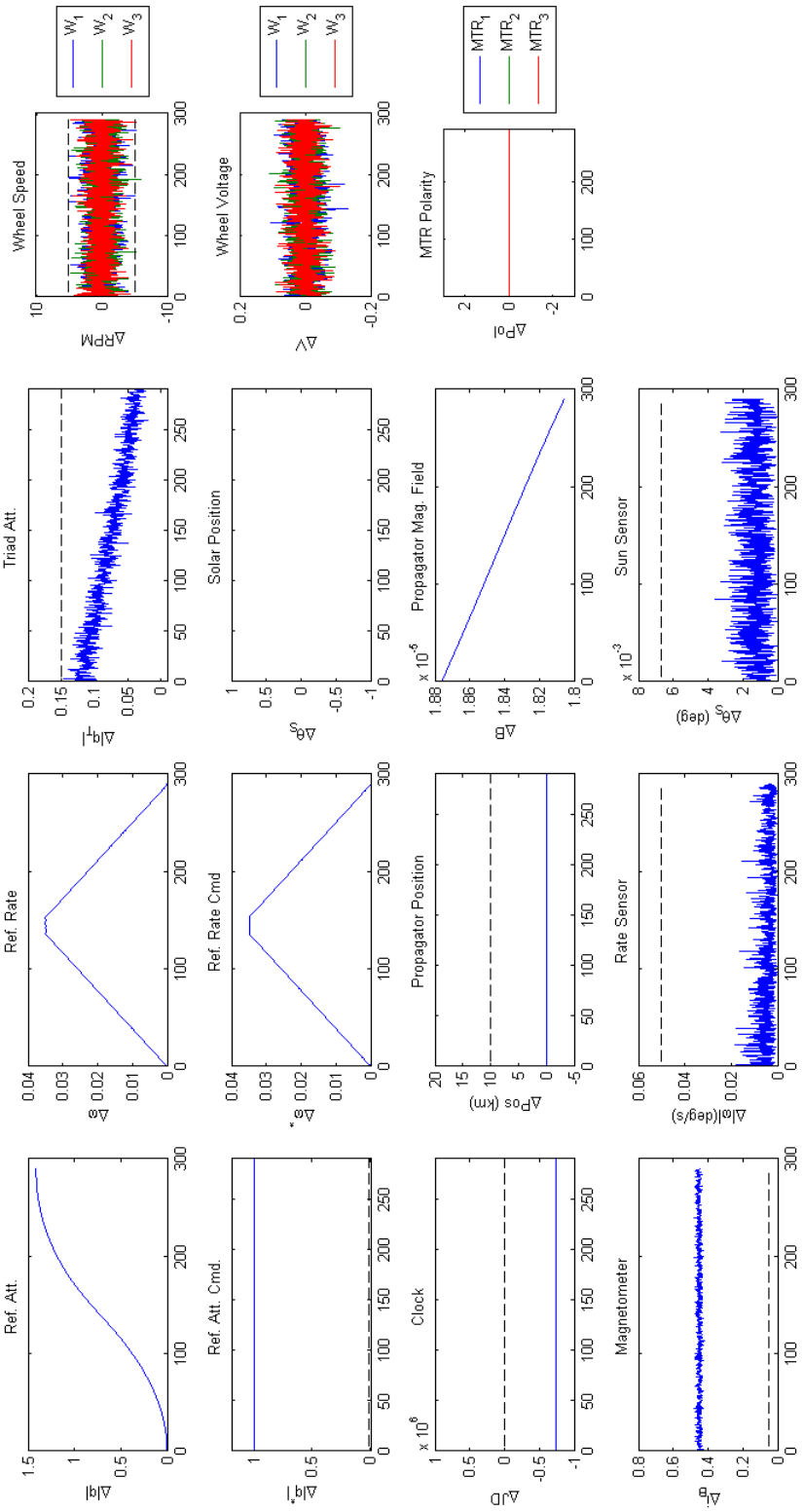

Fig. 5.15: The residuals associated with clock error

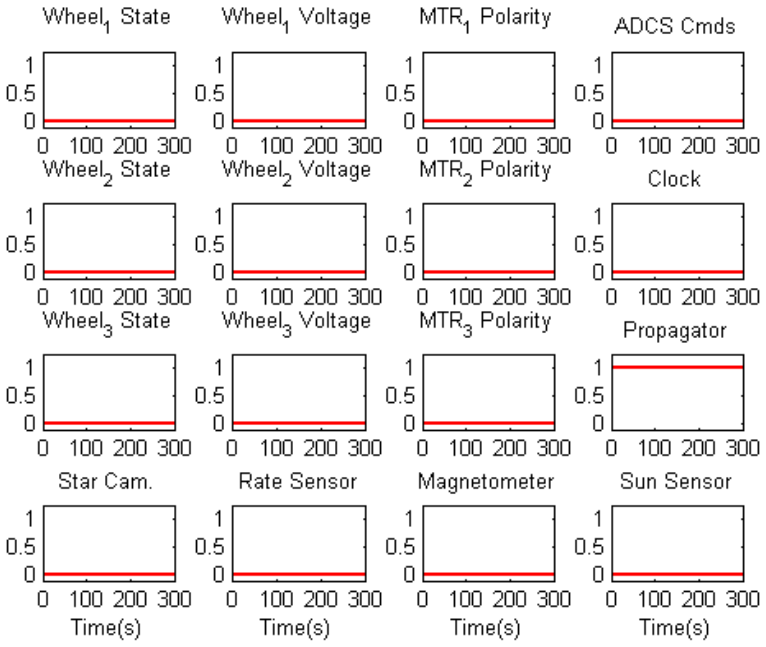

Fig. 5.16: The output desired of the neural network for a propagator error

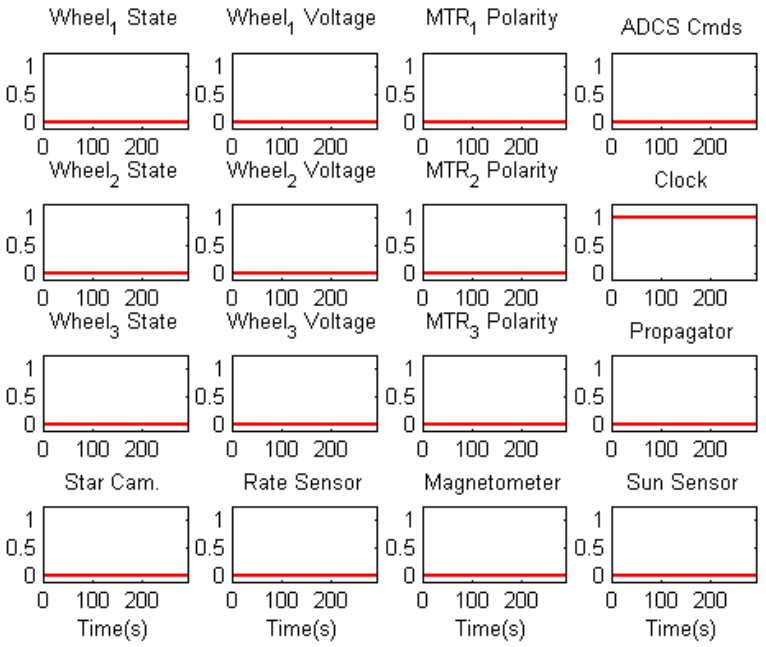

Fig. 5.17: The output desired of the neural network given a clock error

5.4.6 Training for Star Camera Errors

A star camera is often times the primary sensor used for attitude determination and as such its attitude solution is checked against the TRIAD solution of section 3.5.1. If the residual generated by these two solutions exceeds a nominal value, it could be that the star camera is in error. Probably the greatest sources of error in the star camera attitude solution is glint as well as field of view obstruction. Though many star camera manufacturers define "keep out" zones relative to the earth, moon, and sun to minimize the possibility that either of these could adversely effect the attitude solution, the possibility remains that the star camera is not immune from this error. The most likely way in which glint or field of view obstructions manifest themselves would be as an error in the attitude solution for one or more of the spacecraft's body axes. To model this error stationkeeping and 180° slew maneuvers were defined to bound the problem. For each of the maneuvers, Euler angle errors were introduced into the star camera attitude solution for each of the three body axes ranging from 0° to 45° prior to the solution passing through the Kalman filter. The residuals for a star camera attitude solution with a y-axis error due glint or field of view obstruction of 45° are shown in figure 5.19.

The main residual in consideration for this error is the TRIAD attitude solution as the purpose of the TRIAD solution is to act a check against the solution provided by the star camera and Kalman filter. For the case presented in figure 5.19 an error of 45° was introduced into the y-body axis of the star camera's attitude solution prior to filtering at $t = 145\,\mathrm{s}$. To simulate the transient nature of an error due to glint or field of view obstruction the fault was defined to persist for 10 seconds before returning to normal operation. After the error in the attitude solution has passed and the star camera returns to normal operation the ADCS again perceives that is it out of orientation due to its previous correction maneuver. In this instance the sudden shift in the attitude solution caused the ADCS to overreact trying to correct for what it perceived to be a gross error in its desired attitude. It then tries to reorient itself as quickly as it can by commanding its wheels to spin faster than their capability, which ultimately puts the spacecraft into a tumble from which it was not able

to recover. The reader will note also the large bias in the rate sensor residual. Although no fault was injected for the rate sensor in this scenario, the residuals for both the star camera and rate sensor are defined based on the post-filtered sensor solutions. Both the star camera attitude and rate sensor inputs are defined as states in the Kalman filter. Due to the coupling of the solutions in the filter, it was found that if a fault was defined in either the star camera or the rate sensor then the residuals for both the TRIAD attitude solution and the rate sensor would become biased. The false biasing of the residuals will be seen again during the discussion of the training for the rate sensor errors in section 5.4.8.

Based on empirical observation of the nominal TRIAD solution residuals, a threshold value of 0.15 was used to determine the fault state. The neural network's desired response to a fault in the star camera is given below in figure 5.18.

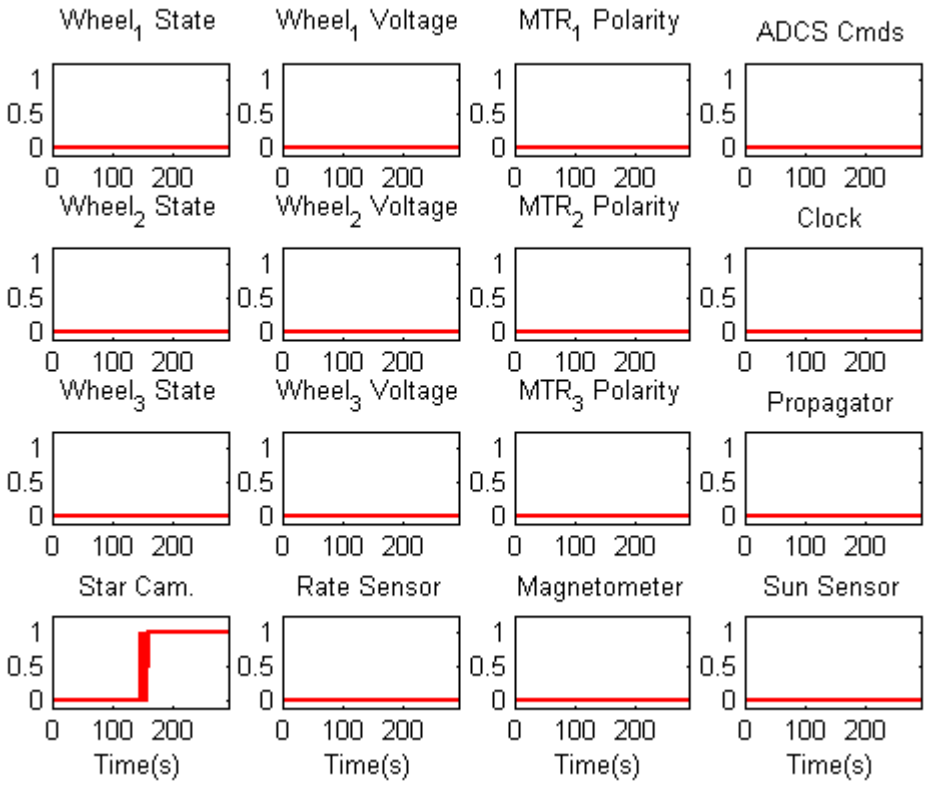

Fig. 5.18: The neural network's desired response to an error in the star camera solution

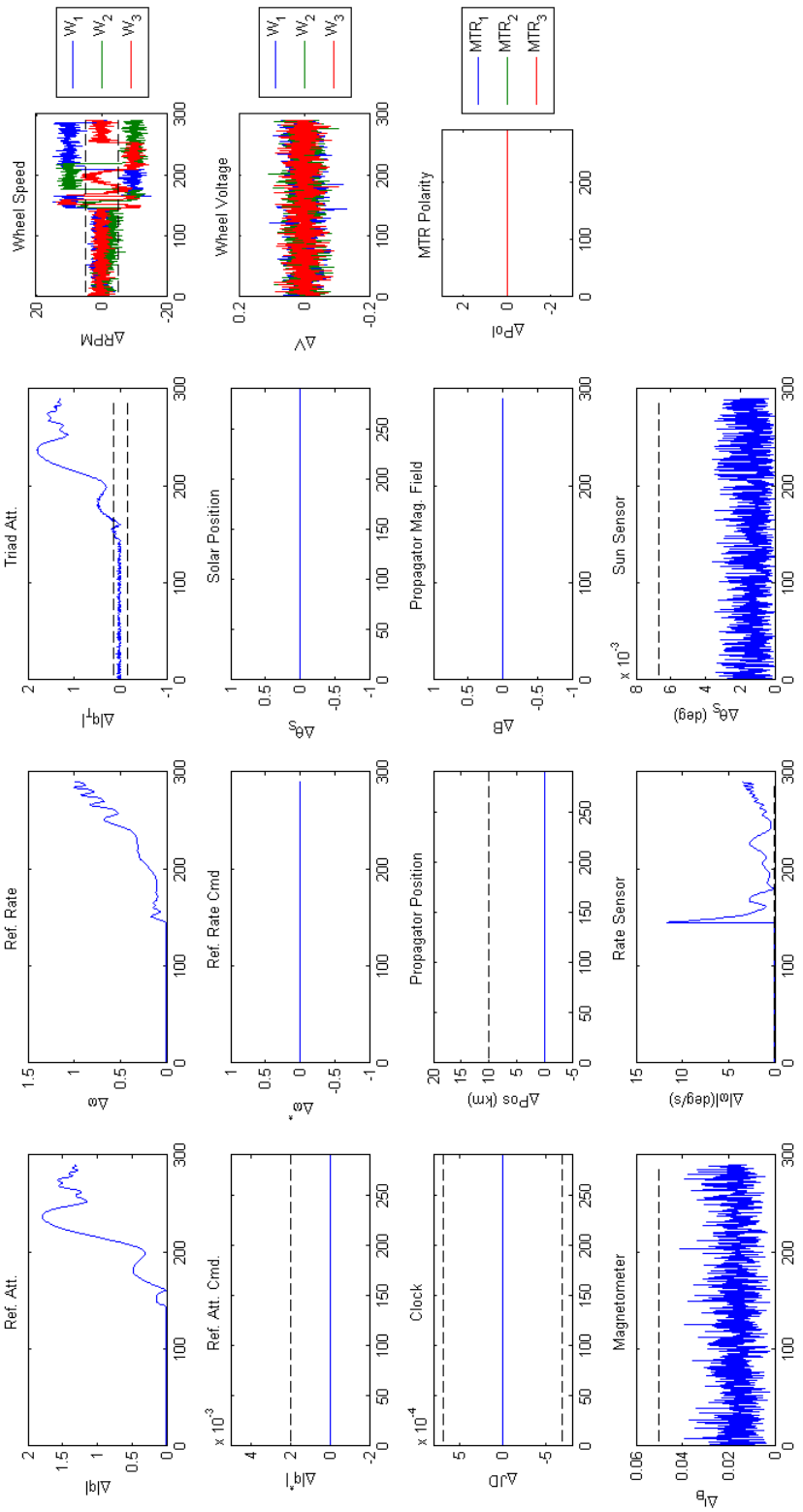

Fig. 5.19: The residuals associated with an attitude error due to the star camera

5.4.7 Training for Magnetometer and Sun Sensor Errors

Errors related to the magnetometer and sun sensor can come from many sources, but in many instances take the form of an unknown noise signal in the sensor measurements. The origin of this noise could be interference generated while the spacecraft is transmitting data or by a power brownout for example. The sensors themselves may also malfunction in such a fashion as to cause the output to "stick," for example if a magnetic torquer fires and saturates the magnetometer. Based on these assumptions, two types of errors were defined for both the magnetometer and sun sensor.

The first assumed either sensor was experiencing added noise of unknown magnitude. Recall from equation (4.33) that the measured magnetic field and solar unit vectors were modeled with noise components. To model a noise of unknown magnitude on either of the sensors, the nominal noise on the measurement was increased by scalar multiples of 2.5, 5, and 10 for spacecraft stationkeeping and 180° pitch maneuvers.

The second error defined for the magnetometer and sun sensor assumed that one of the sensor outputs for an individual axis had become "stuck" and remained constant. The magnetometer was assumed to saturate at $\pm 100,000\,\mathrm{nT}$. Training for this error consisted of commanding the spacecraft to perform the same stationkeeping and pitch maneuvers as before while holding each of the magnetometer's outputs constant at the saturation limit, as well as an intermediate value of $\pm 50,000\,\mathrm{nT}$ to bound the problem. Training data for the sun sensor was generated in a similar fashion. Since the output of the sun sensor is a unit vector, upper and lower limits of 1, with an intermediate limit of 0.5, were applied to each of the sun sensor's measured vector components.

Residuals typical for errors relating to either the magnetometer or sun sensor are presented in figure 5.21. Specifically, the residuals of figure 5.21 were generated by increasing the magnetometer's noise by a factor of 10. Noise of this magnitude easily exceeds the magnetometer's threshold value of 0.05. The residual of the TRIAD attitude solution is also experiencing a bias. As this residual is dependent upon the outputs of both the magnetometer and sun sensor, if either of these sensors is in error, the residual will be biased.

The neural network's desired response for errors in the magnetometer is given in figure 5.20. Errors relating to the sun sensor have the corresponding element of the target vector incremented as well.

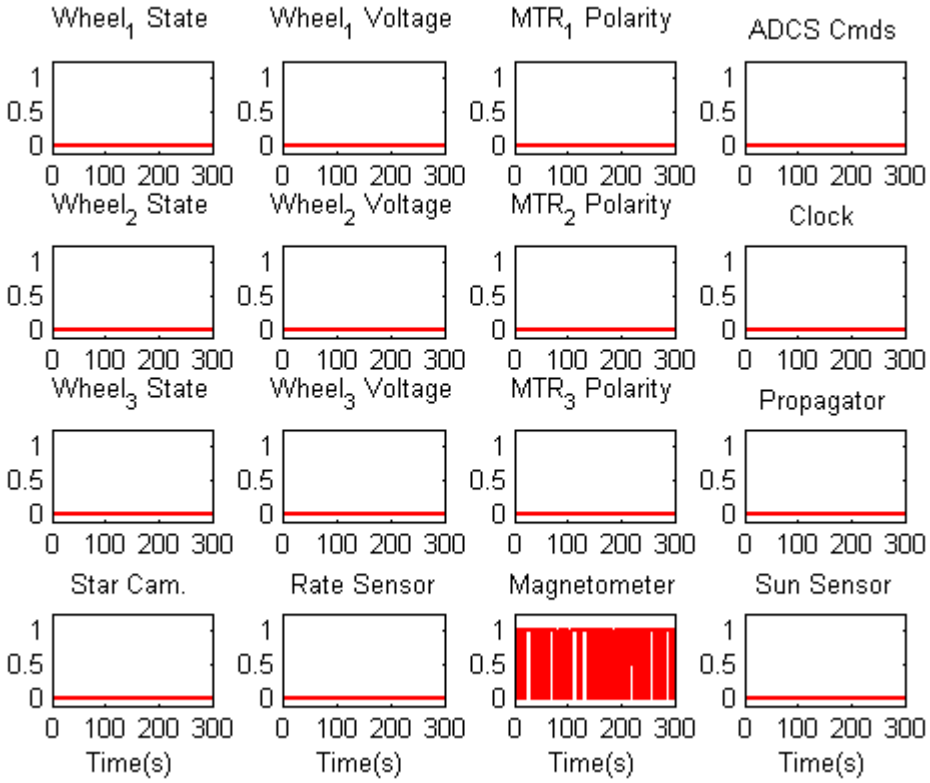

Fig. 5.20: Neural network output typical for a magnetometer error

5.4.8 Training for Rate Sensor Errors

Much like a magnetometer or sun sensor, a rate sensor, such as MEMS IMU, is susceptible to the types of error described in section 5.4.7; however, one of the most important characteristics of a rate sensor is drift in the bias [13]. This drift can be created in many ways. When the rate sensor in turned on initially it can experience a thermal bias as the electrical components heat up. Strain on the aluminum film of a MEMS IMU can create hysteresis which can be seen in the measurement [20]. Therefore, in addition to generating training data based on the errors described in section 5.4.7, the additional error of unknown drifting of the sensor bias was also added.

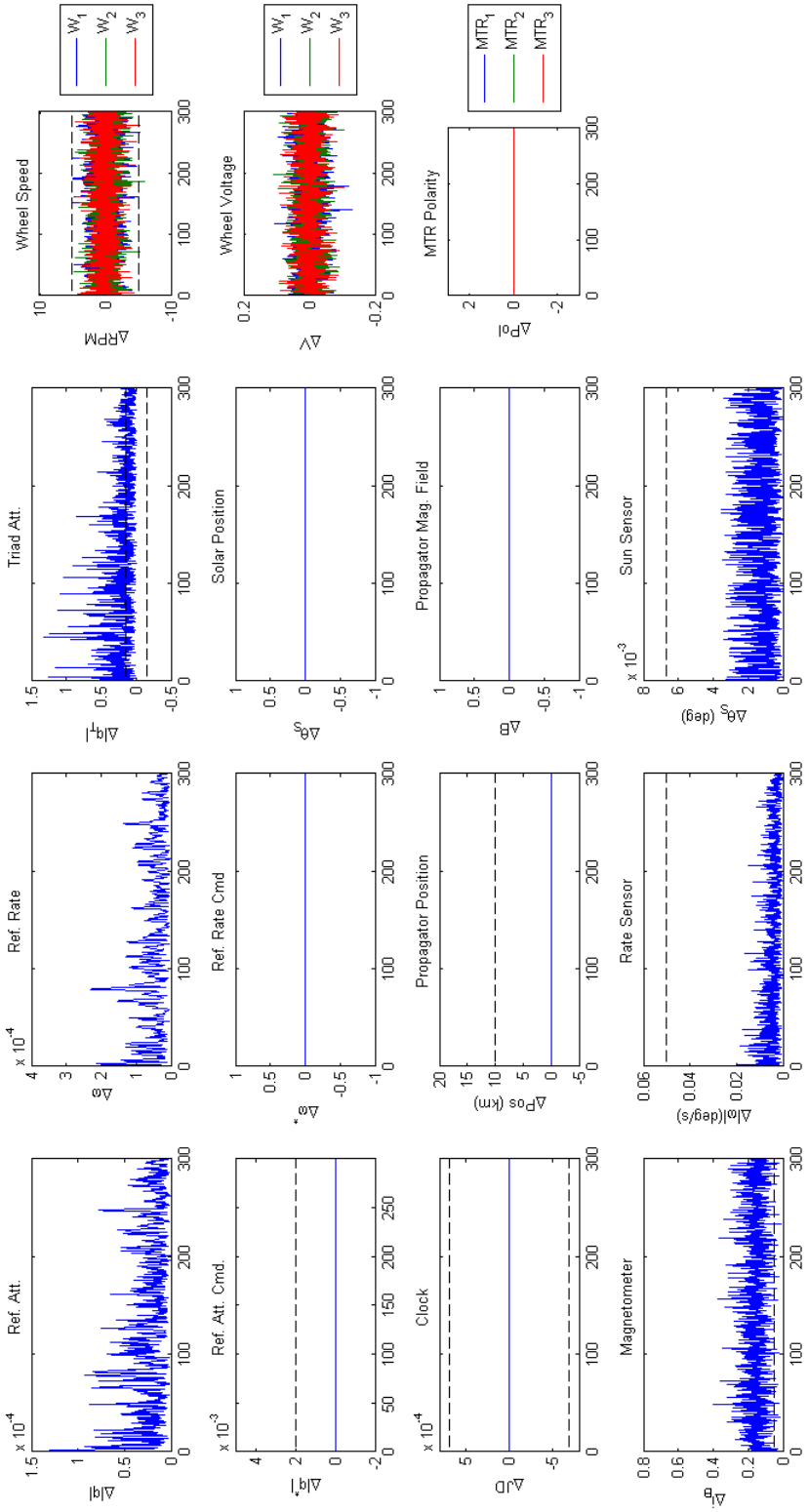

Fig. 5.21: Residuals typical for a magnetometer error

The rate sensor in the software model was assumed to have a nominal constant bias drift of $0.1\,^{\circ}/_{\text{hr}}$. As it is unlikely the bias would change very suddenly on its own, it was assumed that changes in the bias would be due to changes in the thermal operating environment, that is, the bias would increase or decrease as the temperature of the sensor increased or decreased. As such changes in the rate sensor bias were modeled as a simplified form of Gulmammadov [20] equation (2) as

$$\beta = Ae^{bt} \tag{5.7}$$

To model the fault, the bias, β, was held at the nominal value until $t = 60\,\text{s}$. At this time it began to increase exponentially until it had reached the desired value 30 seconds later. The constants, A and b, were determined by curve fitting an exponential function to the initial and final values of the bias. Final bias values of 0.2, 1, 2, 5, and $3,600\,^{\circ}/_{\text{hr}}$ were modeled each for stationkeeping and 180°pitch maneuvers. The case of $3,600\,^{\circ}/_{\text{hr}}$ was included to ensure that training examples for an unrecoverable fault condition would be generated.

The residuals generated for a rate sensor bias increase from $\beta = 0.1\,^{\circ}/_{\text{hr}}$ to $\beta = 1\,^{\circ}/_{\text{hr}}$ are shown in figure 5.23 and are typical of errors from this source.

When the rate sensor experiences a large enough change in its bias, the reference attitude and rate as well as the rate sensor residuals all be biased accordingly. After the sensor bias has reached its final value, the attitude residual returns to an almost nominal value, which is indicative of the spacecraft being able to complete its desired maneuver. The reference rate residual, however, remains biased in this instance as the sensor is not in a nominal operating state. Without correction this could eventually lead to the spacecraft being out of orientation.

Note also the drift in the TRIAD attitude residual. This same residual behavior was noted previously in section 5.4.6 with respect to training for errors with the star camera. The drift seen here is not due to any error in the star camera or magnetometer or sun sensor, but rather it is due to the coupled states of the Kalman filter.

The desired output of the neural network, shown in figure 5.22, was determined by thresholding the rate sensor residual at a value 0.05. This value was determined qualitatively by examining the behavior of the residual for nominal stationkeeping and slew maneuvers.

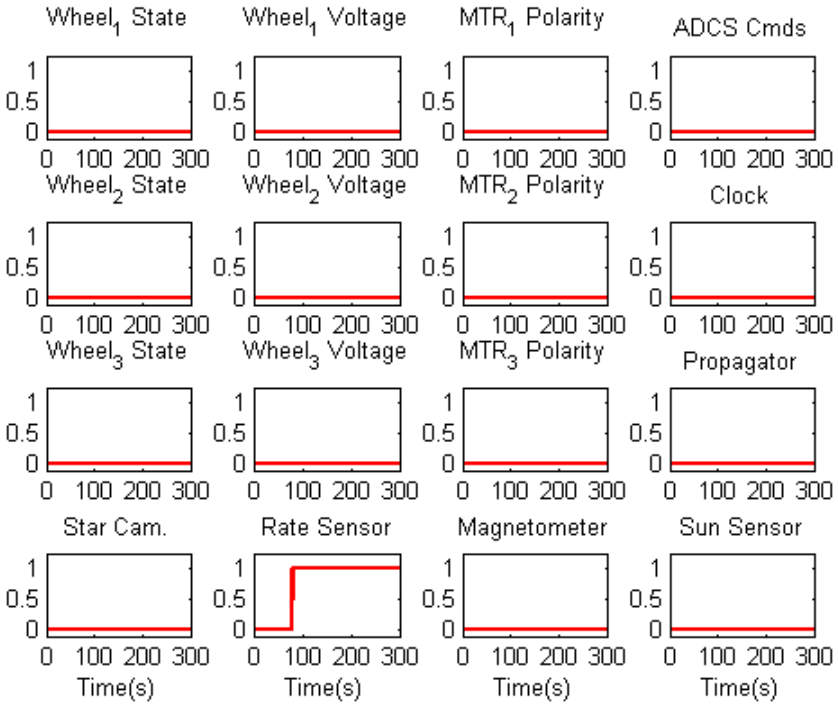

Fig. 5.22: The desired output of the neural network for a rate sensor error

5.5 Summary of Data Generation

A library of neural network training data was generated based on the examples of section 5.4. Although a limited number of examples were presented, there were a total of 112 sets of training data generated for a total of 91,810 individual examples for each of the 21 elements of the training vector (eq. (5.1)). Table 5.1 lists a summary of all of the data generated to train the neural network. A full listing of all the training data that was generated for this thesis is given in Appendix A.

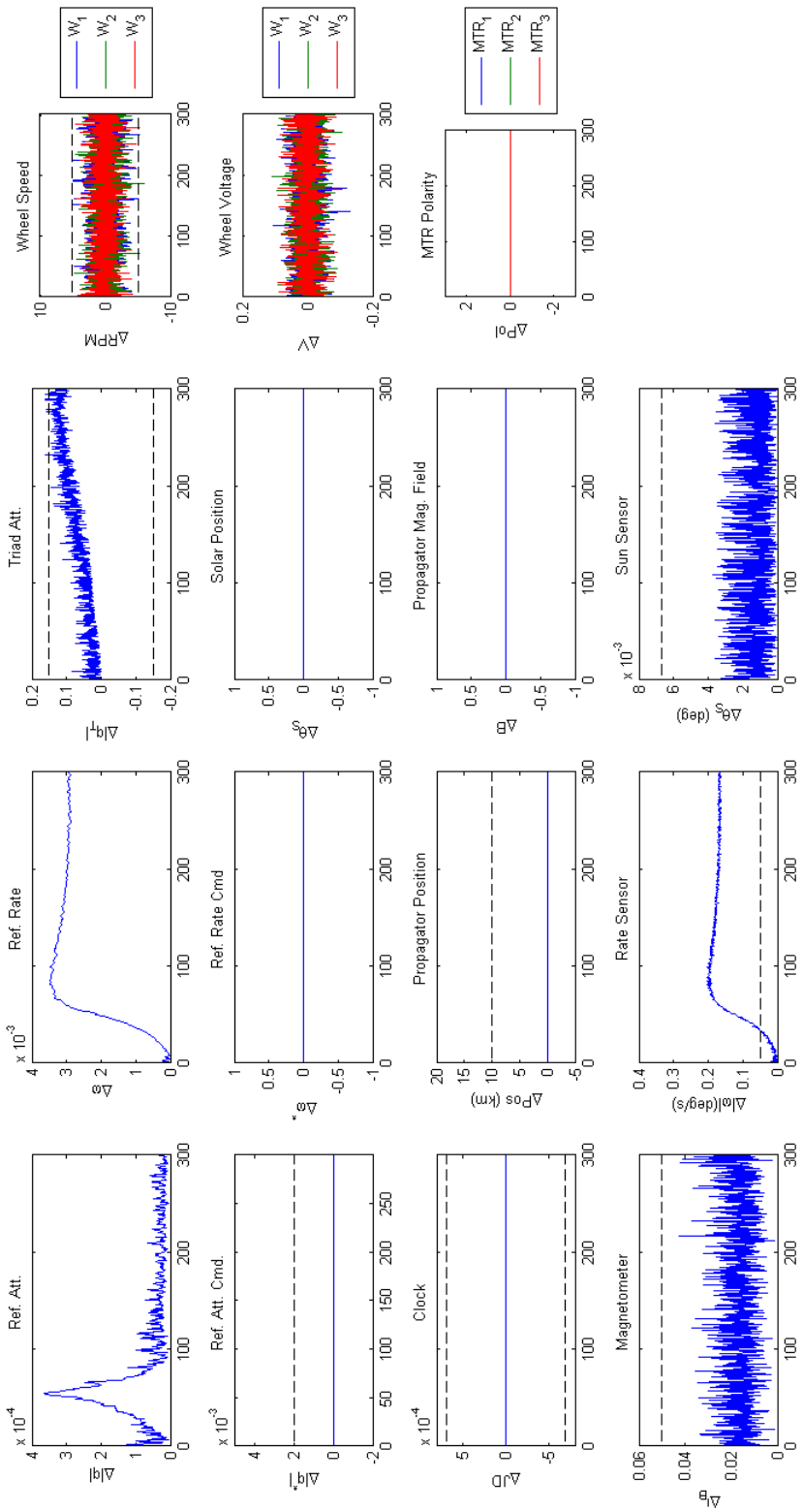

Fig. 5.23: Residuals biases due to rate sensor error

Table 5.1: Summary of the training data

Case	Description
1-4	Seize Reaction Wheel 2
5	Wheel 2 Nominal Operation
6-9	Wheel 2 Power Loss
10-11	Torquer 2 Reverse Polarity
12-13	Wrong ADCS cmds uploaded
14-15	Clock set incorrectly (JD is wrong)
16-19	Seize Reaction Wheel 1
20-23	Wheel 1 Power Loss
24-27	Seize Reaction Wheel 3
28-31	Wheel 3 Power Loss
32	Torquer 1 Reverse Polarity
33	Torquer 3 Reverse Polarity
34	Wheel 1 Nominal Operation
35	Wheel 3 Nominal Operation
36-37 104-108	Guidance command delay
38-45	Rate sensor bias increase
46-57	Glint on star camera (angle offsets)
58-68	Rate sensor unknown noise: x,y or z axes
60	Rate sensor saturates all axes
69	Nominal stationkeeping maneuver
70-77	Magnetometer unknown noise: x,y, or z axes
73-74	Magnetometer saturates all axes
78-85	Sun sensor unknown noise: x,y, or z axes
86-94	Sun sensor value is stuck: x,y, or z axes
95-103	Magnetometer value is stuck: x,y, or z axes
109-112	Propagator drift

The training data were generated for both stationkeeping and typical slew maneuvers of the HARP spacecraft. The magnitudes of the injected faults were varied in an attempt to bound the problem and generate as many different examples of abnormal residuals as possible. It was shown during training that faults pertaining to the star camera and rate sensor could give rise to false biasing of the TRIAD attitude residual.

5.6 Training Algorithms

When training a neural network two items must be taken into consideration. The

first is the method chosen to train the network. A common method used when training a neural network is backpropagation. Backpropagation is a method by which the gradient of an arbitrarily chosen error function, such as the mean squared error (MSE), is calculated repeatedly through use of the chain rule of calculus with respect to the synaptic weights of the neural network. Once the partial derivative of each weight is known a gradient descent is performed, which then defines how the synaptic weights should be updated [18].

The second item is the number of neurons chosen for the network. The number of neurons defined for a neural network is important and will vary depending on the size of the network being created. If too few neurons on chosen there is a risk of having a network that is too poorly generalized; too many and one risks the possibility of over-fitting the data.

Steepest descent methods such as backpropagation have a large disadvantage in that they can be very slow to converge. There are several backpropagation algorithms available that can speed up convergence. The resilient backpropagation (RPROP) algorithm developed by Riedmiller and Braun in [21] addresses one of the key impediments to quick convergence, that is, that convergence is dependent upon the size of the partial derivative of the weights. In other schemes, the size of the weight update is dictated by the magnitude of the partial derivative of the weight. The RPROP algorithm changes the size of the weight update irrespective of the size of the partial derivative. It does this by calculating the sign of each of the partial derivatives of the weights, and then incrementing the weight by a fixed factor depending on the calculated sign. The RPROP algorithm has several advantages: it is simple as it does not require calculation of a Hessian; it is computationally fast; and the memory requirements are modest as there are no large matrices that are needed to be stored. It also eliminates the problem of the solution stalling around a local minima where if the calculated gradient has a very small magnitude the weight update will also be very small even if it is very far away from being optimal [21].

Another popular method for training a neural network is the Levenberg-Marquardt (LM) algorithm. First developed by Marquardt in [22], it was later implemented for use in

training neural networks by Hagan and Menhaj [23]. Whereas the RPROP algorithm is an application of steepest descent, the LM algorithm is a modification of the Gauss-Newton method that approximates the Hessian of a function by application of its Jacobian. The main modification of the LM algorithm is the introduction of a tuning parameter into the Gauss-Newton solution. This parameter, when small, causes the algorithm to be Gauss-Newton; when large, the algorithm becomes steepest descent. The LM algorithm is, then, a union of the Gauss-Newton and steepest descent optimization methods. The advantages of this modification is that the solution is able to take large steps in the direction of the gradient when it is small, and small steps when the gradient is large so as not to oscillate about a minimum solution.

An extension of the Levenberg-Marquardt algorithm has been developed by Forsee and Hagan [24] which aims at improving generalisation of a neural network. By constraining the size of the network weights, the output of a neural network can be smoothed. This is known as regularization. One of the main problems with regularizing a neural network is choosing the appropriate regularization parameters as poor choice of parameters can either lead to over-fitting of the data or poor generalization of the network. Forsee and Hagan's extention of the LM algorithm attempts to automatically regularize the training of a neural network by application of Bayes' rule to choose the regularization parameters and is known as Bayesian Regularization (BR).

5.7 Network Training

How well a neural network can be trained is dependent on the size and complexity of the problem being analyzed. It cannot be definitively stated that one training algorithm will produce better results over another or that there is an optimal number of neurons to include in the network. As such, several neural networks were trained using each of the algorithms described in section 5.6 and comprising varying numbers of neurons to ensure the best generalization of the problem.

Eight neural networks were trained: four using the resisilient backpropagation (RPROP) algorithm; two using the Levenberg-Marquardt algorithm (LM); and two using Levenberg-

Marquardt with Bayesian regularization (BR). In order to produce the best generalized neural network, the number of neurons assigned to each network was varied. The networks trained using resilient backpropagation were defined with 20, 30, 40, and 60 neurons. Those trained with Levenberg-Marquardt or Bayesian regularization were defined with either 20 or 30. Recall that both Levenberg-Marquardt and Levenberg-Marquardt with Bayesian regularization require calculation and storage of the Jacobian of equation 5.1, which limited the number of neurons that could be prescribed to the neural networks trained with these algorithms. The neural networks were trained until each of their mean squared errors (MSE) was minimized. A summary of the trained networks is given in table 5.2.

Table 5.2: Summary of the neural network training

Algorithm	# Neurons	MSE
RPROP	20	0.0075
RPROP	30	0.0069
RPROP	40	0.0072
RPROP	60	0.0063
LM	20	0.0048
LM	30	0.0050
BR	20	0.0102
BR	30	0.0132

The best trained networks were those trained with the Levenberg-Marquardt algorithm and have an average performance of 0.005. The next best trained were those networks trained with resilient backpropagation and had a similar average mean squared error of 0.007. The networks with the worst performance were trained using Levenberg-Marquardt with Bayesian regularization and had an average mean squared error of 0.012. Retraining of these networks did not result in an improved error function.

An important aspect of training a neural network that must also be mentioned is the variability associated with the training. Prior to training, the weights and biases of a neural network are initialized with random values. This has the effect that training a neural network multiple times will produce different levels of performance. As an example, network RP20 was assigned twenty neurons and trained a total of eleven times using the resilient

backpropagation training algorithm. Each time it was trained it achieved a different level of performance. Figure 5.24 shows the distribution of the performance parameter for the training of this network. Over 11 trainings this network achieved a best performance of 0.0072, worst performance of 0.0085, and a median mean squared error of 0.0077. It is important to recognize that this variability exists and that the results presented in this thesis will inherently have some error bound attached to them due to the variability in the training.

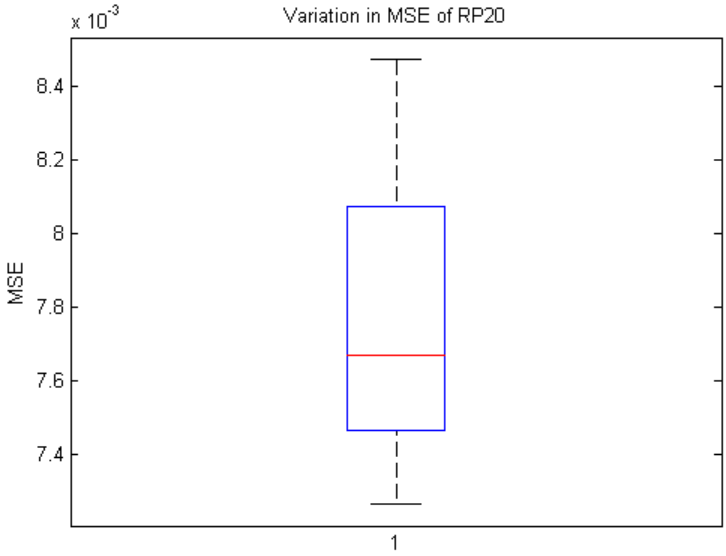

Fig. 5.24: The variation associated with training a neural network

Chapter 6

Results

6.1 Calculating Neural Network Performance

The output of each of the neural networks was calculated for each of the 112 training cases. A threshold of 0.9 was applied to each of the outputs (1 being the maximum value indicative of a fault detected). Each of the $16 \times n$ outputs was thresholded such that if the threshold was exceeded five continuous times a hard fault was recorded for the element under examination. The HARP spacecraft is capable of a 5 Hz attitude solution. Five continuous crossings was chosen to represent one continuous second of a fault state. This was done to aid limiting the number false positive detections. After thresholding, the number of positively and falsely identified faults was recorded. The performance of each network, given by equation (6.1), was determined by ratioing the number of positively identified faults by the number of total identified faults. Appendices B.3-B.5 provide the Matlab scripts used to calculate neural network performance.

$$\frac{Number\,of\,Positively\,Identified\,Faults}{Number\,of\,Positively\,Identified\,Faults + Number\,of\,Falsely\,Identified\,Faults} \times 100\% \tag{6.1}$$

6.2 Performance of the Neural Networks

Performance of the neural networks will be presented for each case and divided into seven categories: actuators, commmands, propagator, star camera, magnetometer, sun sensor, and rate sensor. For each category, the number of positively and falsely identified faults, as well as the overall accuracy of the neural networks will be given. The results

for each category are tabulated and color coded on a green (more accurate) to red (less accurate) scale.

6.2.1 Performance Relative to the Actuators

Reaction wheel speed, reaction wheel power, and the magnetic torquer polarity were grouped into the single category of actuators. The performance of the neural networks relative to the actuators is shown in table 6.1.

The Levenberg-Marquardt and the Bayesian regulated Levenberg-Marquardt networks have uniformly low detection accuracy for most all of the faults pertaining to the actuators. The networks trained with resilient backpropagation perform all around better, with the exception that faults pertaining to seizures in reaction wheel 3 are not detected well. Network performance in detecting actuator faults varies significantly not only network-to-network, but fault-to-fault as well. This is illustrated in figure 6.1, which shows an example network output set for a power loss in reaction wheel number two for two of the neural networks.

The set of outputs on the left of the figure were trained using the Levenberg-Marquardt algorithm, those on the right with resilient backpropagation. Both networks were defined with 30 neurons. In either instance the power failure in reaction wheel number 2 was successfully detected; however, the number of falsely identified faults of the network shown on the right of the figure makes it impossible to determine definitively where exactly the fault has occurred. This network-to-network variation persists in the outputs for all of the actuator faults, making it difficult to make a determination as to the efficacy of one network over another.

The accuracy of the neural networks in detecting actuator faults is shown in figure 6.2. Overall, those networks trained using resilient backpropagation had an average detection accuracy of 72%. Particular difficulty in detecting the fault in reaction wheel 3 served to significantly reduce this percentage. The four Levenberg-Marquardt trained networks had considerable difficulty in correctly detecting any of the faults with consistency and had an average accuracy of only 42%. It is worth noting also that the number of neurons used in each of the networks has positive effect on the detection accuracy of the actuator faults.

Table 6.1: Tabulated network results for the actuators

Case # Description	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
1 Seize Reaction Wheel 2	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
2 Seize Reaction Wheel 2	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
3 Seize Reaction Wheel 2	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
4 Seize Reaction Wheel 2	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
6 Wheel 2 Power Loss	1	9	10%	1	9	10%	1	6	14%	1	10	9%	0	7	0%	1	1	50%	1	3	25%	1	0	100%
7 Wheel 2 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
8 Wheel 2 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
9 Wheel 2 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
10 Torquer 2 Reverse Polarity	1	5	17%	1	0	100%	0	7	0%	1	5	17%	1	7	13%	1	8	11%	1	11	8%	1	5	17%
11 Torquer 2 Reverse Polarity	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	1	50%	0	0	0%	1	0	100%
16 Seize Reaction Wheel 1	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
17 Seize Reaction Wheel 1	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
18 Seize Reaction Wheel 1	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
19 Seize Reaction Wheel 1	0	0	0%	0	0	0%	0	0	0%	1	0	100%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
20 Wheel 1 Power Loss	1	4	20%	1	7	13%	1	9	10%	1	4	20%	0	2	0%	1	1	50%	0	0	0%	0	1	0%
21 Wheel 1 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%	0	0	0%
22 Wheel 1 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%	0	0	0%
23 Wheel 1 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%	0	0	0%
24 Seize Reaction Wheel 3	1	1	50%	1	1	50%	1	1	50%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
25 Seize Reaction Wheel 3	0	1	0%	1	1	50%	1	1	50%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
26 Seize Reaction Wheel 3	0	1	0%	1	1	50%	1	1	50%	1	1	50%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
27 Seize Reaction Wheel 3	0	1	0%	0	1	0%	0	1	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
28 Wheel 3 Power Loss	1	9	10%	1	10	9%	1	7	13%	1	9	10%	0	6	0%	0	3	0%	0	3	0%	0	3	0%
29 Wheel 3 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	0	0	0%	1	0	100%
30 Wheel 3 Power Loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	0	0	0%	1	0	100%
31 Wheel 3 Power Loss	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%	0	0	0%	1	0	100%
32 Torquer 1 Reverse Polarity	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	3	25%	0	0	0%	1	0	100%
33 Torquer 3 Reverse Polarity	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	0	0	0%	1	0	100%

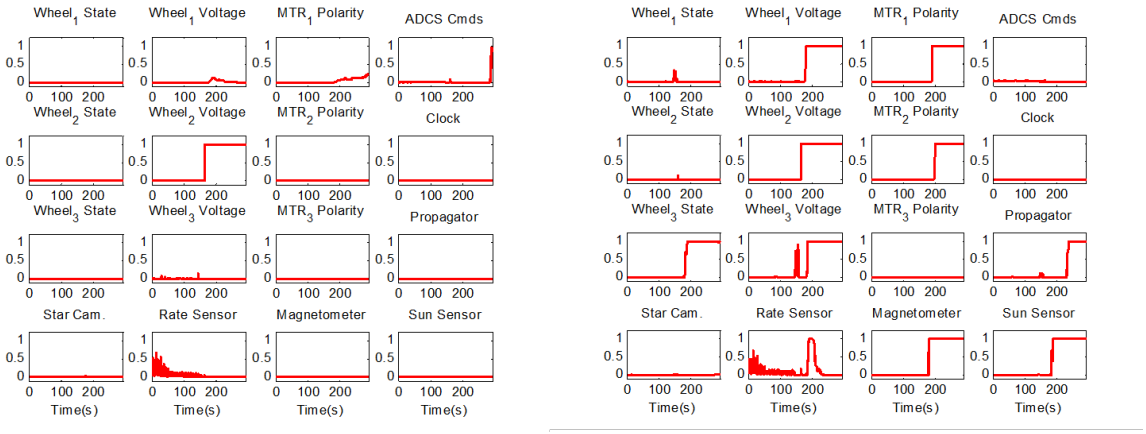

Fig. 6.1: An example of network-to-network variation

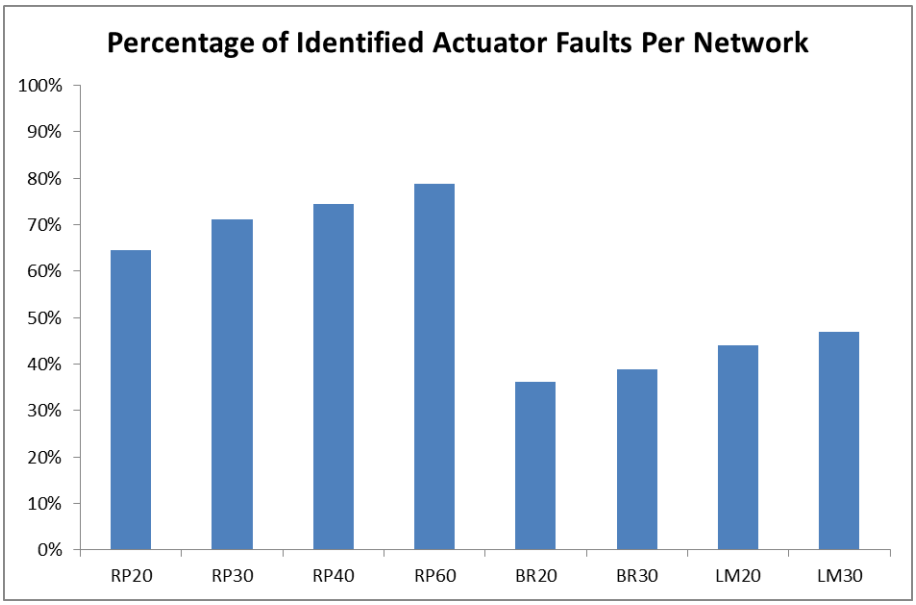

Fig. 6.2: The accuracy of the neural networks in detecting actuator faults

6.2.2 Performance Relative to Commanding

The performance of the neural networks relative to the ADCS's guidance commands is summarized in table 6.2.

The performance of the neural networks in detecting faults pertaining to the commanding of the ADCS displays the same network-to-network variation as for the actuators, except this time it is the Levenberg-Marquardt networks that are better able to detect the

faults. The networks trained with resilient backpropagation generalize poorly when trying to distinguish most of the errors of this type. Note that case 13 generalized poorly during training with respect to all of the network with the exception of networks RP20 and LM30.

Figure 6.3 summarizes the overall accuracy of the neural networks when attempting to identify faults in the commanding of the ADCS. The networks trained with RPROP had an average detection accuracy of only 39%, compared to an average of 82% for the LM networks. With the exception of network RP30, the rest of the resilient backpropagation networks displayed similar accuracy irrespective of the number of neurons assigned to the networks. This is true also of the four Levenberg-Marquardt networks. Whereas the actuators seemed to benefit from the addition of extra neurons in the neural networks, the same conclusion cannot be drawn definitively in this case.

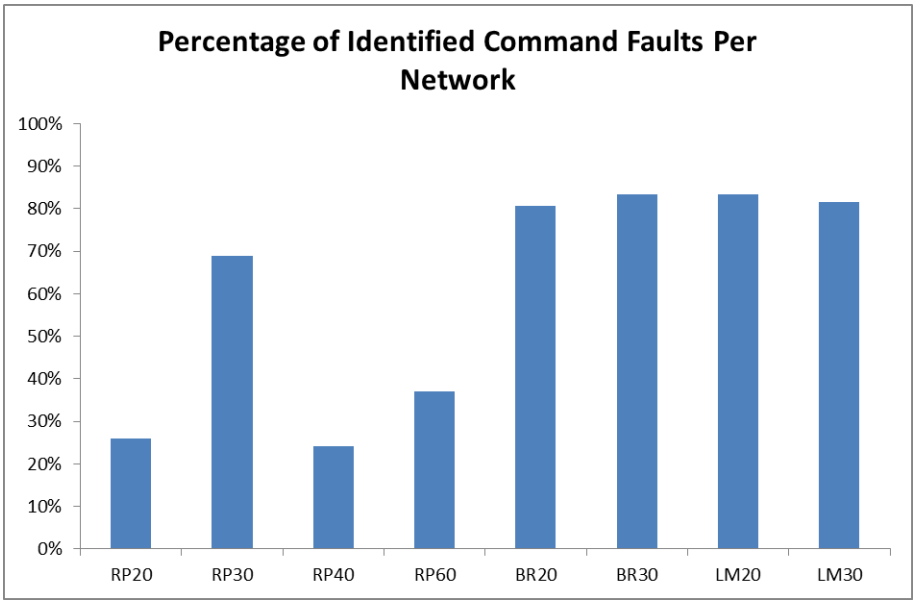

Fig. 6.3: The accuracy of the neural networks in detecting ADCS command faults

6.2.3 Performance Relative to the Propagator

Table 6.3 summarizes the performance of the neural networks in detecting propagator faults. Faults relating to the propagator relate to both the propagator specifically as well as the spacecraft's clock.

Table 6.2: Tabulated results for ADCS command errors

Case #	Description	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
		#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
12	Wrong ADCS cmds uploaded	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
13	Wrong ADCS cmds uploaded	1	0	100%	1	3	25%	1	1	50%	1	1	50%	1	2	33%	1	1	50%	1	1	50%	1	0	100%
36	ADCS Command delay	0	0	100%	0	0	100%	0	0	100%	0	0	100%	0	0	100%	0	0	100%	0	0	100%	0	0	100%
37	ADCS Command delay	0	0	0%	1	0	100%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
104	ADCS Command delay	0	0	0%	1	0	100%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
105	ADCS Command delay	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
106	ADCS Command delay	0	0	0%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
107	ADCS Command delay	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
108	ADCS Command delay	0	0	0%	1	0	100%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	0	100%

All of the neural networks were able to generalize well with respect to almost all of the faults in this category. Case 14 from the table was defined as a clock error where the clock was reset to 1 Jan, 2000 00:00:00.000 UTC. Networks BR20 and LM30 both detected the desired fault as well as a single false positive detection of either a propagator fault (network LM30) or reaction wheel fault (BR20) as shown in figure 6.4. In this instance, neither of these networks was sufficiently able to distinguish the clock error based on the residuals of this case, which led to the false positive detection.

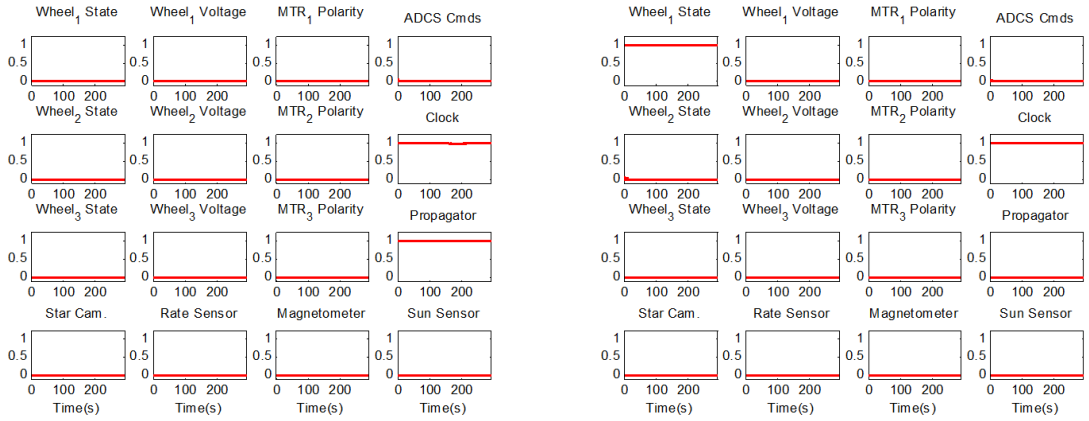

Fig. 6.4: Neural net output for case 13 for networks LM30 (left) and BR20 (right)

The resilient backpropagation networks managed to detect these faults with 100% accuracy. The Levenberg-Marquardt algorithms have a similarly high average detection accuracy of 94%. The overall detection accuracy of each of the neural networks for detecting these faults is given in figure 6.5.

Table 6.3: Tabulated results for detecting propagator errors

	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
Case # Description	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**	**#Pos**	**#False**	**Acc Pct**
14 Clock set incorrectly	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	1	50%	1	0	100%	1	0	100%	1	1	50%
15 Clock set incorrectly	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%
109 Propagator drift	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
110 Propagator drift	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
111 Propagator drift	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
112 Propagator reset	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%

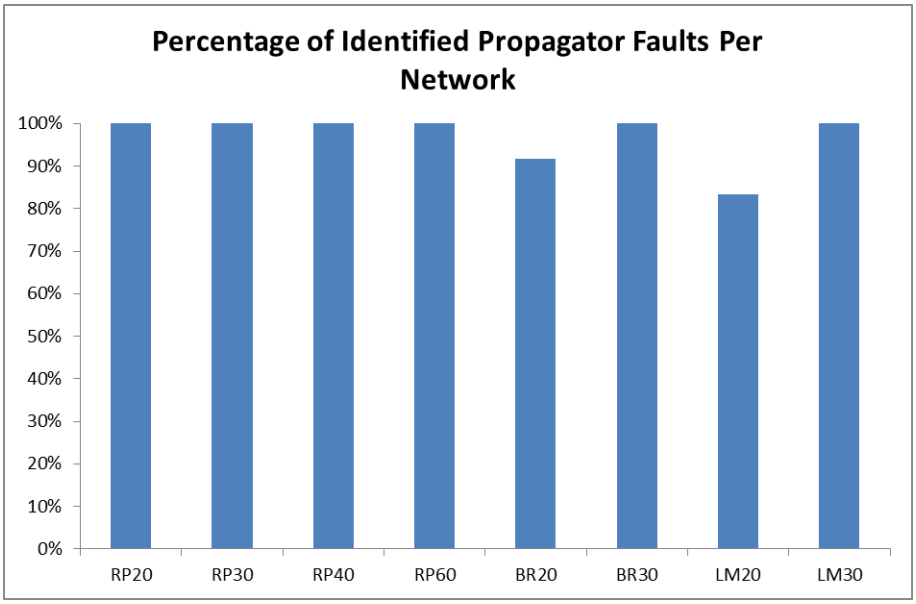

Fig. 6.5: The accuracy of the neural networks for detecting propagator and clock faults

6.2.4 Performance Relative to the Star Camera

The performance of the neural networks in detecting errors in the star camera is given in table 6.4. The poor performance of the neural networks in detecting errors in the star camera is not wholly surprising. Recall from section 5.4.6 that a false biasing of the rate sensor residual occurred when errors were introduced into the star camera solution. Because of this, the residuals for either the star camera or rate sensor faults are not sufficiently distinguished from one another. As a result, the neural networks have a degree of difficulty in determining whether the fault was due to either the star camera or rate sensor, as is the case with the Levenberg-Marquardt trained networks (see the left side of figure 6.6), or in detecting any fault at all, as with the resilient backpropagation networks (see the right side of figure 6.6). Although either set of networks begins to detect an error in both the star camera and rate sensor, the networks trained with RPROP fail to meet the thresholding criteria described in section 6.1, and therefore do not register any fault at all.

On average the networks trained using resilient backpropagation successfully detected the faults only 17% of the time. The Levenberg-Marquardt networks did not fair much better, having an average detection accuracy of 27%. The detecting accuracy of each individual

network is shown in figure 6.7. With respect to the RPROP networks, with the exception of network RP30, which did not detect any faults, either positive or negative, increasing the number of neurons in the network aided generalization of the network and enabled at least partial success in positively detecting the fault while minimizing the number of false detections.

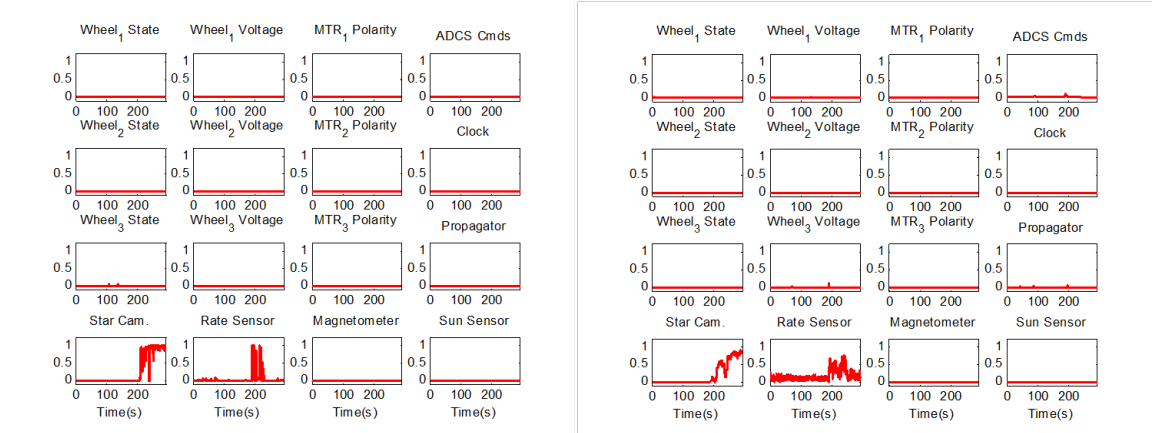

Fig. 6.6: A typical output for star camera faults for the Levenberg-Marquardt networks (left) and the resilient backpropagation networks (right)

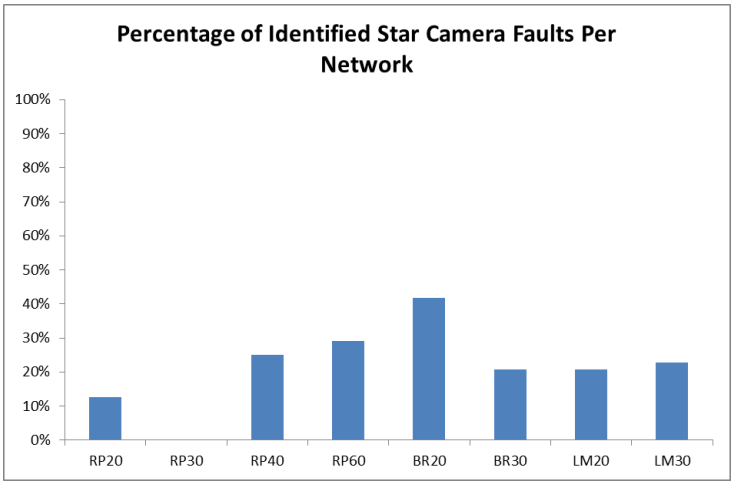

Fig. 6.7: The accuracy of the neural networks in detecting star camera errors

Table 6.4: The detection accuracy of the neural networks for detecting star camera faults

	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
Case # Description	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
46 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	0	0%
47 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	1	0%
48 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	0	0%	0	0	0%
49 Glint on star camera	1	1	50%	0	0	0%	1	1	50%	1	1	50%	1	0	100%	1	1	50%	1	1	50%	1	1	50%
50 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	1	0%
51 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	1	0%
52 Glint on star camera	0	1	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	1	50%	1	1	50%	1	1	50%
53 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	1	0%
54 Glint on star camera	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	1	0%	0	1	0%	0	1	0%
55 Glint on star camera	0	1	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	1	50%	1	1	50%	1	1	50%
56 Glint on star camera	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	1	1	50%	1	1	50%	1	1	50%
57 Glint on star camera	0	1	0%	0	0	0%	1	1	50%	0	1	0%	1	0	100%	1	1	50%	1	1	50%	1	1	50%

6.2.5 Performance Relative to the Magnetometer

Table 6.5 summarizes the performance of the neural networks in detecting faults with the magnetometer.

With the exception of cases 72 and 77 the neural networks generalized errors pertaining to the magnetometer very well, with near 100% detection accuracy. Cases 72 and 77 represent faults of low magnitude for stationkeeping and slew maneuvers. Examination of the outputs of the neural networks (see figure 6.8) shows that the networks are in fact detecting an error in the magnetometer; however, due to thresholding a hard fault is not detected.

Figure 6.9 shows the overall accuracy of the individual networks in detecting magnetometer faults. On average the resilient backpropagation and the Levenberg-Marquardt networks were able to accurately detect the fault 90% and 95% of the time respectively.

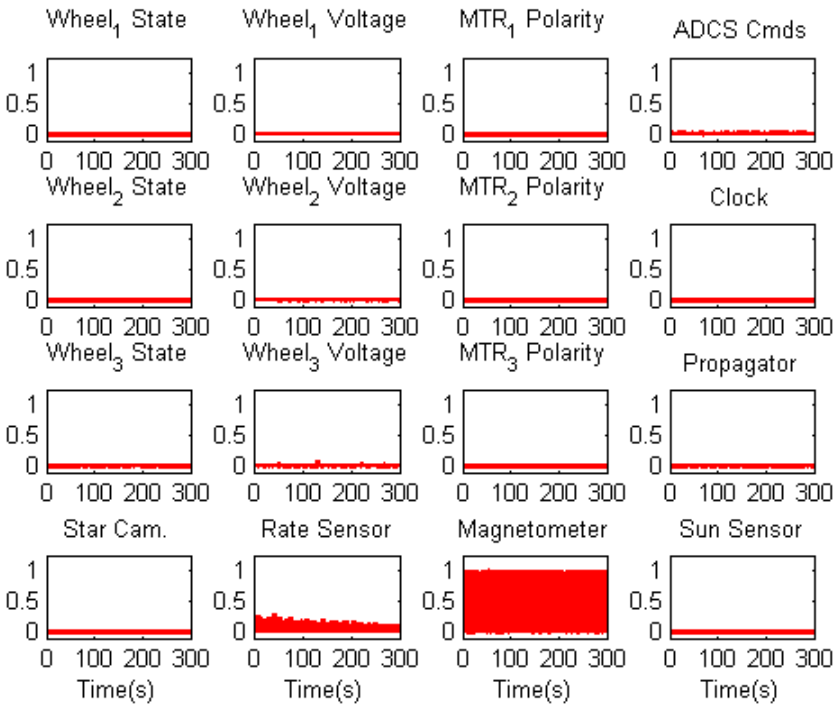

Fig. 6.8: A magnetometer fault on the verge of being detected

Table 6.5: The performance of the neural networks for detecting magnetometer faults

		RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
Case #	Description	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
70	Magnetometer unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
71	Magnetometer unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
72	Magnetometer unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
73	Magnetometer goes dead	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
74	Magnetometer goes dead	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
75	Magnetometer unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
76	Magnetometer unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
77	Magnetometer unknown noise	0	0	0%	1	0	100%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	0	0	0%
95	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
96	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
97	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
98	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
99	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
100	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
101	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
102	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%
103	Magnetometer value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%

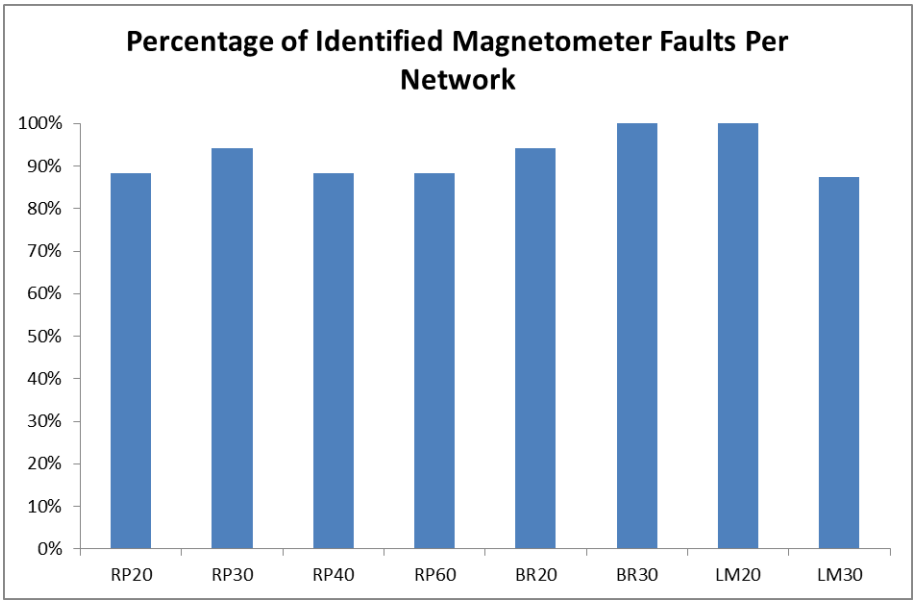

Fig. 6.9: The accuracy of the neural networks in detecting magnetometer faults

6.2.6 Performance Relative to the Sun Sensor

The performance of the neural networks in detecting errors in the sun sensor is given in table 6.6. With the exception of network BR30, which did not generalize well, all of the networks performed very well with respect to detecting faults of the sun sensor. Cases 78 and 79 represent faults of low magnitude. Examination of the neural network outputs (see figure 6.10) shows that the networks are detecting an error in the sun sensor, but in a similar fashion as for the low magnitude magnetometer faults, thresholding prevents determination of a hard fault.

The networks trained using resilient backpropagation had an average detection accuracy of 88%. Those trained with Levenberg-Marquardt, excluding network BR30, had an average accuracy of 90%. The individual network accuracies for detecting sun sensor errors are given in figure 6.11.

Table 6.6: The performance of the neural networks for detecting sun sensor faults

	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
Case # Description	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
78 Sun sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
79 Sun sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
80 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
81 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
82 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
83 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
84 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
85 Sun sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
86 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
87 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
88 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
89 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
90 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
91 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
92 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
93 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%
94 Sun sensor value is stuck	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%

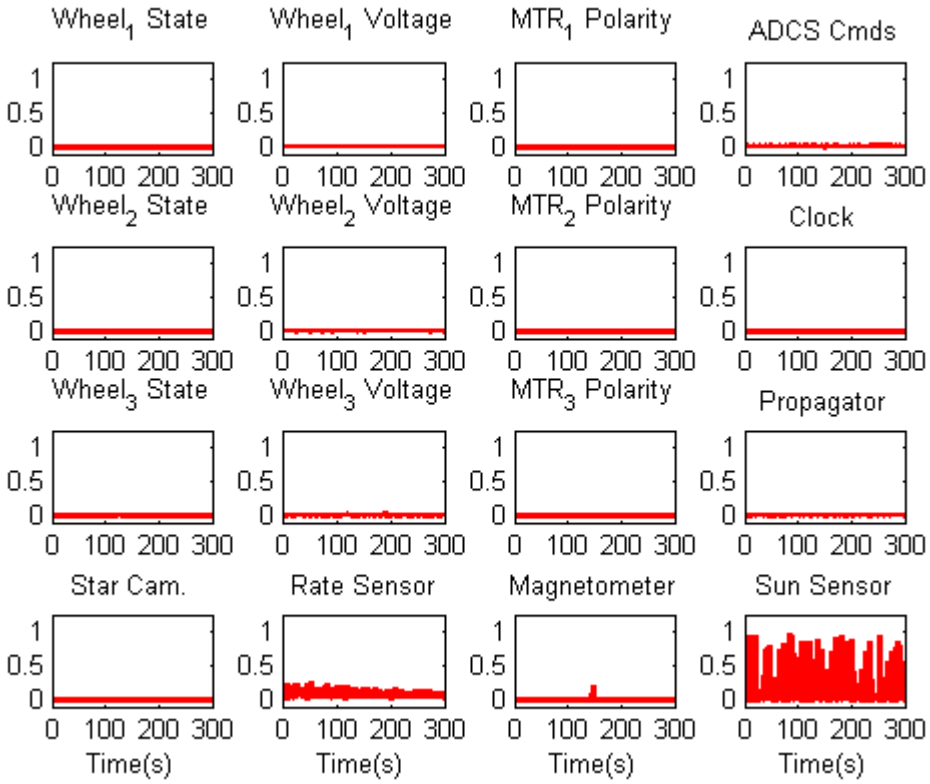

Fig. 6.10: Typical output for a low magnitude sun sensor fault

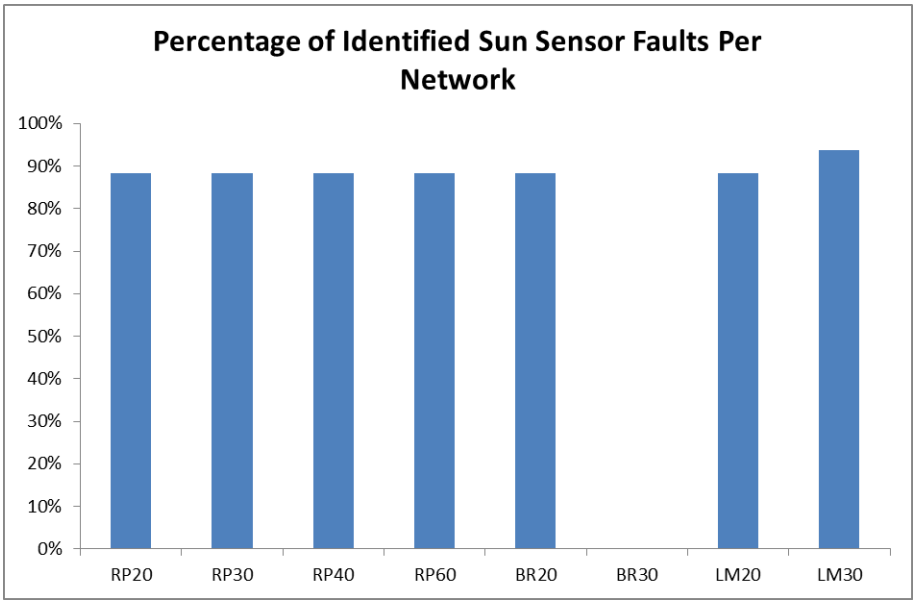

Fig. 6.11: The accuracy of the neural networks in detecting sun sensor faults

6.2.7 Performance Relative to the Rate Sensor

Table 6.7 summarizes the performance of the neural networks in detecting faults relating to the rate sensor. The performance of the neural networks in detecting faults in the rate sensors is generally low. Although the Levenberg-Marquardt networks perform better, there is a large degree of network-to-network variation in detecting these faults. Figure 6.12 presents neural network output of case 66 (medium magnitude error) and is typical for this error for both the resilient backpropagation networks (shown on the left of the figure) and the Levenberg-Marquardt networks (shown on the right of the figure). The networks trained using the Levenberg-Marquardt schemes produce a strong reaction in the presence of these faults. Although in these instances they fail to meet the thresholding criteria, it is still possible to make a positive fault determination based on examination of the behavior of the output.

It must also be noted that some of the neural networks are able to better generalize more than others. Figure 6.13 shows the outputs of case 67 (high magnitude error) for networks RP30 (left of the figure) and LM30 (right of the figure). In this instance the output of the networks is similar to that as for the star camera, where neither set of networks is able to distinguish sufficiently well an error in the rate sensor from that of the star camera. Although thresholding prevented detection of false positives, examination of the residuals would make it difficult to make a positive fault identification.

The overall network accuracy for detecting errors in the rate sensor is given in figure 6.14. The resilient backpropagation networks have an average detection accuracy of 33%. Increasing the number of neurons assigned to the networks did not appear to aid in fault detection for these networks. Excluding network BR20, the Levenberg-Marquardt networks have an average detection accuracy of 59%.

Table 6.7: The performance of the neural networks for detecting rate sensor faults

		RP20			RP30		RP40			RP60			BR20			BR30			LM20			LM30		
Case #	Description	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
38	Rate sensor bias increase	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
39	Rate sensor bias increase	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
40	Rate sensor bias increase	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
41	Rate sensor bias increase	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
42	Rate sensor bias increase	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
43	Rate sensor bias increase	1	4	20%	1	1	50%	1	2	33%	1	2	33%	0	1	0%	1	1	50%	1	1	50%	1	0	100%
44	Rate sensor bias increase	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
45	Rate sensor bias increase	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
58	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
59	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
60	Rate sensor goes dead	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
61	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
62	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
63	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
64	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
65	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%
66	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	0	0	0%
67	Rate sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	1	50%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
68	Rate sensor unknown noise	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%

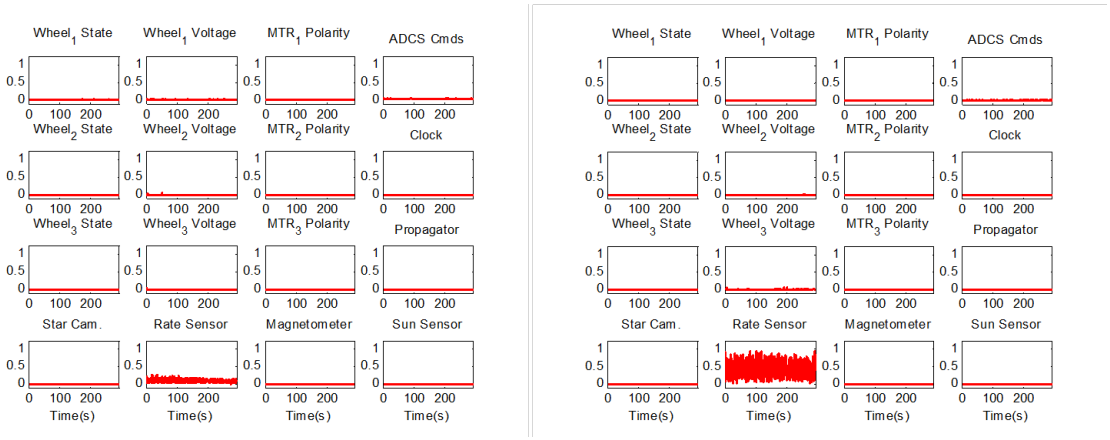

Fig. 6.12: Difference in network response

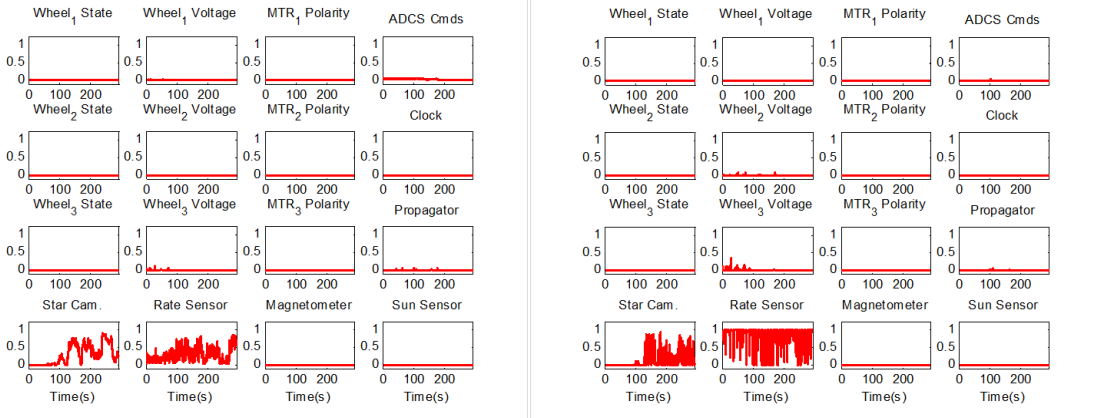

Fig. 6.13: Poor generalization of the RPROP networks (left) and the LM/BR networks (right)

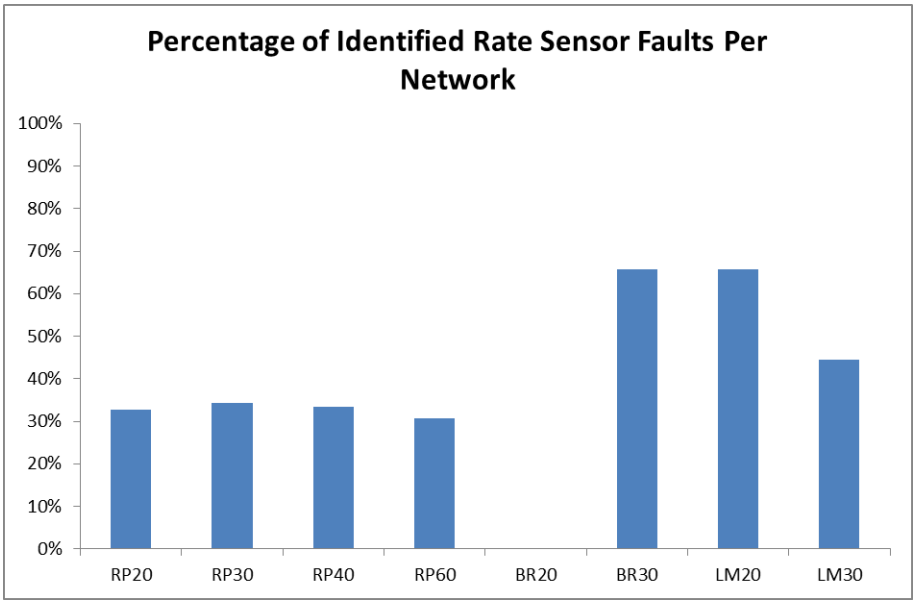

Fig. 6.14: The accuracy of the neural networks in detecting rate sensor faults

6.2.8 Performance in the Presence of Novel Data

In addition to calculating the network outputs for each of the 112 training cases, a number of novel residual sets was also generated and used as inputs to the neural networks. These novel residual sets represent data that the neural networks have no previous knowledge of. The novel data set is comprised of not only individual faults, but also of multiple simultaneous faults. Table 6.8 below summarizes the novel data.

Table 6.8: Summary of novel data cases

Case #	Description
113	Seize reaction wheels 1 & 2
114	Sieze reaction wheels 2 & 3
115	Reaction wheel 2 power loss; wrong guidance command uploaded
116	Magnetometer dead; propagator reset
117	Clock is wrong; propagator reset
118	Reaction wheel 2 suffers 90% power loss
119	All reaction wheels suffer 90% power loss
120	Seize reaction wheel 2
121	Rate sensor has unknown noise

To generate truly novel data, random scalars were used wherever possible, for example to scale the available torque authority or power to the reaction wheels. In all cases faults were injected at random intervals such that they could occur before, during, or after a maneuver. The performance of the neural networks in the presence of this novel data is summarized in table 6.9.

Examining the data, in most instances the neural networks have difficulty in isolating the multiple fault events, and instead report many false positive identifications. This result is not surprising. It was shown in the previous sections that the neural networks at times had difficulty distinguishing one fault from another if the faults produced similar patterns of residuals.

Those cases with single fault events generally report positive identifications, with those networks that positively identified certain faults previously doing so again, and those that were unable to previously demonstrating that same inability. Overall the neural networks positively identified the faulty components 37% of the time.

6.3 Summary of Results

Figure 6.15 presents the overall accuracies of the neural networks for both the validation and novel data sets. Across all networks there is a 62.5% chance of successfully isolating a faulty component based on the validation data and a 37% chance based on the novel data.

Table 6.9: Performance summary of the novel data cases

Case #	Description	RP20			RP30			RP40			RP60			BR20			BR30			LM20			LM30		
		#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct	#Pos	#False	Acc Pct
113	Sieze reaction wheels 1 & 2	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
114	Sieze reaction wheels 2 & 3	0	6	0%	0	8	0%	0	9	0%	0	9	0%	0	2	0%	0	4	0%	0	4	0%	0	4	0%
115	Wheel 2 power loss wrong ADCS command	0	2	0%	0	2	0%	0	4	0%	0	1	0%	0	1	0%	0	2	0%	0	1	0%	0	2	0%
116	Magnetometer is dead propagator reset	1	1	50%	1	1	50%	1	2	33%	1	1	50%	1	2	33%	1	0	100%	1	0	100%	1	2	33%
117	clock is wrong and propagator reset	0	1	0%	1	4	20%	0	3	0%	0	1	0%	0	2	0%	0	1	0%	0	2	0%	1	3	25%
118	90% wheel 2 power loss	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	1	0	100%	1	0	100%
119	all wheels 90% power loss	1	6	14%	1	8	11%	1	9	10%	1	6	14%	0	4	0%	1	6	14%	0	5	0%	0	4	0%
120	seize reaction wheel 2	1	0	100%	1	0	100%	1	0	100%	1	0	100%	1	0	100%	0	0	0%	1	0	100%	0	0	0%
121	Rate sensor unknown noise	0	0	0%	0	0	0%	0	0	0%	0	0	0%	0	0	0%	1	0	100%	1	0	100%	1	0	100%

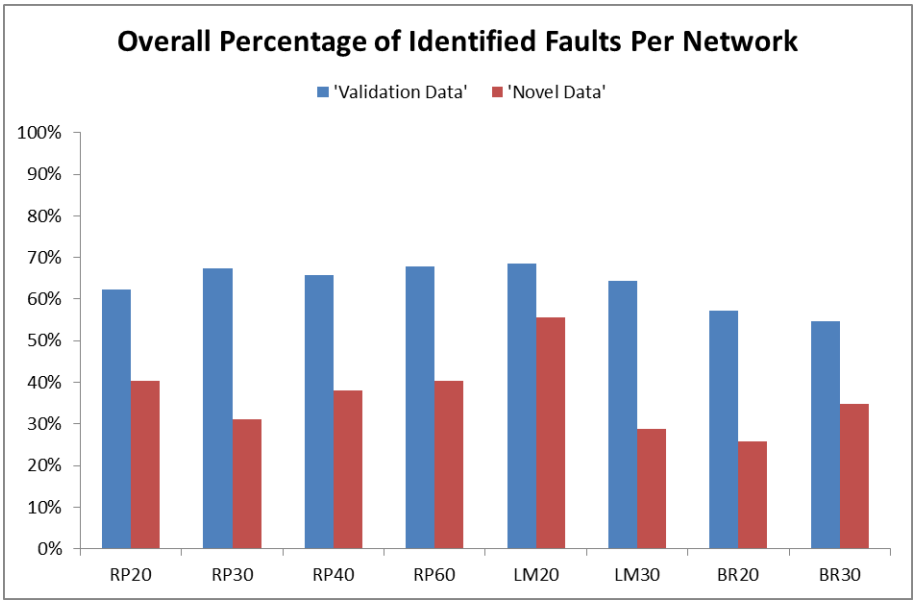

Fig. 6.15: The overall accuracy of each neural network

The networks trained with the resilient backpropagation algorithm all have similar overall rates of fault detection with an average of 64% across all data. Those networks trained with either the Levenberg-Marquardt algorithm or its Bayesian regulation extension vary in accuracy between 53% (BR30) to 68% (LM20) across all data. Based on the results of the validation and novel data sets it is difficult to say with certainty which, if either, of the training algorithms is superior or what number of neurons to include in the neural network is optimal. Each type of network displayed strengths and weaknesses with regard to positively identifying the faults in the ADCS. A few observations can be made, however.

Resilient backpropagation with a high number of neurons was able to produce better generalized networks with regard to positively identifying actuator faults. The best performing Levenberg-Marquardt network in this series had 30 neurons, was unregulated, and had a positive detection rate of 47%, compared to the equivalent network trained with resilient backpropagation that had an accuracy of 71%. The highest performing network in this category, RP60, was able to successfully identify 79% of the actuator faults. The Levenberg-Marquardt networks demonstrated superior performance in identifying rate sensor faults, having almost twice the successful detection rate, 66%, as the resilient backpropagation networks, 34%. The LM networks again demonstrated high detection fidelity

with respect to ADCS commanding faults, with the exception that network RP30 was at least on par with these.

In an overall sense neither of the Bayesian regulated Levenberg-Marquardt networks generalized very well, having an average detection accuracy of only 56% based on the validation data. Although many of the networks produced overall accuracies greater than 60% with the validation data, only one network came close to approaching even that with the novel data, network LM20, also at 56%. Table 6.10 provides a final summary of the highest and average detection accuracies for each fault, and the corresponding best network.

Table 6.10: Summary of highest and average detection accuracies and the neural networks that produced them for each of the fault categories

Fault	Best	Avg	Best Network
Actuators	79%	57%	RP60
ADCS Cmds	83%	65%	LM20, BR30
Propagator/Clock	100%	96%	RP20, RP30, RP40, RP60, BR30, LM30
Star Camera	42%	27%	BR20
Magnetometer	100%	93%	LM20, BR30
Sun Sensor	94%	74%	LM30
Rate Sensor	66%	40%	LM20, BR30

These results can be expanded upon further. Table 6.10 summarizes the performance of each network relative to isolating a faulty component of an ADCS. These components can each be grouped into subsystems of the ADCS: attitude control (AC), guidance, and attitude determination (AD). Grouping the ADCS components into subsystems as per table 6.11 allows for further analysis of the performance of the neural networks by measuring how well they were able to isolate the fault to the proper subsystem, if not to the proper component. For example if a fault was injected into the star camera and the neural network identified the fault as a problem with the rate sensor, even though it misidentified the faulty component, in either instance it properly identified the fault as being in the attitude determination subsystem.

Table 6.11: The subsystems of the ADCS with their corresponding components

Attitude Control	Guidance	Attitude Determination
Reaction Wheels	Guidance Commands	Star Camera
Magnetic Torquers	Clock	Rate Sensor
	Propagator	Magnetometer
		Sun Sensor

The performance of the neural networks relative to isolating a fault to a particular subsystem is summarized in figure 6.16. Those networks trained with resilient backpropagation had a 75% success rate in isolating the faulty subsystem, compared to a 64% chance in isolating a faulty component. The Levenberg-Marquardt networks had almost identical success rates of 66% and 65% for identifying the faulty subsystem or component, whereas the Bayesian regulated Levenberg-Marquardt networks had only a marginally higher chance to isolate the faulty subsystem, 59%, than it did isolating a faulty component, 54%. Matlab scripts for calculating subsystem fault identification accuracy are provided in appendices B.6 and B.7

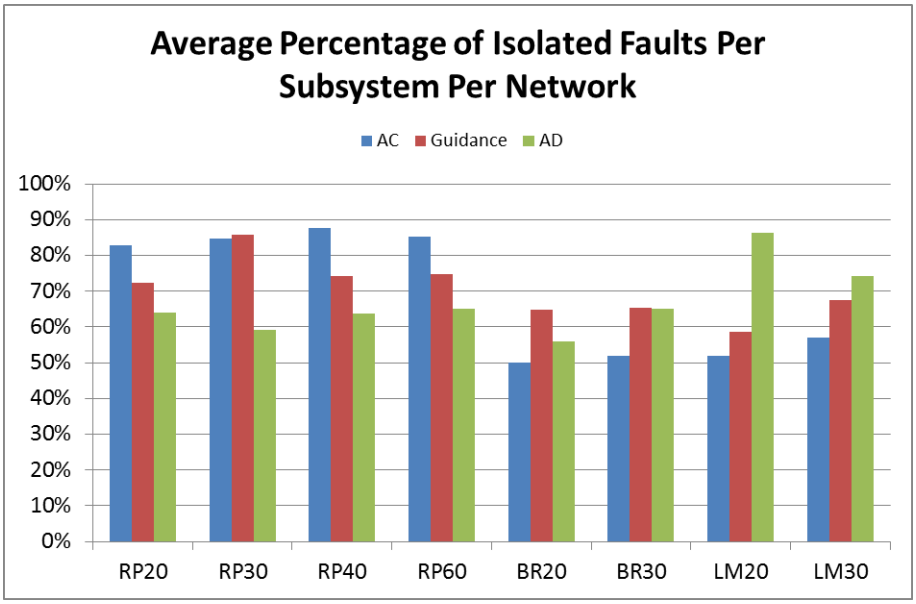

Fig. 6.16: The performance of the neural networks in isolating faults to a particular subsystem

From an overall standpoint (reference figure 6.15) it would be difficult to make a determination as to which network was more successful in isolating faulty components. From the standpoint of isolating a faulty subsystem, perhaps more of a generalization can be made. From figure 6.16 it is easily seen that the resilient backpropagation networks excel in isolating attitude control and guidance subsystem faults, while their performance in isolating attitude determination faults is no better than the other networks. The four Levenberg-Marquardt networks have about the same positive isolation rate for the attitude control and guidance subsystems, but the success rate in isolating attitude determination faults varies between 56% (BR20) to 86% (LM20). This inconsistency in isolating attitude determination faults, coupled with the generally low isolation rates for attitude control and guidance faults, makes the case for use of the resilient backpropagation training algorithm for isolating faults on both a component and subsystem basis.

Chapter 7

Discussion and Future Considerations

7.1 Discussion

This thesis has presented a method of fault detection and diagnosis in a spacecraft's attitude determination and control system. Using an offline model of a spacecraft's ADCS, dynamics, and environment as a state estimator and to act as a reference, a vector of residuals was defined in conjunction with the telemetry from a modeled spacecraft's ADCS. The purpose of this vector is to act as the state of the spacecraft and provides an instantaneous snapshot as to the status of the ADCS. The modeled spacecraft was then subjected to a battery of fault injection scenarios, where the residual vector from each scenario was recorded at every timestep and saved in order to construct a library of example data. Using this library, several neural networks were trained, and their outputs analyzed. The performance of the neural networks in detecting faults with this scheme leaves much to interpretation. Though all of the networks were trained from the same example set, significant differences exist in the ability of the networks to positively detect and identify the faults with any consistency. Where one network may excel in detecting a certain fault, it may fare poorly at another, as evidenced by an average overall performance of 64% across all networks.

Certain discoveries made during the course of this research can perhaps shed light on this. First, the definition of the residual space warrants examination. The residuals that define the state of the spacecraft were chosen to be as broad as possible with respect to the state of the spacecraft's attitude, actuators, and sensors, and defined with redundancies built in where possible. In theory, this would differentiate the state of the spacecraft as much as possible so that one fault did not produce residuals that were not sufficiently dissimilar from another. In practice, this introduced confusion into the residuals, making it more difficult to distinguish the residuals between faults. The residuals were further muddled

by the fusion of the star camera and rate sensor within the Kalman filter, as the residuals generated by either sensor would experience a bias when either sensor was in error.

Perhaps the most obfuscating factor, though, is the size and complexity of the problem being examined. Employing this method to identify even a modest number of faults in a modest number of target systems required well over 100 faults to be modeled, with almost 2,000,000 total training examples generated. Expansion of this method to include other systems, or training for other faults, increases the complexity further, so much so that consideration must be given not only to the capabilities of the computational hardware used to train the neural networks, but also to the capability of a neural network to generalize the larger problem. to determine the number of positively identified and falsely identified faults for the set of training data, as well as new, novel data of which the networks had no *a priori* knowledge.

The method of fault detection and diagnosis presented in this thesis encompasses an entire attitude determination and control system. Other authors have also developed FDD algorithms that are more compartmentalized and focus on only a single subsystem of an ADCS. Pirmoradi et al, for example, have proposed a scheme for diagnosing faults in a spacecraft's attitude determination system [3]. In the proposed scheme faults in a spacecraft's attitude determination sensors are detected by processing the sensor residuals through a series of extended Kalman filters, and the source of the fault is isolated through statistical analysis. Although the example cited in the paper used only a single rate sensor and single vector sensor, it could be expanded to include a spacecraft's full complement of attitude determination sensors by increasing the number of Kalman filters employed during primary isolation. The method proposed by Pirmoradi, et al, is limited to use in a noisy environment by inclusion of the Kalman filters and would be constrained to fault detection in the attitude determination sensors; however, since it is the statistical analysis of the residuals generated by the Kalman filters that isolates the faults, it is not inconceivable that the Kalman filters could be replaced by classical observers for use in isolating faults in components whose residuals are more deterministic.

A method proposed by Li, et al, attempts to detect and isolate faults in a spacecraft's control actuators via use of a series of neural networks [4]. In this scheme, each of three reaction wheels is assigned a neural network which is used to estimate the reaction torque. During operation the torques commanded by the controller and the outputs of the wheels act as the inputs to the neural networks. The difference between the estimated and actual torques serves as the residual, which is then thresholded to determine whether or not a fault has occurred. Since Li's neural networks serve only to generate residuals, it is feasible that this method could be employed in either a random or deterministic environment.

Although these three schemes are all proposed to detect faults in spacecraft attitude determination systems, key differences exists between them. The method proposed in this thesis attempts to detect and isolate faults in the entire ADCS, rather than only in the attitude control system (Li) or the attitude determination system (Pirmoradi). It also makes an attempt at determining when the instructions, i.e. the guidance commands, to the ADCS are the source of error. The choice of state estimators between methods is also a key difference. Pirmoradi and Li employ either Kalman filters or neural networks to serve as residual generators. This thesis employs a ground-side reference model in combination with the spacecraft's guidance and control commands to serve as the residual generators. This structure of residual generation would preclude use of this method onboard a spacecraft, and, therefore, prohibit also the chance for real-time fault detection –a marked contrast to either Pirmoradi's or Li's methods, which suffer no such constraint. Another key difference is the function of the neural networks used in this thesis and by Li. Whereas Li uses neural networks only as residual generators, this thesis uses neural networks as residual *analyzers*. It is through analysis of a residual set by a neural network that a fault determination is made, rather than by thresholding of a residual generated by a neural network.

The performance of the neural networks in detecting faults with this scheme leaves much to interpretation. Though all of the networks were trained from the same example set, significant differences exist in the ability of the networks to positively detect and identify the faults with any consistency. Where one network may excel in detecting a certain fault,

it may fare poorly at another, as evidenced by an average overall performance of 64% across all networks.

Certain discoveries made during the course of this research can perhaps shed light on this. First, the definition of the residual space warrants examination. The residuals that define the state of the spacecraft were chosen to be as broad as possible with respect to the state of the spacecraft's attitude, actuators, and sensors, and defined with redundancies built in where possible. In theory, this would differentiate the state of the spacecraft as much as possible so that one fault did not produce residuals that were not sufficiently dissimilar from another. In practice, this introduced confusion into the residuals, making it more difficult to distinguish the residuals between faults. The residuals were further muddled by the fusion of the star camera and rate sensor within the Kalman filter, as the residuals generated by either sensor would experience a bias when either sensor was in error.

Perhaps the most obfuscating factor, though, is the size and complexity of the problem being examined. Employing this method to identify even a modest number of faults in a modest number of target systems required well over 100 faults to be modeled, with almost 2,000,000 total training examples generated. Expansion of this method to include other systems, or training for other faults, increases the complexity further, so much so that consideration must be given not only to the capabilities of the computational hardware used to train the neural networks, but also to the capability of a neural network to generalize the larger problem.

7.2 Future Considerations

Several considerations could be made with regard to the future direction of this research, especially with respect to the system residuals. Recall that the primary driver behind the low component-level isolation rates was caused by "residual confusion," where the residuals of one fault sometimes closely resembled the residuals of another fault. This confusion of the residuals impacts the ability of the neural networks to distinguish one fault from another, leading to poor generalization. A future path of research then would be to determine if there is a better way to distinguish between residuals. One proposed method

would be normalization, whereby all of the residuals could be placed onto a common scale. Having a common scale for the residuals may allow for better generalization of the neural networks and aid in pattern recognition.

Another factor relative to the residuals that warrants further attention is drift. The residuals in this thesis are largely defined as functions of either sensors or software, the outputs of which, over time, will experience a certain degree of drift. Drift over long periods of time were not explicity considered as part of this research. As such, were data collected over a long enough time period, it is entirely likely that the sensor and software outputs could drift enough such that the residuals would begin to become biased not due to the presence of a fault, but rather to the presence of drift instead. An important question then that remains unanswered is how long could data be collected and analyzed before drfit begins to bias the residuals and lead to false positive fault identification? How long could the residuals be analyzed before the reference model would need to be resynchronized with the telemetry in order to account for drift? The timescale over which the residuals are collected and analyzed is then of the utmost importance, as too long a time scale would allow the accumulation of drift, and too short a timescale would require constant resynchronization.

References

[1] Ure, N., Kaya, Y., and Inalhan, G., "The Development of a Software and Hardware-in-the-Loop Test System for ITU-PSAT II Nano Satellite ADCS," *IEEE Aerospace Conference*, March 2011, pp. 1–15.

[2] Boskovic, J. D., Li, S. M., and Mehra, R. K., "Intelligent Control of Spacecraft in the Presence of Actuator Failures," *Proceedings of the 38th IEEE Conference on Decision and Control*, Vol. 5, IEEE, 1999, pp. 4472–4477.

[3] Pirmoradi, F., Sassani, F., and De Silva, C., "Fault Detection and Diagnosis in a Spacecraft Attitude Determination System," *Acta Astronautica*, Vol. 65, No. 5, 2009, pp. 710–729.

[4] Li, Z., Ma, L., and Khorasani, K., "Fault Diagnosis of an Actuator in the Attitude Control Subsystem of a Satellite Using Neural Networks," *International Joint Conference on Neural Network*, IEEE, 2007, pp. 2658–2663.

[5] Izadi-Zamanabadi, R. and Larsen, J. A., "A Fault Tolerant Control Supervisory System Development Procedure for Small Satellites: The AAUSAT-II Case," *IFAC Symposium on Automatic Control in Aerospace*, 2007.

[6] Bidner, F., "Fault Tree Analysis of the HERMES CubeSat," *University of Colorado at Boulder, USA*, 2010.

[7] Brumbaugh, K. M. and Lightsey, E. G., "Application of Risk Management to University CubeSat Missions," *Journal of Small Satellites*, Vol. 2, No. 1, 2013, pp. 147–160.

[8] Boskovic, J., Li, S. M., and Mehra, R. K., "Intelligent Spacecraft Control Using Multiple Models, Switching, and Tuning," *Proceedings of the IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, IEEE, 1999, pp. 84–89.

[9] Barua, A., Sinha, P., and Khorasani, K., "A Diagnostic Tree Approach for Fault Cause Identification in the Attitude Control Subsystem of Satellites," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 45, No. 3, 2009, pp. 983–1002.

[10] Izadi-Zamanabadi, R., "Structural Analysis Approach to Fault Diagnosis with Application to Fixed-wing Aircraft Motion," *Proceedings of the American Control Conference*, Vol. 5, IEEE, 2002, pp. 3949–3954.

[11] Narendra, K. S. and Balakrishnan, J., "Adaptive Control Using Multiple Models," *IEEE Transactions on Automatic Control*, Vol. 42, No. 2, 1997, pp. 171–187.

[12] Shuster, M. D., "The TRIAD Algorithm as Maximum Likelihood Estimation," *The Journal of the Astronautical Sciences*, Vol. 54, No. 1, 2006, pp. 113–123.

[13] Sidi, M., *Spacecraft Dynamics and Control*, Cambridge University Press, Cambridge, UK, 1997.

[14] Dorf, R. C. and Bishop, R. H., *Modern Control Systems*, Pearson (Addison-Wesley), Menlo Park, CA, 1998.

[15] Crassidis, J. L. and Junkins, J. L., *Optimal Estimation of Dynamic Systems*, CRC Press, Boca Raton, FL, 2011.

[16] Curtis, H., *Orbital Mechanics for Engineering Students*, Butterworth-Heinemann, Burlington, MA, 2013.

[17] Larson, W. J. and Wertz, J. R., *Space Mission Analysis and Design*, Microcosm Press, El Segundo, CA, 2nd ed., 1999.

[18] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, New Jersey, 2nd ed., 1999.

[19] Holick, A., "Analysis of Noncatastrophic Failures in Digital Guidance Systems," *IEEE Transactions on Electronic Computers*, Vol. EC-12, No. 4, 1963, pp. 365–371.

[20] Gulmammadov, F., "Analysis, Modeling and Compensation of Bias Drift in MEMS Inertial Sensors," *4th International Conference on Recent Advances in Space Technologies*, IEEE, 2009, pp. 591–596.

[21] Riedmiller, M. and Braun, H., "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *IEEE International Conference on Neural Networks*, IEEE, 1993, pp. 586–591.

[22] Marquardt, D. W., "An Algorithm for Least-squares Estimation of Nonlinear Parameters," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, No. 2, 1963, pp. 431–441.

[23] Hagan, M. T. and Menhaj, M. B., "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, 1994, pp. 989–993.

[24] Foresee, F. D. and Hagan, M. T., "Gauss-Newton Approximation to Bayesian Learning," *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, IEEE, 1997, pp. 1930–1935.

Appendices

Appendix A

Summary of Fault Modeling

The purpose of this appendix is to provide a complete listing of the simulated faults, summarized in the following tables. The headings of the summary tables are defined below:

Table A.1: Description of the summary headings

Heading	Description
Case	Case designation: 1-112
Tfault	Time at which fault occurs
Duration	Time, in seconds, that the fault persists
TrqScale	Scaling parameter, e.g. how much to scale wheel torque
FaultNum	Denotes the row element of the neural net target vector
Description	Short description of the fault
phi/theta/psi	The commanded euler angles for the maneuver
rate	Commanded angular rate, deg/s
Threshold Value	Value used to trip fault designation in target vector

Tables A.2 - A.6 list the cases that were simulated to provide the training data for the neural networks.

Table A.2: Summary of simulated faults cases 1-24

Case	Tfault	Duration	TrqScale	FaultNum	Description	phi	theta	psi	rate	Threshold Value
1	145	10	0	2	Seize Reaction Wheel 2	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
2	145	10	0.25	2	Seize Reaction Wheel 2	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
3	145	10	0.5	2	Seize Reaction Wheel 2	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
4	145	10	0.75	2	Seize Reaction Wheel 2	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
5	0	0	1	0	Wheel 2 Nominal Operation	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
6	145	3000	0	5	Wheel 2 Power Loss	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
7	145	3000	0.25	5	Wheel 2 Power Loss	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
8	145	3000	0.5	5	Wheel 2 Power Loss	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
9	145	3000	0.75	5	Wheel 2 Power Loss	0	180	0	2	5 RPM 3 sigma.. used 5 sigma
10	0	3000	1	8	Torquer 2 Reverse Polarity	0	0	0	0	>0
11	0	3000	1	8	Torquer 2 Reverse Polarity	0	180	0	0	>0
12	0	3000	1	10	Wrong ADCS cmds uploaded	180	0	0	2	0.002
13	0	3000	1	10	Wrong ADCS cmds uploaded	0	-180	0	2	0.002
14	0	3000	1	11	Clock set incorrectly (JD is wrong)	0	180	0	2	6.9e-4 (1 mins in Julian time)
15	0	3000	1	11	Clock set incorrectly (JD is wrong)	0	180	0	2	6.9e-4 (1 mins in Julian time)
16	145	10	0	1	Seize Reaction Wheel 1	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
17	145	10	0.25	1	Seize Reaction Wheel 1	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
18	145	10	0.5	1	Seize Reaction Wheel 1	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
19	145	10	0.75	1	Seize Reaction Wheel 1	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
20	145	3000	0	4	Wheel 1 Power Loss	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
21	145	3000	0.25	4	Wheel 1 Power Loss	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
22	145	3000	0.5	4	Wheel 1 Power Loss	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
23	145	3000	0.75	4	Wheel 1 Power Loss	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
24	145	10	0	3	Seize Reaction Wheel 3	0	0	180	2	5 RPM 3 sigma.. used 5 sigma

Table A.3: Summary of simulated faults cases 25-52

Case	Tfault	Duration	TrqScale	FaultNum	Description	phi	theta	psi	rate
25	145	10	0.25	3	Seize Reaction Wheel 3	0	0	180	2
26	145	10	0.5	3	Seize Reaction Wheel 3	0	0	180	2
27	145	10	0.75	3	Seize Reaction Wheel 3	0	0	180	2
28	145	3000	0	6	Wheel 3 Power Loss	0	0	180	2
29	145	3000	0.25	6	Wheel 3 Power Loss	0	0	180	2
30	145	3000	0.5	6	Wheel 3 Power Loss	0	0	180	2
31	145	3000	0.75	6	Wheel 3 Power Loss	0	0	180	2
32	0	3000	1	7	Torquer 1 Reverse Polarity	180	0	0	2
33	0	3000	1	9	Torquer 3 Reverse Polarity	0	0	180	2
34	0	0	1	0	Wheel 1 Nominal Operation	180	0	0	2
35	0	0	1	0	Wheel 3 Nominal Operation	0	0	180	2
36	145	3000	1	10	Guidance Command Delay	0	180	0	2
37	145	3000	1	10	Guidance Command Delay	0	180	0	2
38	60	30	1	16	Rate sensor thermal bias increase	0	0	0	0
39	60	30	1	16	Rate sensor thermal bias increase	0	0	0	0
40	60	30	1	16	Rate sensor thermal bias increase	0	0	0	0
41	60	30	1	16	Rate sensor thermal bias increase	0	0	0	0
42	60	30	1	16	Rate sensor thermal bias increase	0	0	0	0
43	60	30	1	16	Rate sensor thermal bias increase	0	180	0	2
44	60	30	1	16	Rate sensor thermal bias increase	0	180	0	2
45	60	30	1	16	Rate sensor thermal bias increase	0	180	0	2
46	145	10	1	13	Glint on star camera	0	0	0	0
47	145	10	1	13	Glint on star camera	0	0	0	0
48	145	10	1	13	Glint on star camera	0	0	0	0
49	145	10	1	13	Glint on star camera	0	180	0	0
50	145	10	1	13	Glint on star camera	0	180	0	0
51	145	10	1	13	Glint on star camera	0	180	0	0
52	145	10	1	13	Glint on star camera	0	180	0	0

Table A.4: Summary of simulated faults cases 53-84

Case	Tfault	Duration	TrqScale	FaultNum	Description	phi	theta	psi	rate	Threshold Value
53	145	60	1	13	Glint on star camera	0	180	0	0	TRIAD solution... .15
54	145	60	1	13	Glint on star camera	0	180	0	0	TRIAD solution... .15
55	145	60	1	13	Glint on star camera	0	180	0	0	TRIAD solution... .15
56	145	60	1	13	Glint on star camera	0	180	0	0	TRIAD solution... .15
57	145	60	1	13	Glint on star camera	0	180	0	0	TRIAD solution... .15
58	60	3000	1000	16	Rate sensor unknown noise	0	0	0	0	0.05
59	60	3000	500	16	Rate sensor unknown noise	0	0	0	0	0.05
60	60	3000	1	16	Rate sensor goes dead	0	0	0	0	0.05
61	60	3000	250	16	Rate sensor unknown noise	0	0	0	0	0.05
62	60	3000	10000	16	Rate sensor unknown noise	0	0	0	0	0.05
63	0	3000	250	16	Rate sensor unknown noise	0	180	0	0	0.05
64	0	3000	500	16	Rate sensor unknown noise	0	180	0	0	0.05
65	0	3000	1000	16	Rate sensor unknown noise	0	180	0	0	0.05
66	0	3000	10000	16	Rate sensor unknown noise	0	180	0	0	0.05
67	0	3000	100000	16	Rate sensor unknown noise	0	180	0	0	0.05
68	0	3000	100000	16	Rate sensor unknown noise	0	0	0	0	0.05
69	0	3000	1	0	Nominal hold maneuver	0	0	0	0	N/A
70	0	3000	5	14	Magnetometer unknown noise	0	0	0	0	.01 1
71	0	3000	10	14	Magnetometer unknown noise	0	0	0	0	.01 1
72	0	3000	2.5	14	Magnetometer unknown noise	0	0	0	0	.01 1
73	0	3000	0	14	Magnetometer goes dead	0	0	0	0	.01 1
74	0	3000	0	14	Magnetometer goes dead	0	0	0	0	.01 1
75	0	3000	0	14	Magnetometer unknown noise	0	180	0	0	.01 1
76	0	3000	0	14	Magnetometer unknown noise	0	180	0	0	.01 1
77	0	3000	2.5	14	Magnetometer unknown noise	0	180	0	0	.01 1
78	0	3000	1.5	15	Sun sensor unknown noise	0	0	0	0	.004 3 sigma... used 5 sigma
79	0	3000	2	15	Sun sensor unknown noise	0	0	0	0	.004 3 sigma... used 5 sigma
80	0	3000	2.5	15	Sun sensor unknown noise	0	0	0	0	.004 3 sigma... used 5 sigma
81	0	3000	5	15	Sun sensor unknown noise	0	0	0	0	.004 3 sigma... used 5 sigma
82	0	3000	10	15	Sun sensor unknown noise	0	0	0	0	.004 3 sigma... used 5 sigma
83	0	3000	2.5	15	Sun sensor unknown noise	0	180	0	0	.004 3 sigma... used 5 sigma
84	0	3000	5	15	Sun sensor unknown noise	0	180	0	0	.004 3 sigma... used 5 sigma

Table A.5: Summary of simulated faults cases 85-110

Case	Tfault	Duration	TrqScale	FaultNum	Description	phi	theta	psi	rate	Threshold Value
85	0	3000	10	15	Sun sensor unknown noise	0	180	0	0	.004 3 sigma... used 5 sigma
86	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
87	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
88	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
89	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
90	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
91	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
92	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
93	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
94	0	3000	1	15	Sun sensor value is stuck	0	0	0	0	.004 3 sigma... used 5 sigma
95	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
96	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
97	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
98	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
99	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
100	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
101	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
102	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
103	0	3000	1	14	Magnetometer value is stuck	0	0	0	0	.01 1 sigma... used 5 sigma
104	145	3000	1	10	Guidance Command Delay	0	180	0	2	0.002
105	145	3000	1	10	Guidance Command Delay	0	180	0	2	0.002
106	145	3000	1	10	Guidance Command Delay	0	180	0	2	0.002
107	145	3000	1	10	Guidance Command Delay	0	180	0	2	0.002
108	145	3000	1	10	Guidance Command Delay	0	180	0	2	0.002
109	0	3000	1	12	Propagator drift	0	0	0	0	10
110	0	3000	1	12	Propagator drift	0	0	0	0	10

Table A.6: Summary of simulated faults cases 111-121

Case	Tfault	Duration	TrqScale	FaultNum	Description	phi	theta	psi	rate	Threshold Value
111	0	3000	1	12	Propagator drift	0	0	0	0	10
112	0	3000	1	12	Propagator reset	0	0	0	0	10
VALIDATION CASES START AT 113										
113	100+10randn	10	[.2+.1*randn .3+.1*randn]	[1 2]	Sieze reaction wheels 1 & 2	90	90	0	2	5 RPM 3 sigma.. used 5 sigma
114	100+10*randn	10+5*rand	[rand rand]	[2 3]	Sieze reaction wheels 2 & 3	90	90	0	2	5 RPM 3 sigma.. used 5 sigma
115	100+10*randn	10+5*rand	rand	5 10	Wheel 2 power loss, wrong ADCS command	180	0	0	2	5 RPM 3 sigma.. used 5 sigma
116	0	3000	1	12 14	Magnetometer is dead, propagator reset	0	0	0	0	10, .01*5sigma
117	0	3000	1	11 12	clock is wrong and propagator reset	0	0	0	0	6.9e-4, 10
118	100+10*randn	3000	0.1	5	90% wheel 2 power loss	0	0	0	0	5 RPM 3 sigma.. used 5 sigma
119	100+10*randn	3000	0.1	4 5 6	all wheels 90% power loss	0	0	0	0	5 RPM 3 sigma.. used 5 sigma
120	100+10*randn	30*rand	rand	2	seize reaction wheel 2	0	180	0	0	5 RPM 3 sigma.. used 5 sigma
121	0	3000	100000*rand	16	Rate sensor unknown noise	0	0	0	0	0.05

Appendix B

Matlab Scripts

Following are the scripts used to generate the residual training and target vectors used to train the neural networks, as well as the scripts used for post processing the neural network outputs.

B.1 Residual Training

This script (MakeNNDataVector.m) takes the output of the reference and fault model simulations, calculates the residuals, and packages them.

```
% Configure data to train a neural network
% In general the residuals are defined as "observed - expected"

%% Attitude Reference Checks
% Check the attitude as reported by spacecraft to those
% from the reference model. If the SC is doing what it should be doing then
% the attitude and guidance parameters should have a small residual, which
% is indicative of "all is well" and that the spacecraft is likely in its
% desired orientation.

%Kalman Filter estimated attitude from SC
ATTEstFLT = def.Fault.Sensor.EstimatedQuaternion(1:10:end,:);
AngEstFLT = 2*acos(ATTEstFLT(:,4));

%Kalman Filter estimated attitude from SC
ATTEstREF = def.Observer.Sensor.EstimatedQuaternion(1:10:end,:);
AngEstREF = 2*acos(ATTEstREF(:,4));

ATTRes = ATTEstFLT - ATTEstREF; %quaternion residual
```

```
20 AngResREF = AngEstFLT - AngEstREF; %attitude residual
21
22 %Kalman Filter estimated rate from SC
23 WEstFLT = def.Fault.Sensor.EstimatedAngularRate(1:10:end,:);
24
25 %Kalman Filter estimated rate from Reference
26 WEstREF = def.Observer.Sensor.EstimatedAngularRate(1:10:end,:);
27 n = length(def.Time(1:10:end));
28
29 RateEstREF= zeros(n,1);
30 RateEstFLT= zeros(n,1);
31 ATTresREF = zeros(n,1);
32 RateResREF = zeros(n,1);
33
34 for i=1:n
35     ATTresREF(i) = norm(ATTRes(i,:)); %use quaternion norm
36     RateEstREF(i) = norm(WEstREF(i,:)); %use norm of rate estimate
37     RateEstFLT(i) = norm(WEstFLT(i,:)); %use norm of rate estimate
38     RateResREF(i) = norm(WEstFLT(i,:)-WEstREF(i,:)); %rate residual
39 end
40
41 % RateResREF = RateEstFLT - RateEstREF; %rate residual
42 NeuralNet.Attitude.AngResREF = AngResREF;
43 NeuralNet.Attitude.ATTresREF = ATTresREF;
44 NeuralNet.Attitude.RateResREF = RateResREF;
45
46 clearvars RateResREF ATTresREF RateEstREF WEstREF WEstFLT ...
47     AngEstREF ATTEstREF AngEstREF
48 %% Guidance Reference Checks
49 % By checking the echoed attitude/rate commands from the spacecraft versus
50 % those estimated by the reference, one can determine that if the
51 % spacecraft is out of orientation is it simply due to it receiving the
52 % wrong commands
53
54 %Attitude commands as echoed from SC
```

```
ATTcmdFLT = def.Fault.Guidance.AttitudeCmd(1:10:end,:);
AngCmdFLT = 2*acos(ATTcmdFLT(:,4));

%Attitude commands as echoed from SC
ATTcmdREF = def.Observer.Guidance.AttitudeCmd(1:10:end,:);
AngCmdREF = 2*acos(ATTcmdREF(:,4));

AngCmdResREF = AngCmdFLT - AngCmdREF; %rotation angle command residual
ATTCmdRes = ATTcmdFLT - ATTcmdREF; %quaternion command residual

%Attitude commands as echoed from SC
WCmdFLT = def.Fault.Guidance.RateCmd(1:10:end,:);
%Attitude commands as echoed from reference
WCmdREF = def.Observer.Guidance.RateCmd(1:10:end,:);
RateCmdFLT = zeros(n,1);
RateCmdREF = zeros(n,1);
ATTCmdResREF = zeros(n,1);
RateCmdResREF = zeros(n,1);
for i=1:n
    ATTCmdResREF(i) = norm(ATTCmdRes(i,:));%use norm of attitude quaternion
%     RateCmdFLT(i) = norm(WCmdFLT(i,:)); %use norm of rate command
%     RateCmdREF(i) = norm(WCmdREF(i,:)); %use norm of rate command
    RateCmdResREF(i) = norm(WCmdFLT(i,:)-WCmdREF(i,:)); %rate cmd residual
end

% RateCmdResREF = RateCmdFLT-RateCmdREF; %rate command residual
NeuralNet.Guidance.ATTCmdResREF = ATTCmdResREF;
NeuralNet.Guidance.AngCmdResREF = AngCmdResREF;
NeuralNet.Guidance.RateCmdResREF = RateCmdResREF;

clearvars WCmdFLT WCmdREF WEstFLT WEstREF ATTcmdFLT AngCmdFLT ATTcmdREF ...
    AngCmdREF ATTCmdResREF WCmdFLT WCmdREF RateCmdFLT RateCmdREF ...
    RateCmdResREF ATTCmdResREF

%% Actuator Checks
```

```
90 % Do not compare actuator telemetry to reference model. If attitude/rate
91 % are ok, the actuators will be working correctly. If attitude or rate are
92 % off w.r.t. the reference model, then it goes without saying that the
93 % actuators will be off as well; however, this is not indicative of a fault
94 % in the actuators, for example if the desired orientation was [0 0 0 1]
95 % and the spacecraft went to [0 1 0 0] the actuators are working fine.
96 %
97 % Instead check the actuator outputs to their command inputs to determine
98 % their operational state.
99
100 % Wheel speed commands and output
101 WheelSpdAct = def.Fault.Actuator.WheelSpdAct(1:10:end,:);
102 WheelSpdCmd = def.Fault.Actuator.WheelSpdCmd(1:10:end,:);
103 WheelSpdRes = WheelSpdAct-WheelSpdCmd;
104
105 % Wheel voltage commmands and output
106 WheelVoltageAct = def.Fault.Actuator.WheelVoltageAct(1:10:end,:);
107 WheelVoltageCmd = def.Fault.Actuator.WheelVoltageCmd(1:10:end,:);
108 WheelVoltageRes = WheelVoltageAct-WheelVoltageCmd;
109
110 % Mag torquer polarity
111 MagPolCmd = -sign(def.Fault.Actuator.MagTrqCmd(1:10:end,:)); %Note minus sign!
112 MagPolAct = sign(def.Fault.Actuator.MagTrqAct(1:10:end,:));
113 MagPolRes = MagPolAct-MagPolCmd;
114
115 NeuralNet.Actuator.WheelSpdRes = WheelSpdRes;
116 NeuralNet.Actuator.WheelVoltageRes = WheelVoltageRes;
117 NeuralNet.Actuator.MagPolRes = MagPolRes;
118
119 clearvars WheelSpdRes WheelVoltageRes MagPolRes WheelSpdAct WheelSpdCmd ...
120     WheelVoltageAct WheelVoltageCmd MagPolCmd MagPolAct
121
122 %% Propagator Checks
123 % The purpose of the propagator checks is to determine whether or not the
124 % attitude is out of place due to an error in the propagator. By comparing
```

```
125 % the propagator's output to the validated reference model the spacecraft's
126 % position in space and time can be confirmed. If it is determined that the
127 % propagator is out of synch with the reference model, it would be an
128 % extremely likely source of attitude error. Note that propagator errors
129 % can be due, for example, to clock errors, or orbit ephemeris updates.
130
131 %spacecraft echoed Julian date
132 JulianDateFLT = def.Fault.Propagator.JulianDate(1:10:end);
133
134 %expected JD from reference model
135 JulianDateREF = def.Observer.Propagator.JulianDate(1:10:end);
136 JulianDateRes = JulianDateFLT-JulianDateREF;
137
138 InertialPositionFLT = def.Fault.Propagator.InertialPosition;
139 InertialPositionREF = def.Observer.Propagator.InertialPosition;
140 PosVecRes = InertialPositionFLT-InertialPositionREF;
141
142 InertialMagFieldFLT = def.Fault.Propagator.InertialMagField(1:10:end,:);
143 InertialMagFieldTruthFLT = def.Fault.Orbit.InertialMagField(1:10:end,:);
144 InertialMagFieldREF = def.Observer.Propagator.InertialMagField(1:10:end,:);
145
146
147 InertialSolarVectorFLT = def.Fault.Propagator.InertialSolarVector;
148 InertialSolarVectorREF = def.Observer.Propagator.InertialSolarVector;
149 MagVecRes = zeros(n,1);
150 PositionRes = zeros(n,1);
151 MagFieldRes = zeros(n,1);
152 SolarPositionRes = zeros(n,1);
153 SolVecRes = zeros(n,1);
154
155 % Calculate residual norms
156 for i=1:n
157     MagVecRes(i) = norm(InertialMagFieldFLT(i,:)-InertialMagFieldREF(i,:));
158     SolVec = real(acos(dot(InertialSolarVectorFLT(i,:),...
159         InertialSolarVectorREF(i,:))));
```

```
160     SolVecRes(i) = mod(SolVec,2*pi)*180/pi;
161     PositionRes(i) = norm(PosVecRes(i,:));
162     MagFieldRes(i) = norm(MagVecRes(i,:));
163     SolarPositionRes(i) = norm(SolVecRes(i,:));
164 end
165 
166 NeuralNet.Propagator.JulianDateRes = JulianDateRes;
167 NeuralNet.Propagator.PositionRes = PositionRes;
168 NeuralNet.Propagator.MagFieldRes = MagFieldRes;
169 NeuralNet.Propagator.SolarPositionRes = SolarPositionRes;
170 
171 clearvars JulianDateFLT JulianDateREF JulianDateRes InertialPositionFLT ...
172     InertialPositionREF PosVecRes PositionRes  ...
173     MagVecRes MagFieldRes InertialSolarVectorFLT InertialSolarVectorREF ...
174     SolVecRes SolarPositionRes
175 
176 %% Sensor Checks
177 % Performing the sensor checks allows us to determine whether or not either
178 % the sensors or the propagator are off in some way. The measured inertial
179 % magnetic field will be compared to the propagator field (the propagator
180 % magnetic field will be verified with a different test). The measured
181 % angular rate will be compared to the body rate estimates derived from the
182 % attitude quaternion and its time derivative. Finally the kalman filter
183 % attitude estimate derived from the star camera input will be compared to
184 % a TRIAD algorithm attitude based on the measured magnetic field and
185 % propagator supplied nadir vector.
186 %
187 % **Note that if the propagator is in a fault state the TRIAD solution will
188 % also be in error.**
189 
190 % Compare measured magnetic field to
191 % MagFieldMeasured = def.Fault.Sensor.SensedBodyMagField(1:10:end,:);
192 MagSensorRes = zeros(n,1);
193 MagFieldProp = zeros(n,3);
194 SunSensorRes = zeros(n,1);
```

```
195 Sun_I_measured = zeros(n,3);
196
197 % Compare measured body rates to estimated body rates (Sidi 7.2.20)
198 BodyRateEstimate = def.Fault.Sensor.BodyRateEstimate; %Sidi 7.2.20
199
200 %from kalman filter
201 BodyRateMeasured = def.Fault.Sensor.EstimatedAngularRate(1:10:end,:);
202
203 RateSensorRes = zeros(n,1);
204 AttTriad = zeros(n,4);
205 AngTriad = zeros(n,1);
206 AttTriadRes = zeros(n,1);
207 Nadir_ORF = [0;0;1];
208 nadir_check = zeros(n,1);
209 % calc residual norms
210 ResidualMagField = def.init.sensors.mag.SC_Residual_Mag_Field;
211 sigma_mag = .01;
212 % sigma_sun = .0005*1;
213 sigma_sun = 1*5; %degrees
214 P = zeros(n,3);
215 Radical = zeros(n,4);
216 for i=1:n
217 % for i=950:951
218 %     i=949
219     R_ORF_to_b = def.Fault.Orbit.Rotation.OrbRef2Body(:,:,i);
220 %     R_ORF_to_b = def.Observer.Orbit.Rotation.OrbRef2Body(:,:,i);
221     R_i_to_ORF = def.Fault.Propagator.Inertial2OrbRefRotation(:,:,i);
222 %     R_i_to_ORF = def.Observer.Orbit.Rotation.Inertial2OrbRef(:,:,i);
223     % USE THIS ROTATION INSTEAD
224     R_i_to_b= R_ORF_to_b*R_i_to_ORF;
225     MagNoise = sigma_mag*[randn;randn;randn];
226     R_i_to_ORF_truth = def.Observer.Orbit.Rotation.Inertial2OrbRef(:,:,i);
227     R_i_to_b_truth = R_ORF_to_b*R_i_to_ORF_truth;
228
229     %uncertainty due to inaccurate mag field modeling
```

```
230     MagUncertainty = [cosd(10*rand);cosd(10*rand);cosd(10*rand)];
231
232     MagFieldMeasured = R_i_to_b*InertialMagFieldTruthFLT(i,:)'.*...
233         MagUncertainty+ResidualMagField;
234
235     %normalized measured mag field
236     B_BFCS_Unit = MagFieldMeasured/norm(MagFieldMeasured)+MagNoise;
237
238     R_ORF_to_i = R_i_to_ORF';
239
240
241     MagFieldProp(i,:) = (R_i_to_b*(InertialMagFieldFLT(i,:)'))';
242 %     MagFieldProp(i,:) = ((InertialMagFieldFLT(i,:)'))';
243     MagSensorRes(i) = norm(B_BFCS_Unit'-MagFieldProp(i,:)/...
244         norm(MagFieldProp(i,:)));
245     RateRes = BodyRateEstimate(i,:)-BodyRateMeasured(i,:);
246     RateSensorRes(i) = norm(RateRes)*180/pi;
247
248     % Setup Triad algorithm between ORF/BFCS solar and magnetic field
249     % vector
250     Sun_I = def.Fault.Propagator.InertialSolarVector(i,:);
251 %     Sun_I = def.Observer.Propagator.InertialSolarVector(i,:);
252     Sun_I = Sun_I/norm(Sun_I);
253
254     %COMMENT OUT WHEN FINISHED!!!
255 %     Sun_I(3) = 1;
256
257     Sun_B = R_i_to_b*def.Observer.Propagator.InertialSolarVector(i,:)';
258     Sun_B = ...
259         [cosd(sigma_sun*rand) 0 0;0 cosd(sigma_sun*rand) 0;...
260         0 0 cosd(sigma_sun*rand)]*Sun_B;
261
262     B_ECI_Unit = InertialMagFieldFLT(i,:)/norm(InertialMagFieldFLT(i,:));
263
264     % Calculate R_ORF2B from Triad algorithm; convert to quaternion
```

```
265
266     [R_i_to_b_4triad,Cov] = triad_with_covariance(Sun_I',B_ECI_Unit',...
267         Sun_B,B_BFCS_Unit,sigma_sun,sigma_mag);
268     P(i,:)=norm([sqrt(Cov(1,1)) sqrt(Cov(2,2)) sqrt(Cov(3,3))]);
269     R_ORF_to_b_triad = R_i_to_b_4triad*R_ORF_to_i;
270
271     [AttTriad(i,:),Radical(i,:)] = ...
272         rotation_matrix_to_quaternion(R_ORF_to_b_triad);
273     AttTriadRes(i) = norm(ATTEstFLT(i,:)-AttTriad(i,:));
274
275
276 %     AngTriad(i) = 2*acos(AttTriad(i,4));
277     AngTriad(i) = acos((trace(R_ORF_to_b_triad)-1)/2);
278
279     %rotate measured sun vector (B) to inertial coordinates to compare to
280     %propagator inertial sun vector
281     Sun_I_measured = R_i_to_b_4triad'*Sun_B;
282     SunSensorRes(i) = norm(Sun_I'-Sun_I_measured);
283     SunSensorRes(i) = norm(R_i_to_b_4triad*Sun_I'-Sun_B); %JUST A CHECK!
284 end
285
286 TriadRes = AttTriadRes;
287
288 NeuralNet.Sensor.Magnetometer = MagSensorRes;
289 NeuralNet.Sensor.SunSensor = SunSensorRes;
290 NeuralNet.Sensor.RateSensor = RateSensorRes;
291 NeuralNet.Sensor.Triad = TriadRes;
292
293 clearvars MagFieldMeasured InertialMagFieldFLT MagSensorRes MagRes ...
294     InertialMagFieldFLT R_i_to_b MagFieldProp MagSensorRes RateSensorRes ...
295     RateRes BodyRateMeasured BodyRateEstimate AngEstFLT AngTriad AttTriad ...
296     BodyRateEstimate BodyRateMeasured InertialMagFieldFLT InertialMagFieldREF ...
297     MagFieldMeasured MagFieldProp MagSensorRes R_ORF_to_b R_ORF_to_i ...
298     R_i_to_b R_i_to_b_4triad RateEstFLT RateRes RateSensorRes Sun_B Sun_I ...
299     Sun_I_measured TriadRes SunSensorRes
```

```
300
301 %% Now construct neural net training vector
302 NNtraining = zeros(21,n);
303 for i=1:n
304     x1 = NeuralNet.Attitude.ATTresREF(i);
305     x2 = NeuralNet.Attitude.RateResREF(i);
306     x3 = NeuralNet.Actuator.WheelSpdRes(i,1);
307     x4 = NeuralNet.Actuator.WheelSpdRes(i,2);
308     x5 = NeuralNet.Actuator.WheelSpdRes(i,3);
309     x6 = NeuralNet.Actuator.WheelVoltageRes(i,1);
310     x7 = NeuralNet.Actuator.WheelVoltageRes(i,2);
311     x8 = NeuralNet.Actuator.WheelVoltageRes(i,3);
312     x9 = NeuralNet.Actuator.MagPolRes(i,1);
313     x10 = NeuralNet.Actuator.MagPolRes(i,2);
314     x11 = NeuralNet.Actuator.MagPolRes(i,3);
315     x12 = NeuralNet.Propagator.JulianDateRes(i);
316     x13 = NeuralNet.Propagator.PositionRes(i);
317     x14 = NeuralNet.Propagator.MagFieldRes(i);
318     x15 = NeuralNet.Propagator.SolarPositionRes(i);
319     x16 = NeuralNet.Sensor.Magnetometer(i);
320     x17 = NeuralNet.Sensor.SunSensor(i);
321     x18 = NeuralNet.Sensor.RateSensor(i);
322     x19 = NeuralNet.Sensor.Triad(i);
323     x20 = NeuralNet.Guidance.ATTCmdResREF(i);
324 %     x21 = NeuralNet.Guidance.AngCmdResREF(i);
325     x22 = NeuralNet.Guidance.RateCmdResREF(i);
326
327     NNtraining(:,i) = ...
328         [x1;x2;x20;x22;x3;x4;x5;x6;x7;x8;x9;x10;x11;x12;x13;x14;x15;...
329         x16;x17;x18;x19];
330 end
331
332 NeuralNet.Training = NNtraining;
333 % NeuralNet.Target = NNtarget';
334
```

```
clearvars x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 ...
    x19 NNtarget NNtraining x20 ATTCmdRes ATTEstFLT ATTRes AngCmdResREF ...
    AngResRef B_BFCS_Unit B_ECI_Unit B_ECI_debug InertialMagFieldTruthFLT ...
    Nadir_ORF R_ORF_to_b_triad R_i_to_ORF SolVec nadir_check sigma ...
    sigma_mag sigma_sun ResidualMagField MagUncertainty MagNoise AngResREF ...
    AttTriadRes Cov P x20 x21 x22 R_i_to_ORF_truth R_i_to_b_truth Radical ...
    i
```

B.2 Generate Target Vector

The script MakeTargetVector.m applies a user-defined threshold value to specified residual quantity. When the threshold is exceeded, the appropriate element of the target vector is changed from zero to one, where the element of the vector is determined by the parameter "FaultNum" (see table A.1).

```
ErrorRes = NeuralNet.Sensor.RateSensor; %Specify residual

n = length(ErrorRes);

NNtarget = zeros(16,n);
sigma = .05/5; %Specify residual threshold

% Use this code for single faults
for i=1:n
    if abs(ErrorRes(i,1))> 5*sigma %&& i>Tfault/.2
        NNtarget(FaultNum(1),i)=1;
    end
    if abs(ErrorRes(i,2))> 5*sigma %&& i>Tfault/.2
        NNtarget(FaultNum(2),i)=1;
    end
    if abs(ErrorRes(i,3))> 5*sigma %&& i>Tfault/.2
        NNtarget(FaultNum(3),i)=1;
    end
end
```

```
20
21 % Use this code for multiple faults
22 % for i=1:n
23 %     if abs(ErrorRes(i))> 5*sigma %&& i>Tfault/.2
24 %         NNtarget(FaultNum,i)=1;
25 %     end
26 % end
27
28  NeuralNet.Target = NNtarget;
29
30 figure;plot(0:.2:def.Time(end),NeuralNet.Target(FaultNum,:))
```

B.3 Determination of Faults

Script DetermineFaults.m loads each of the specified sets of residuals and then loads the specified neural network. For each case, the script CreateFaultVector.m is called, which generates a 1x16 element vector whose elements denote where the fault has occurred. After each of the individual fault vectors is created, script CalcAccuracy.m is called, which calculates the number of positively and falsely identified faults.

```
1 % This script performs the following operations:
2 % Loads the specified neural network
3 % Loads each case
4 % Calls the script CreateFaultVector.m which forms a 1x16 vector,
5 %   corresponding to the number of fault categories and populates it with
6 %   the detected faults
7 % Calls script CalculateAccuracy which determines the number of positively
8 %   identified and falsely identified faults
9
10 addpath(genpath(pwd))
11 clear all
12
13 str1 = 'NeuralNet_case_';
```

```
14 str2 = '.mat';
15
16 load('BR_22neurons_Interleved_1'); %load neural net
17 Thresholds = zeros(16,2);
18 Thresholds(:,1) = .9;
19
20 %% Create Fault Vector
21
22 NumCrossings = 5; %num. of continuous threshold crossings before hard fault
23 SigmaLevel = 5; %STUB, number of std for fault detection
24
25 StartCase = 1; %First case to load
26 NumCases = 112; %Number of case files to load
27 for i=StartCase:NumCases
28     if i<10
29         case_num = strcat('0',num2str(i));
30     else
31         case_num = num2str(i);
32     end
33     CaseName = strcat(str1,case_num,str2);
34     load(CaseName)
35     output = net(NeuralNet.Training(:,300:end-300));
36     FaultVector = ...
37         CreateFaultVector(output,Thresholds,SigmaLevel,NumCrossings);
38     FileName = strcat('Fault_Vector_Case_',case_num);
39     save(FileName,'FaultVector');
40     clearvars def NeuralNet
41 end
42
43 CalcAccuracy
44 save('Accuracy','A')
45
46 clearvars NumPos NumFalsePos NumCrossings PositiveFault SigmaLevel ...
47     Thresholds output i j m n net tr STDS def counter
```

B.4 Create Fault Vector

CreateFaultVector.m is a function which creates a 1x16 element vector corresponding to the 16 defined fault states. The default vector elements are equal to zero. If a fault is detected the element is changed to one. The function has four inputs: output, Thresholds, SigmaLevel, and NumCrossings. Output refers to the calculated output of the neural network; Thresholds is a 16x2 matrix where the first column contains the thresholds for each of the elements of the neural network output, which was set at 0.9 for this thesis. The second column is a switch to tell the thresholding algorithm the number of standard deviations (SigmaLevel) above the threshold to determine a hard fault and is set to zero by default. NumCrossings refers to the number of consecutive threshold crossings to consider before a hard fault is determined (a value of 5 was used for this thesis to correspond to the 5 Hz output of the HARP ADCS).

```
1  % This script creats a 1x16 element vector corresponding to the 16 defined
2  % fault states. The default vector elements are equal to zero. If fault is
3  % detected the element is changed to equal one.
4
5  function FaultVec = CreateFaultVector(output,Thresholds,SigmaLevel,...
6      NumCrossings)
7
8  FaultVec = zeros(1,16); %vector to track where faults are detected
9  counter = 0; %initialize number of continuous crossings
10 [m,n] = size(output);
11
12 for i=1:m
13     for j=1:n
14         if output(i,j) > Thresholds(i,1) + SigmaLevel*Thresholds(i,2)
15             counter = counter+1;
16             if counter >= NumCrossings
17                 FaultVec(i) = 1;
18             end
19         else
```

```
            counter = 0;
        end
    end
end
```

B.5 Calculating Neural Network Accuracy

The accuracy of the neural networks is calculated by script CalcAccuracy.m. It first loads the file CaseFaults.mat, which is a 112x2 element matrix. Column one is numbered 1 through 112. Column two contains numbers between 1 and 16 and denotes which fault is the positive fault for each of the cases. The output of this script is a 112x2 matrix, "A." Column one of the matrix contains the number of positively identified faults (always 1 is positive detection is made). Column contains the number of false identified faults.

```
% This script is used to determine the accuracy of the trained neural nets.
% Accuracy = (# Positive Detections)/(# Positive + # False Detections)
% clear all
load('CaseFaults.mat') %load case numbers with corresponding fault numbers

str1 = 'Fault_Vector_Case_';
str2 = '.mat';

A = zeros(NumCases-StartCase+1,2);
%matrix format of A is [#Positive #FalsePositive], nx2
NumFalsePos = 0; %number of false positive fault detections

for i=StartCase:NumCases
    if i<10
        case_num = strcat('0',num2str(i));
    else
        case_num = num2str(i);
    end
    CaseName = strcat(str1,case_num,str2);
```

```
20      load(CaseName)
21      for j=1:16
22          if FaultVector(j) ~= 0 %is there a fault present?
23              if j == CaseFaults(i,2) %Check which case is Positive Detection
24                  A(i,1) = 1; %Mark as Positive Detection
25              else
26                  NumFalsePos = NumFalsePos+1;
27              end
28              A(i,2) = NumFalsePos; %Mark as False Positive Detection
29          end
30      end
31      NumFalsePos = 0; % Reset counter
32      clearvars FaultVector
33  end
```

B.6 Determine Subsystem Fault Isolation Accuracy

Determining the faults on a subsystem level first requires the existing faults of FaultVector.m to be bundled into one of three categories, one for each of the defined ADCS subsystems. Script CalcSubSysAccuracy.m loads the specified neural network and each of the fault vectors associated with that network. File SubSysFaults.mat is loaded, which describes which fault belongs to which subsystem. The script then cycles through each of the fault vectors and determines the number of positively and falsely isolated faults on a subsystem level by calling script CreateSubSysFaultVector.m. This scripts creates a 1x3 vector called SubSystemFaultVector whose purpose is the same as the 1x16 FaultVector. For each of the subsystem fault vectors a matrix, "B" is populated. The structure of "B" is exactly the same as matrix "A" except the positively and falsely identified faults refer to the subsystem level.

```
1  % This script is used to determine the accuracy of the trained neural nets.
2  % Accuracy = (# Positive Detections)/(# Positive + # False Detections)
3  clear all
```

```
4
5  load('SubSysFaults.mat') %load case numbers with corresponding fault numbers
6  load('BR_22neurons_Interleved_1'); %load neural net
7
8  str1 = 'Fault_Vector_Case_';
9  str2 = '.mat';
10
11 NumCases = 112;
12 StartCase = 1;
13
14 B = zeros(NumCases-StartCase+1,2);
15 %matrix format of B is [#Positive #FalsePositive], nx2
16 NumFalsePos = 0; %number of false positive fault detections
17
18 for i=StartCase:NumCases
19     if i<10
20         case_num = strcat('0',num2str(i));
21     else
22         case_num = num2str(i);
23     end
24     CaseName = strcat(str1,case_num,str2);
25     load(CaseName)
26     SubSystemFaultVector = CreateSubSysFaultVector(FaultVector);
27     FileName = strcat('SubSys_Fault_Vector_Case_',case_num);
28     save(FileName,'SubSystemFaultVector');
29     for j=1:3
30         if SubSystemFaultVector(j) ~= 0 %is there a fault present?
31             if j == SubSysFaults(i,2) %Check which is Positive Detection
32                 B(i,1) = 1; %Mark as Positive Detection
33             else
34                 NumFalsePos = NumFalsePos+1;
35             end
36             B(i,2) = NumFalsePos; %Mark as False Positive Detection
37         end
38     end
```

```
39      NumFalsePos = 0; % Reset counter
40      clearvars FaultVector SubSystemFaultVector
41  end
42
43  save('SubSysAccuracy','B')
```

B.7 Create Subsystem Fault Vector

CreateSubSysFaultVector.m bundles the component-level faults of FaultVector.mat into one of three categories that correspond to either the attitude control, guidance, or attitude determination subsystems.

```
1  % This script creats a 1x3 element vector corresponding to the 3 defined
2  % subsystems. The default vector elements are equal to zero. If fault is
3  % detected the element is changed to equal one.
4  function SubSystemFaultVector = CreateSubSysFaultVector(FaultVector)
5
6  SubSystemFaultVector = zeros(1,3);
7      for j=1:16
8          if FaultVector(j) ~= 0 %is there a fault present?
9              if j<=9
10                 SubSystemFaultVector(1) = 1; %AC Fault
11             elseif  j<=12
12                 SubSystemFaultVector(2) = 1; %Guidance Fault
13             elseif  j<=16
14                 SubSystemFaultVector(3) = 1; %AD Fault
15             end
16         end
17     end
18 end
```