Model-based Fault Detection for Low-cost UAV Actuators

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

INCHARA LAKSHMINARAYAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

PETER SEILER, ADVISOR

SEPTEMBER, 2016

Acknowledgements

First and foremost I am grateful to my advisor Prof. Peter Seiler for patiently guiding and supporting me through the project. He has been a great mentor throughout my graduate education and a huge contributor to my learning experience. I also thank Prof. Demoz Gebre-Egziabher for his motivation and support.

I am thankful for everybody involved with this project. I specially thank the post-doctoral researcher Dr. Daniel Ossmann for all the insightful discussions. I also thank Johanna Balas , Apurva Badithela and The Director of the UAV laboratory Brian Taylor for helping with the experimental setup.

I thank the project sponsors. This work was supported by a MnDrive Exploratory Grant entitled "Smart Actuators for Preventative Maintenance of Small Uninhabited Aircraft". This work was also supported by the NASA Langley NRA Cooperative Agreement under Grant no. NNX12AM55A entitled "Analytical Validation Tools for Safety Critical Systems Under Loss-of-Control Conditions".

Finally I thank my family and friends. Specially, I am grateful to my mother Indira Lakshminarayan for always supporting my academic endeavours and inspiring me to push higher, Dr.Niteesh Bharadwaj for his support towards my career and encouraging me to pursue graduate studies.

Dedication

To my mother.

Abstract

The focus of this thesis is on the use of analytical redundancy to improve the reliability of low-cost unmanned aerial vehicles (UAVs). Specifically, a model-based fault detection algorithm is designed and tested for one critical UAV component: a servo-actuator. As the name suggests, a key requirement to developing this type of fault detection algorithm on actuators is the availability of an accurate actuator model. This is accomplished by developing a dedicated Arduino based experimental test-bed to analyze servos. Using input-output data from these experiments, a second/third-order dynamic model is identified for healthy actuators using system identification methods in MATLAB software. Using the identified model, a fault detection filter is designed based on polynomial basis vectors to generate a residual proportional to fault. The performance of the fault detection algorithm is experimentally tested on both healthy and faulty actuators and the detection thresholds are set. Finally, the actuator model and the fault detection filter are validated using actuator commands from recent flight tests conducted at the University of Minnesota.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unmanned Aircraft Systems (UAS) are evolving at a tremendous pace to become an integral part of everyday operations. Commonly known as drones, these aircrafts have experienced a worldwide boom in both the military and commercial sector. These vehicles serve a variety of purposes such as search and rescue missions, precision agriculture, monitoring and surveys, to name a few. According to the Association for Unmanned Vehicle Systems International (AUVSI), UAS will create \$82 billion in economic impact over the 10-year span from 2015 to 2025. Despite their emerging role, safe integration of Unmanned Aerial Vehicles (UAV) in the National Airspace System (NAS) is a challenge for both the Federal Aviation administration (FAA) and the aviation community.

The capability of UAV to "sense and avoid" is one major, as-yet undeveloped technology which is a key factor to full NAS integration. UAVs are not developed to comply with existing airworthiness standards and fail to exhibit technical capabilities analogous to manned aircraft. The FAA's roadmap [1] acknowledges these shortcomings and outlines the actions and considerations needed to enable UAS integration into the NAS. Large or commercial aircrafts are required to meet two main reliability requirements: 1) no more than one catastrophic failure per 10^9 flight hours and 2) no single point of failure [2, 3]. Commercial aircraft manufacturers achieve these reliability requirements primarily through the use of hardware redundancy. In this approach, the main Guidance, Navigation and Control (GNC) system is made of several parallel, functionally identical and independant GNC systems. If there is a failure in one of the component then another identical component takes over. This tried and tested approach cannot be used in small UAVs due to severe size, weight, cost and power constraints. As a result, civil

UAS have reliabilities that are several orders of magnitude below the 10^{-9} failures per flight standards. For instance, the UAV Research Group at the University of Minnesota (UMN) [4] operates an *Ultra Stick 120* aircraft with single-string, off-the-shelf components. A theoretical estimation of the failure rate using a comprehensive fault tree analysis yielded 2.2×10^{-2} failures-per-flight-hour for this aircraft [5]. Note that this is only a theoretical estimate and no aircraft has been lost to date.

An alternate approach is to introduce analytical redundancy in UAV. In analytical redundant schemes, a residual signal is generated by a fault detection filter. This residual will be zero during normal operation of the system for all control, disturbance and noise inputs. In the presence of a fault, the residual generates amplitudes proportional to the fault. Various approaches for solving the fault detection and isolation filter design problem are available in the literature [6–8]. This design makes use of mathematical models of the monitored process and hence is often referred to as the ‘model-based’ approach to fault diagnosis. Although this method is widely used to increase safety and reliability requirements in small unmanned aircrafts, a limited degree of analytical redundancy has been applied to some commercial aircrafts such as the Airbus A380 [9].

Analytical redundancy can be applied for small UAVs either at the system or component level. The system level approach, e.g. [10–14], combines models of the aircraft dynamics with inertial measurements. This approach is challenging as it requires accurate models of the flight dynamics which may not be available for low-cost UAVs. Alternatively, fault detection algorithms can be developed at the sensor or actuator level. For example, actuator faults can be detected by using fault detection filters which propagate only the actuator input and output signals. This approach has recently been studied for hydraulic actuators on larger commercial aircrafts [9, 15–17].

The subsequent chapters in this thesis are focused on a component level fault detection algorithm for one specific UAV component which is a servo motor. The significance of designing a dedicated fault detection filter for servos is detailed in the next chapter. This is followed by the description of a dedicated laboratory experimental test-bed for UAV actuators developed by the UMN UAV lab. Using this, a servo model has been identified and a fault detection filter is designed. The design is experimentally validated by performing tests on healthy and broken actuators using data from flight tests performed at the UMN UAV lab.

Chapter 2

Problem Formulation

A complex mechanical system such as an unmanned aircraft can fail in many ways. In case of a failure of one/more of the hardware components in an aircraft, it is possible to still land the aircraft safely by switching the control law. This has been demonstrated in [18] where advanced control techniques were used to safely land a severely degraded aircraft However, this requires the fault to be detected so that the controller can reconfigure. Fault detection and identification is the precursor to control law reconfiguration.

[5] and [19] present the failure modes and effects analysis (FMEA) of the Ultrastick 120 aircraft which is a model aircraft used at the University of Minnesota UAV lab. This section provides a brief description of this aircraft and presents an overview of the fault detection approach developed in this report to address some of the problems recognized in [5] and [19].

2.1 Flight Test Equipment

The experimental vehicle is called *FASER*, and is a commercial, off-the-shelf, radio-controlled unmanned aircraft with the Ultra Stick 120 airframe, shown in Figure 2.1. The FASER platform was obtained by University of Minnesota from NASA Langley Research Center. *FASER* has a wingspan of 1.92m and a mass of about 7.4kg. The aircraft has a flight endurance of 30 minutes and a cruise speed of 23m/sec. Additional details can be found in [20, 21]. NASA Langley identified the aerodynamic coefficients for *FASER* from wind tunnel experiments [22, 23]. This aerodynamic data provides the foundation for a high fidelity, six degree-of-freedom nonlinear simulation model. This simulation model and all flight data are publicly available [4].

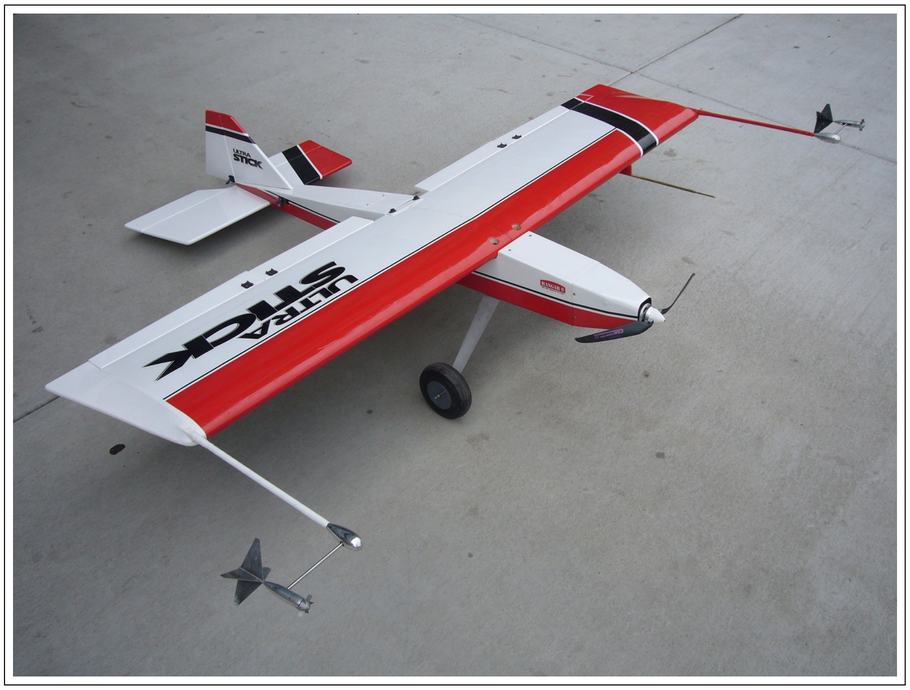

Figure 2.1: Ultrastick 120 aircraft

FASER is retrofitted with UMN avionics hardware and software for real-time control, guidance, navigation, and fault detection. The avionics include a flight computer, telemetry radio and sensors. The aerodynamic control surfaces (flaps, ailerons, elevator and rudder) are actuated by their own servo motor. The system has a throttle input and 4 servo inputs to deflect the elevator, rudder, symmetric left/right aileron, and left/right flap.

Results from FMEA and fault tree analysis (FTA) on the Ultrastick 120 aircraft are presented in [5] and summarized in Table 2.1. The aircraft uses a single-string architecture and the FTA predicted one failure per 50 flight hours. The components that were identified to have failures with a high risk of catastrophic damage are: the fail-safe switch, R/C receiver, avionics battery, battery eliminator circuit (BEC), the rudder and elevator servos. A fault in these components was classified as 'Type 1A' and these faults led to an uncontrolled emergency landing with a high risk of catastrophic damage. Among these, the effector servos were found to have the highest failure rate. 'Type 1B' faults led to a controlled emergency landing with low risk of catastrophic damage. The electronic speed control (ESC), motor battery, motor, aileron and flap servos and the propeller exhibited this level of failure. The last category of faults were

'Type 2' faults which led to a mission critical failure and subsequent loss of data. This type of failure was exhibited by the GPS antenna, flight computer, GPS receiver and IMU.

Failure type	Consequence	Failure rate (failures/100 hours)	Component with highest failure rate
Type 1A	Uncontrolled emergency landing	2.17	Elevator/Rudder servos
Type 1B	Controlled emergency landing	2.14	Propeller
Type 2	Mission critical failure	2.32	IMU

Table 2.1: Reliability analysis of the Ultrastick 120 UAV components

This type of reliability analysis gives us an idea about the most critical components of a low-cost UAV. Specifically in the area of fault tolerant control, knowledge of the location and type of actuator failure is vital for reconfiguration of the guidance and control laws of the aircraft to prevent a loss of vehicle.

2.2 UAV Actuator Failure Modes

Servos are used to actuate the control surfaces in an aircraft and they are one of the most critical components of a UAV. Figure 2.2 represents the outside and inside of one such servo commonly used in UAVs today. These servos are an assembly of four parts: a DC motor, gear train, control circuit and a position sensor (usually a potentiometer). The servo shown in the figure is small, lightweight (26.93 g) and cheap ($\sim$\$20) with a high speed (0.14 sec/60^o at 4.8V) and high torque (3.9 kg/cm at 4.8V).

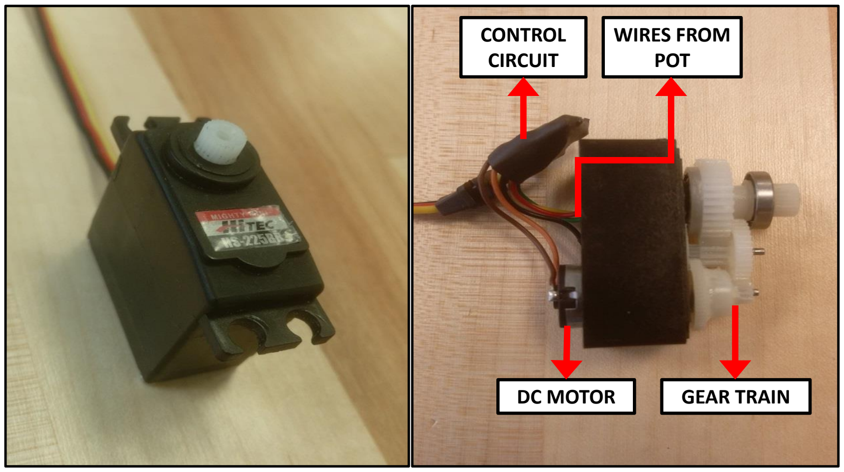

Figure 2.2: The outside and inside of HITEC HS-225BB servo

Regardless of the cost or make, these servos tend to degrade over time and usage. Servo failures can be categorized based on the cause and effect. Based on the cause, there are six principal modes of effector servo failures that can occur in a UAV during flight. These faults are recognized as: Bias, Stuck surface, Hardover, Floating surface, Oscillatory mode, Increased deadband/stiction [19]. Classifying by effects according to the NASA standards for flight vehicle FMEA, the fault modes can be catastrophic, critical, significant or minor.

In this thesis we deal with the following two modes of actuator failures encountered in *FASER* flight tests:

- *Stuck Fault:* This is a catastrophic fault type. It occurs due to a damage in the servo drive shaft, linkage or due to an unbalanced surface. This may lead to a loss of motion, loss of control, and in severe cases, loss of vehicle. The damage in the servo driveshaft can be detected by physical inspection of the servo by rotating the shaft. The exact type of the fault can be confirmed by analyzing the output from the actuator.

- *Increased Deadband/Stiction:* This is a critical fault known to occur due to slippage of gears, or damaged servo drive shaft. This may lead to a loss of motion. Upon physical inspection of the servo, noticeable clicking sounds can be heard when the servo shaft is rotated.

2.3 Fault Detection Architecture

As stated before, analytical redundancy in control can be applied for small UAVs either at the system level or component level. Here we use a model based fault detection scheme to detect actuator faults as represented by Figure 2.3.

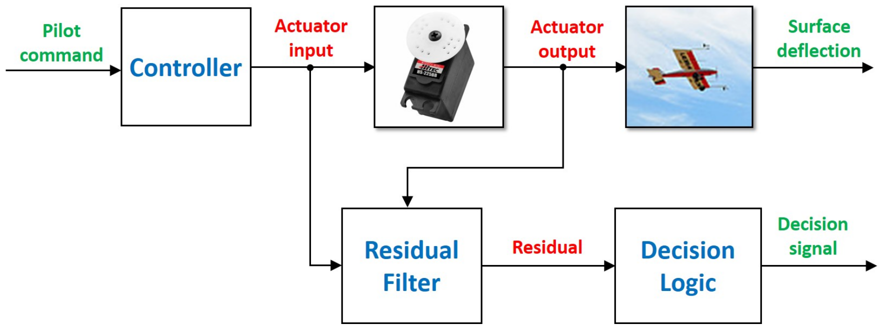

Figure 2.3: Fault monitoring setup

The model based fault diagnosis has been defined in [8] as ”the determination of faults of a system from the comparison of available system measurements with a priori information represented by the system’s mathematical model, through generation of residual quantities and their analysis”. Hence the first step in actuator fault detection is to identify a robust actuator model to provide this a priori information for the fault detection filter. For this purpose, a dedicated bench top experimental testbed for actuators has been developed at the UAV lab of the University of Minnesota. This is described in Chapter 3. The setup of the *FASER* system allows us to record the data sent from the controller to the different actuators and also collect the position outputs from the aircraft sensor. We have used this data in the simulation to excite the actuator and test the model based fault detection filter. The residual filter generates signals proportional to the level of fault which is used by the decision logic to raise a fault flag. This is detailed in Chapter 4. Finally, the conclusion and scope for future work is presented in Chapter 5.

Chapter 3

Actuator Modeling

Actuators used in UAVs such as the Ultrastick 120 are radio control hobby servos. Figure 3.1 represents the feedback mechanism in servo motors.

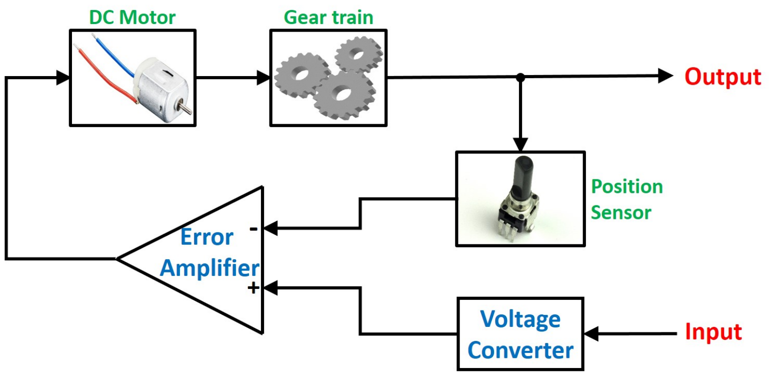

Figure 3.1: Block diagram of a closed-loop servo mechanism

The first step is to obtain a model of the actuator being used on the UAV. There are multiple approaches to obtain a model for the servo actuator. Theoretically, a mathematical model can be derived by coupling the linear dynamics of the DC motor with the nonlinear dynamics of the gear train. But it is not easy to determine the motor speed and torque constants for low cost UAV actuators. Instead, if we have input-output data from an actuator, we can do black box system identification. The latter method is chosen for our purposes.

In this section we describe the experimental setup developed to obtain the input-output data and the system identification procedure to obtain a model for a healthy actuator. The setup is such that the input-output data can be obtained quickly with less effort. The data can then be processed in MATLAB to generate the transfer function of the actuator. Results of the model validation are presented in both time and frequency domains.

3.1 Laboratory Experimental Setup

Figure 3.2 depicts the laboratory experimental setup used for actuator analysis. The objective of the setup is to perform offline analysis of the servos used in flight for modeling and fault diagnosis. The heart of the setup is an Arduino based microcontroller used to control the RC servo. The board used is a Teensy 3.1 microcontroller which costs \$19.8 [24] and uses a 72MHz Cortex M4 processor. It has a 32 bit ARM, 64KB RAM and 2KB EEPROM, making it a good choice to handle large amount of input and output data. Arduino 1.6.1 software is used to program the microcontroller in the Arduino programming language.

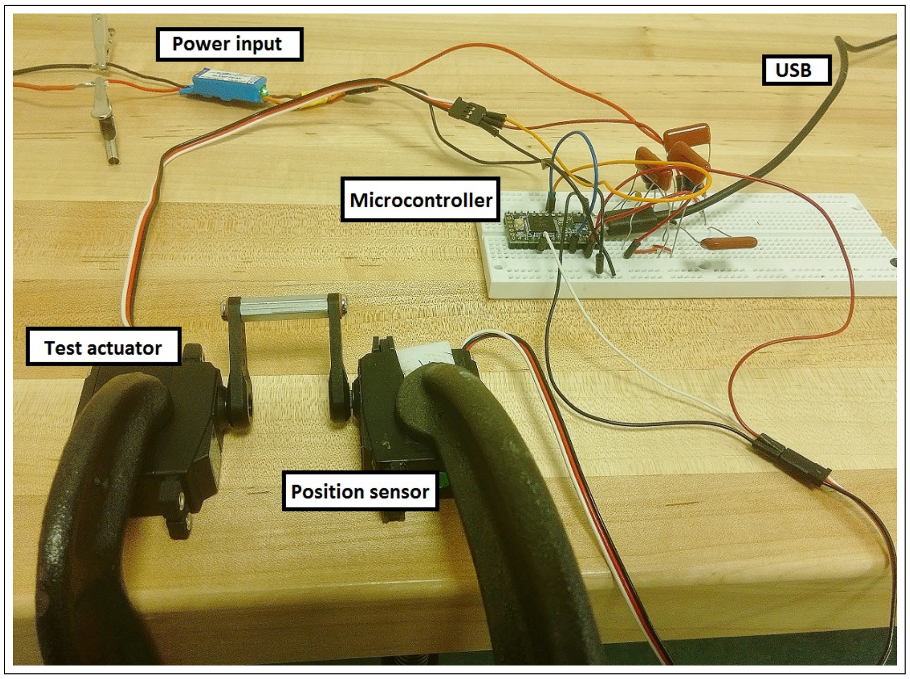

Figure 3.2: Experimental setup

Servo motors have three wires: power (red wire), ground (black/brown wire) and signal (yellow/orange/white wire). The power to the test servo is supplied by a regulated DC power supply along with a Castle creations battery eliminator circuit (CC BEC) as shown in Figure 3.2. Most RC servos require a DC input of 4.8V or 6V. The current drawn by these servos is in tens of mA in idle state or a few hundreds of mA when operating on no load. A common ground is established for the servo, the microcontroller and also the position sensor. The third (signal) wire of the servo is connected to any one of the pulse width modulation (PWM) pins of the Teensy board. This establishes the input circuitry for our setup.

The position output of the test servo is measured using a second servo modified to operate as just a potentiometer. This is done by removing the gearset from a normal servo and wiring the power and ground. The power to the potentiometer is supplied from the 3.3V pin of the Teensy 3.1. The potentiometer measurements are calibrated to remove any DC offsets.

3.2 Data collection

In order to analyze the frequency domain and time domain responses of the servo, three tests were conducted on the setup in Figure 3.2. This section gives a description of these tests.

3.2.1 Calibration test

Servo motors are controlled by PWM pulses. The microcontroller commands the servo to go to a specific position by varying the width of these pulses. Hence the input commands sent by the Teensy is in units of time in microseconds. A servo has a predefined neutral point which is the position where the servo has exactly the same amount of potential rotation in the clockwise direction as it does in the counter clockwise direction. This value is usually given as part of the manufacturer specifications and for most servos it is around 1500 μ s. If the servo can perform a full sweep of 180° then the neutral point corresponds to a position of 90°. But the output from the potentiometer is recorded by the Teensy in units of bits, the range of which depends on the analog read resolution set by the programmer. Hence we need to calibrate the readings from the potentiometer and find a mapping between the output in bits to shaft position in degrees.

Two methods were used to calibrate the potentiometer. The first method applies to servos that exhibit the full range of sweep from 0° to 180°. In this test, the servo shaft was manually set to the extreme positions and the POT output was recorded. This gave us the bit to degree mapping. For servos that don't display a 180° sweep, a bit to microsecond mapping was found by commanding the servo to two specific shaft positions.

3.2.2 Step response

The second set of tests involved recording input-output data for step commands. This test is useful in determining the time domain specifications, mainly the the rate limits of the servo. The servo was commanded from a position corresponding to 1000 μs to a position corresponding to 1800 μs. The software ran in real-time at 10kHz. The rate limit of a servo is the slope of the output in the linear region. Figure 3.3 shows the input-output plot for one of HITEC's high speed digital servo, HITEC 5625MG for an operating voltage of 6V. The slope of the output in the linear region is 456 deg/sec which is the rate limit for this servo.

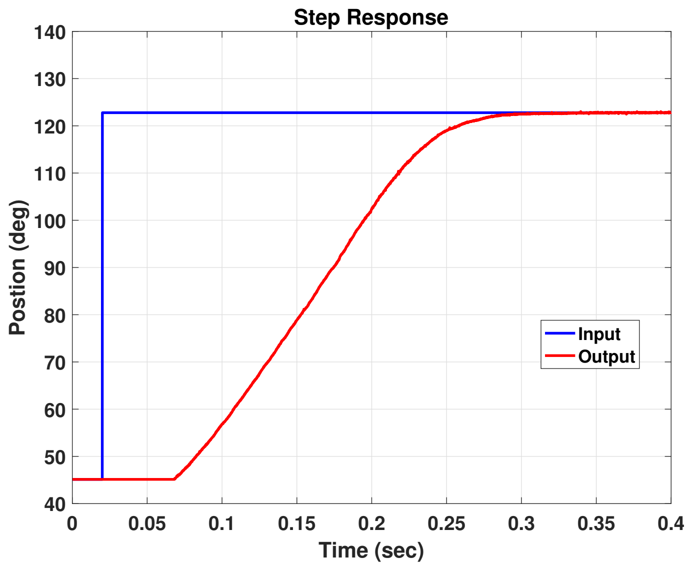

Figure 3.3: Step response of HITEC 5625MG digital servo

3.2.3 Chirp response

This test is used to determine the frequency domain specifications of the servo. The input-output data can be used to perform system identification to determine the transfer function and the bandwidth. The servo is commanded a chirp signal with a frequency sweep from 1Hz to 25Hz and amplitudes of $\pm$4deg over a period of 120 seconds. The software is run at 100Hz. Figure 3.4 shows the input-output data for frequency sweep commands sent to HITEC 5625MG.

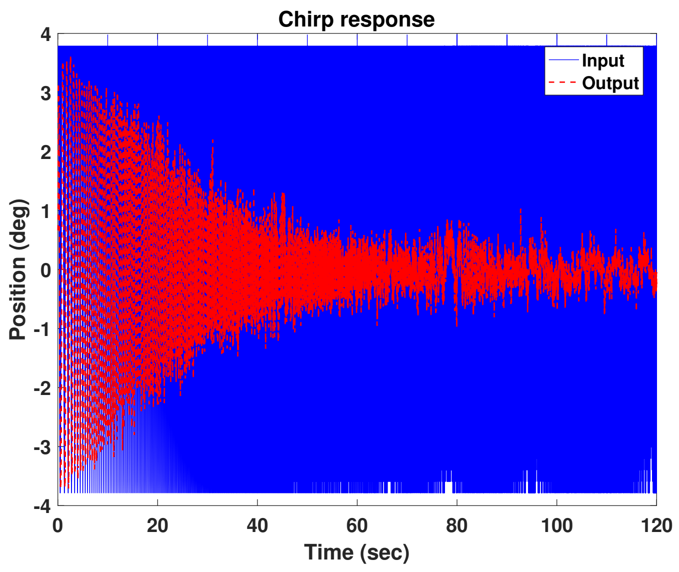

Figure 3.4: Chirp response of HITEC 5625MG digital servo

It is also necessary to perform all the above three tests at an optimal PWM update frequency. This parameter affects the behavior of the servo, the quality of the logged data and the actuator model itself. Most analog and digital servos have a maximum safe update rate of 50Hz. A 50Hz update rate corresponds to seeing a pulse every 20ms. If the PWM update frequency is too high then the servo can malfunction and miss incoming data. Some Futaba digital servos are known to have much higher limits of 333Hz. Such servos also have greater shaft speeds. In our experiments with PWM frequencies near the limits, we found the servo to exhibit uncontrolled motion in the form of sudden jerks and a high current consumption with overheating. However, a high PWM update rate within the safety limit yields a higher servo bandwidth, tighter deadband, more holding power and less time to develop full torque.

The next section describes the process of identifying a system model using input-output data from chirp tests.

3.3 System Identification

Frequency domain identification techniques are used in MATLAB to identify the actuator model. Fast fourier transformations of the chirp input-output data are computed and used to obtain the empirical transfer function and frequency response $G_{raw}(j\omega)$ of the raw data. An additional angle offset is added to the phase to account for factors of 360° that arise due to noise/low coherence at low frequencies. The MATLAB `spa` function is used to obtain the smoothed frequency response $G_{sm}(j\omega)$ of the raw data. This function estimates a frequency response with fixed frequency resolution using spectral analysis. An optimal width of the Hanning window is chosen to minimize bias while still giving a smooth estimate.

In order to account for variations in the response from different actuators, even of the same make and model, due to factors such as manufacturing defects and noise levels, the above experiment and simulation is repeated on multiple healthy servos. An average of the magnitude and the phase response is calculated and a nominal frequency response $G_{avg}(j\omega)$ of the smoothed data is used for model identification.

The nominal model consists of the average response on a frequency grid. The next step is to fit this data with a transfer function. An optimal transfer function fit $G_{fit}(j\omega)$ is obtained using the `fitmagfrd` function of the robust control toolbox. `fitmagfrd` fits frequency response magnitude data with minimum-phase state-space model using log-Chebychev magnitude design. This provided better fits for data just beyond the bandwidth. Alternative functions that use least squares, such as `fitfrd`, are very accurate at low frequencies but at the expense of poor fits just above the bandwidth. Since `fitmagfrd` assumes a stable system with minimal phase, a time delay is manually set in order to match the phase of the transfer function with that of the smoothed frequency response.

For further clarity, analysis from system identification of HITEC HS-225BB servo is presented here. Figure 3.5 represents a linear third order model fit using experimental data from three healthy HITEC HS-225BB servos. The blue line gives the averaged empirical frequency response obtained from raw data $G_{avg,raw}$. The red line is the smoothed response $G_{avg,sm}$ obtained by using `spa` function. Finally, a servo model is obtained by fitting a third order model using `fitmagfrd` function. A delay of 4 milliseconds is added for this servo model.

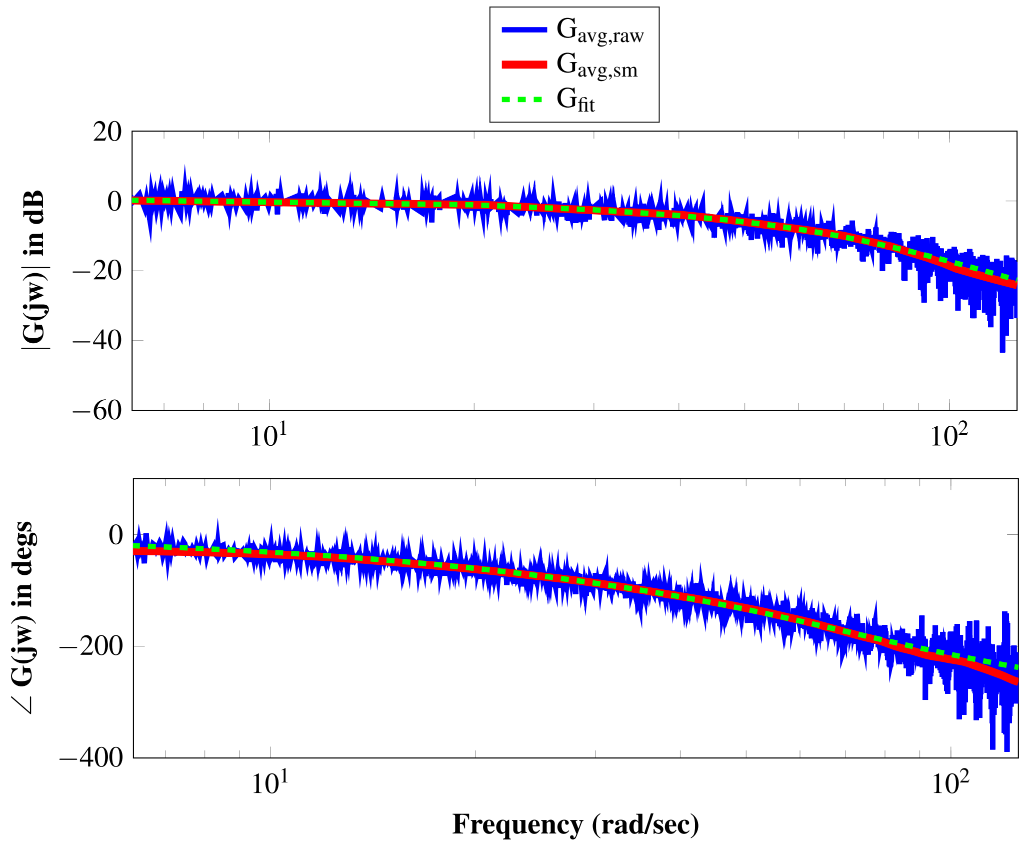

Figure 3.5: HITEC HS-225BB: Frequency response of model fit computed from averaging the frequency response of three healthy servos.

Equation (3.1) describes the transfer function fit for HITEC HS-225BB actuator:

$$G_{fit}(s) = e^{-0.004s}\frac{122300}{(s+24.84)(s^2+70.04s+4921)} \tag{3.1}$$

3.4 Model Validation

This subsection presents time and frequency domain validation results to justify the identification of the model for HITEC HS-225BB actuator. Figure 3.6 illustrates the frequency response of the transfer function fit against the smoothed frequency response of all the tested servos. The dotted lines in Figure 3.6 show the smoothed data for three servos. i in $G_{sm,i}$ denotes the individual servos (here i∈[1,3]). The bandwidth of HITEC HS-225BB actuator is determined to be 4.5Hz.

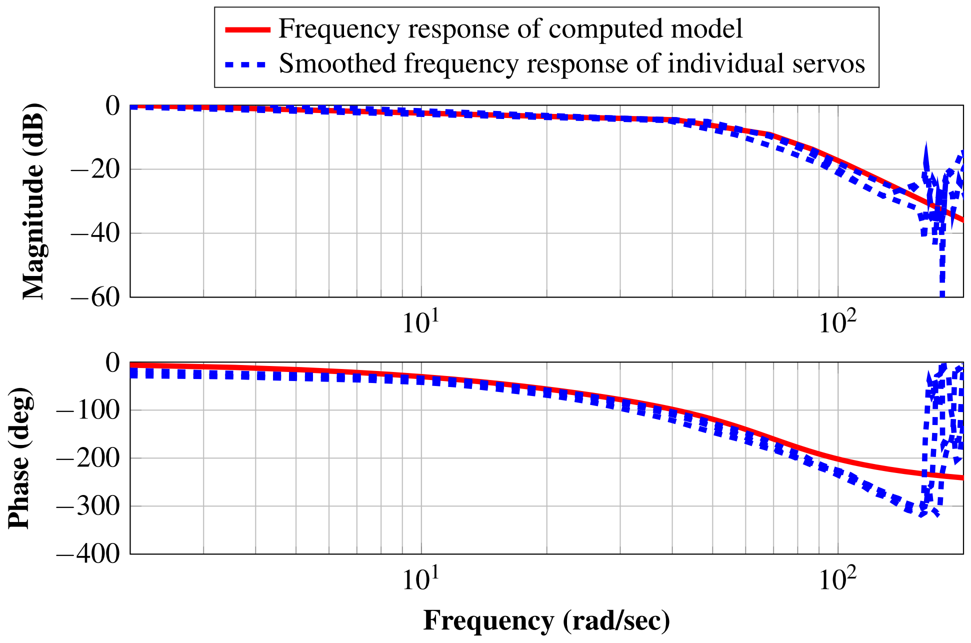

Figure 3.6: HITEC HS-225BB: Comparison of smoothed frequency response of individual servos ($G_{sm,i}(j\omega)$) and the model fit $G_{fit}(j\omega)$.

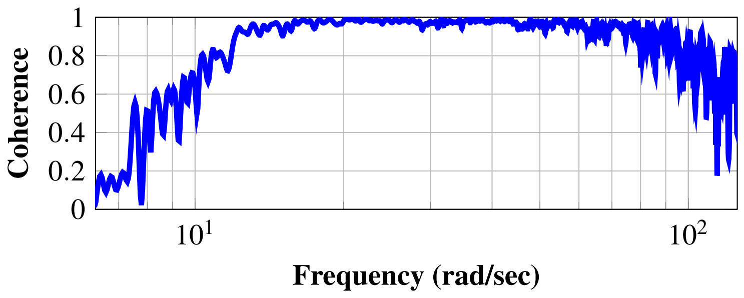

Figure 3.7: HITEC HS-225BB: The coherence plot between raw experimental data and model fit.

Figure 3.8 depicts the normalized magnitude error given by,

$$e(j\omega) = \frac{|G_{sm,i}(j\omega) - G_{fit}(j\omega)|}{G_{fit}(j\omega)} \tag{3.2}$$

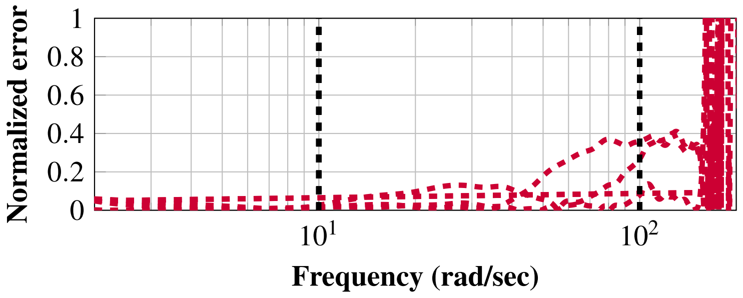

Figure 3.8: HITEC HS-225BB: Normalized error of the fitted model with respect to the individual servos.

Since the servo excitation starts at 6 rad/sec, the coherence between input and output data is found to be good (> 0.8) from frequencies starting at 10 rad/sec as shown in Figure 3.7. In general, the coherence describes how well the input/output data is fit with a linear model as a function of frequency. The coherence of the input and output data gets unsatisfactory at frequencies greater than 100 rad/sec for this servo as high frequency noise and rate limits influence the output. This is the range in which the model adequately reflects the measured data, which is also marked in Figure 3.8.

At frequencies greater than 150 rad/sec, there is a surge in $e(j\omega)$ indicating noise influences at high frequency. The window of frequencies from 10-100 rad/sec depicts the region of best coherence between input and output. The normalized error for the different servos in this window varies from 0.01-0.37.

Figure 3.9 represents time domain validation results performed using flight test data. Actuator commands to the right aileron of the *FASER* aircraft are logged during a test flight. These commands are used as an input to the servo in the bench-top experiment at a sampling rate of 100Hz. The blue line in Figure 3.9 shows the logged command. The output obtained from the actuator on the bench-top experimental setup for the given flight command input is denoted in green. The simulated output obtained from the actuator model identified in Equation (3.1) for the same command input is denoted in red. The simulated output replicates the actual response with an adequate accuracy. The standard deviation of the error between the simulated and true actuator output in Figure 3.9 is 0.2^o . Note that the actuator in the bench-top experiment is unloaded. This is a significant difference from the flight tests where the actuator experiences the effects of the aerodynamic loads on the controllable surfaces. A modification of the experimental setup to replicate the aerodynamic loads experienced by the UAV would be beneficial in developing more accurate models and simulations.

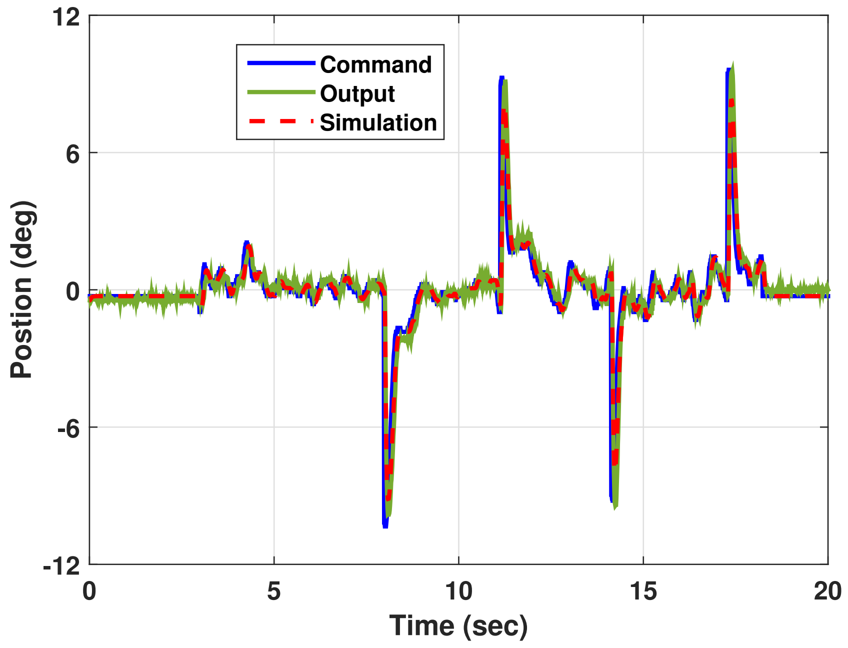

Figure 3.9: Servo true output and simulated output for *FASER* flight commands

Once the actuator model has been verified, the simulated output can be taken as the true/ expected output of the actuator. Hence if an actuator is broken, then the experimental output (from flight tests/ bench top experiments) will reflect the fault and the simulated output becomes the reference/true output.

Figure 3.10 in the next page summarizes the analysis and results of system identification performed on several low-cost servos used in UAVs. The chirp signal data used to identify these actuator models vary from a frequency of 1-25 Hz which correspond to a maximum of 157 rad/sec. Hence the bandwidth of the model fit is within this chirp frequency. We can also notice from the figure that some of the more expensive servos can be approximated to a first-order model if desired. Since our application is for fault detection, using a second order model over a first order fit helps us to reduce modeling errors. Cheap servos such as the HITEC HS-225BB and Futaba S3151 have poles that are oscillatory in nature. This is due to poor-grade gears in these servos. Plastic gears such as those used in such cheap servos tend to have loose gear coupling and hence exhibit an oscillatory response. On the other hand, the HITEC 9360TH is a high speed titanium gear servo known to be one of the robust servos available in the market. Also, as stated earlier in this thesis, the Futaba servos such as the S9254 and S9256 models have high PWM update rate limits of upto 333Hz which can also be observed in the high rate limits. The safe PWM frequency for the HITEC 9360TH and HS-225BB models is only 50Hz. The time delay represented in the figure is the delay that was recognized while performing system identification. This delay corresponds to delay in servo response due to electronics of the setup which includes the internals of the servo and the Teensyduino board.

SERVO TYPE	PRICE	TRANSFER FUNCTION	BANDWIDTH (RAD/SEC)	POLES	RATE LIMIT (DEG/SEC)	TIME DELAY (SEC)
Futaba S9254	\$120	$\frac{6.475\text{x}10^5}{s^2 + 5202s + 6.475\text{x}10^5}$	127.2	127.6, 5074	1000	0.01
Futaba S9256	\$60	$\frac{5.165\text{x}10^4}{s^2 + 579.3s + 5.165\text{x}10^4}$	104.5	110.1, 469.2	1263	0.01
Futaba S3151	\$30	$\frac{654.1}{s^2 + 17.27s + 654.1}$	5.8	-8.63 ±24.74i	265	0.01
HITEC 9360TH	\$180	$\frac{1.091\text{x}10^5}{s^2 + 3455s + 1.091\text{x}10^5}$	31.73	31.87, 3423	605	0.015
HITEC HS-225BB	\$18	$\frac{1.856\text{x}10^5}{s^3 + 135.8s^2 + 9253s + 1.856\text{x}10^5}$	30.8	-30.85, -52.4 ±57.11i	370	0.004

Figure 3.10: Actuator models identified for the different servos used in UMN UAV laboratory

Chapter 4

Fault Detection

This section presents the methodology for implementing a residual filter using the model from Section 3, for the fault diagnosis approach to detect actuator faults. The model-based fault diagnosis is defined in [8] as “the determination of faults of a system from the comparison of available system measurements with a priori information represented by the system’s mathematical model, through generation of residual quantities and their analysis”. The actuator model identified in Chapter 3 provides this a priori information, while the input and the output signals are the available system measurements. The fault detection filter is based on this derived model and uses the system measurements as inputs to generate a residual which is only prone to faults.

4.1 Fault Detector Design

The fault detection filter (diagnostic observer) design problem for the servo is to generate a filter which:

(a) decouples the input from the output

(b) couples the fault to the residual and

(c) is stable and proper.

Various approaches to solve this problem based on parity space calculation, nullspace calculation or observer based approaches are available in literature. The contribution [25] explains in detail, that if the design problem is directly solvable, all of these approaches are able to generate the same fault detection filter. The design problem is directly solvable as there are no unknown inputs that need to be decoupled from the system.

The approach used in this paper to solve the residual filter design problem for the linear actuator model is based on simple nullspace computations as presented in [26]. This approach only involves basic algebraic operations and also provides the designer with the freedom to directly select the poles of the filter.

Modeling the actuator fault as an additive input, the input-output form is given by

$$\mathbf{y}(s) = G_{fit}(s)(\mathbf{u}(s) + \mathbf{f}(s)) \tag{4.1}$$

where $\mathbf{y}(s)$, $\mathbf{u}(s)$ and $\mathbf{f}(s)$ are the Laplace-transformed quantities of the servo position $y(t)$, the commanded input $u(t)$ and the fault input $f(t)$, respectively. To solve the fault detection problem for the system in Equation (4.1) a residual filter of the form

$$\mathbf{r}(s) = Q(s) \begin{bmatrix} \mathbf{y}(s) \\ \mathbf{u}(s) \end{bmatrix} \tag{4.2}$$

shall be generated, which uses the available command input signal u and the measured servo position y of the actuator to generate a residual r. In Equation (4.2), $\mathbf{r}(s)$ is the Laplace-transformed quantity of the residual signal $r(t)$. The idea of the nullspace methodology becomes clear when inserting Equation (4.1) into the residual filter of Equation (4.2), resulting in

$$\mathbf{r}(s) = Q(s) \begin{bmatrix} G_{fit}(s) \\ 1 \end{bmatrix} \mathbf{u}(s) + Q(s) \begin{bmatrix} G_{fit}(s) \\ 0 \end{bmatrix} \mathbf{f}(s) \tag{4.3}$$

The residual r shall be zero in any fault free situation and non-zero if a fault occurs. Also, the residual shall be zero in case of no fault ($f = 0$) only if it is decoupled from the input u. Thus, the residual filter $Q(s)$ must guarantee

$$Q(s) \begin{bmatrix} G_{fit}(s) \\ 1 \end{bmatrix} = 0 \tag{4.4}$$

implying that $Q(s)$ belongs to the left nullspace of $G_{fit}(s)$. Note that the most intuitive solution of the filter problem considering the design constraints (a)-(c) for the actuator dynamics, is the filter

$$Q(s) = \begin{bmatrix} 1 & -G_{fit}(s) \end{bmatrix} \tag{4.5}$$

This filter generates the residual as a difference of the measured servo position y and its estimate $\hat{y} = G_{fit}u$. Equation (4.6) is obtained by substituting Equation (4.5) in Equation (4.3). As desired, the influence of the input u gets decoupled from the residual r leaving only f

$$\mathbf{r}(s) = G_{fit}(s)\mathbf{f}(s) \tag{4.6}$$

allowing the detection of the fault. The drawback of this result is that the fault-to-residual transfer behavior is equal to the underlying system behavior $G_{fit}(s)$. This might be undesirable sometimes, such as when the designer wishes to decrease the detection time by making the fault-to-residual transfer behavior faster. Note that any unmodelled servo dynamics or noise in the output will be transferred to the residual through the transfer function. Thus, to filter out these effects it could also make sense to decrease the dynamics. At this point, instead of directly changing the filter we present a general design approach which can also be used for more complex systems.

A filter design via the calculation of polynomial basis vectors for the required nullspace is presented in [26]. This approach provides the designer with maximum degree of design freedom. This process is summarized below for the no disturbance case and then applied to the servo fault detection problem.

In case of no disturbances the solution is based on a matrix fractional decomposition of the known input to the measurement matrix: $G(s) = D^{-1}(s)N(s)$. The transfer function matrix $B(s) = [D(s) \quad -N(s)]$ contains row vectors forming the nullspace basis of $G(s)$. Thus every row and every linear combination of the rows of $B(s)$ is a potential residual filter, solving (a) of the design problem. The second step is to shape the fault-to-residual transfer behavior, thus solving (b) and (c) of the design process. To fulfill (b) any parametric linear combination of the rows of $B(s)$, which couples the fault to the residual can be selected, by choosing a (polynomial) matrix $\phi(s)$ of suitable dimension [26]. Finally, as $B(s)$ is a polynomial basis, its rows are not proper. To fulfill the design constraint (c) $\phi(s)N(s)$ can be further parametrized with a diagonal polynomial matrix $M(s)$ containing the desired poles of the filter to make it (strictly) proper, defining the desired transfer behavior. Hence, the filter $Q(s)$ is given by

$$Q(s) = M^{-1}(s)\phi(s)B(s) \tag{4.7}$$

Thus, the only constraint limiting the design freedom is given by the row degree of $M(s)$ that is needed to make the residual filter strictly proper.

Applying the presented approach to the actuator model requires the decomposition of the actuator model from Equation (3.1) into

$$G_{fit}(s) = d^{-1}(s)n \tag{4.8}$$

with

$$\begin{aligned} n &= 122300 \\ d(s) &= (s+24.84)\,(s^2+70.04s+4921) \end{aligned} \tag{4.9}$$

resulting in the basis $B(s) = [d(s) \quad -n(s)]$. As the basis in the case of a single transfer function consists of only one vector, no further parameterization with $\phi(s)$ is required.

The last step is to define $M(s)$, which is a single transfer function in our case. To provide a fast detection of the fault and still filter out the sensor induced noise, $M(s)$ is selected as $k(0.02s+1)^4$. k denotes the dc-gain for the fault-to-residual transfer and needs to be selected. Choosing $k = n$ ensures a fault to residual dc-gain of 1. Thus, the resulting strictly proper filter is given by

$$Q(s) = \frac{1}{n(0.02s+1)^4}\left[\begin{array}{cc} d(s) & -n(s) \end{array}\right] \tag{4.10}$$

The fault-to residual transfer behavior can be derived by inserting the filter Equation (4.10) and the model Equation (4.1) into Equation (4.2), resulting in:

$$\mathbf{r}(s) = \frac{1}{(0.02s+1)^4}\mathbf{f}(s) \tag{4.11}$$

The decision logic for the detection of a fault involves comparing the generated residual to an upper and lower constant threshold $\pm\tau$. The decision variable i is defined as a boolean variable

$$i = \begin{cases} 0 & \text{if } |r| < \tau \\ 1 & \text{otherwise} \end{cases} \tag{4.12}$$

indicating the presence of a fault in the system ($i = 1$) or its absence ($i = 0$).

4.2 Results

The selection of threshold is based on residuals generated by healthy actuator data. Note that the analysis and results presented in this section are all based on testing of HITEC HS-225BB actuators using the laboratory experimental setup. Figure 4.1 shows the residuals generated by a healthy actuator for logged flight data run on the bench-top experiment (with no aerodynamic loads). The commands correspond to aileron and elevator *FASER* commands involving 180 deg turns of the aircraft in a rectangular pattern. The fault detection filter receives this flight command inputs as well as the measured actuator output of the experimental setup. We observe from Figure 4.1 that for the test signal, the residual stays well between ± 1.5 units. Hence, ± 1.5 units appears to be a reasonable threshold level.

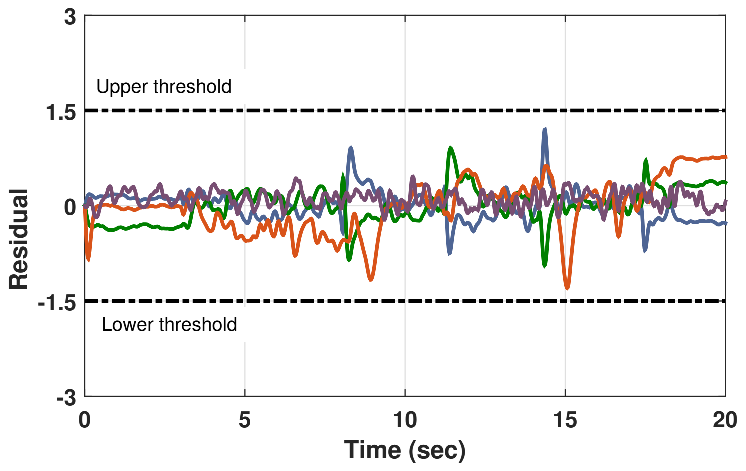

Figure 4.1: A collection of residuals generated for *FASER* flight commands sent through a healthy HITEC HS-225BB servo. The threshold for fault declaration is decided based on these residuals.

To further validate our choice of threshold, the residuals from healthy and faulty actuators are compared against the selected threshold. Figure 4.2 shows a comparison of residual generated by a healthy servo and a servo with a critical fault type (increased deadband). The ± 1.5 units threshold level is successful in differentiating between the two actuator performances. A fault is declared in the first instance of the residual crossing the set threshold limit. Note that the magnitude of the residual and the time taken to sense the fault depends on the fault level and the input command amplitude.

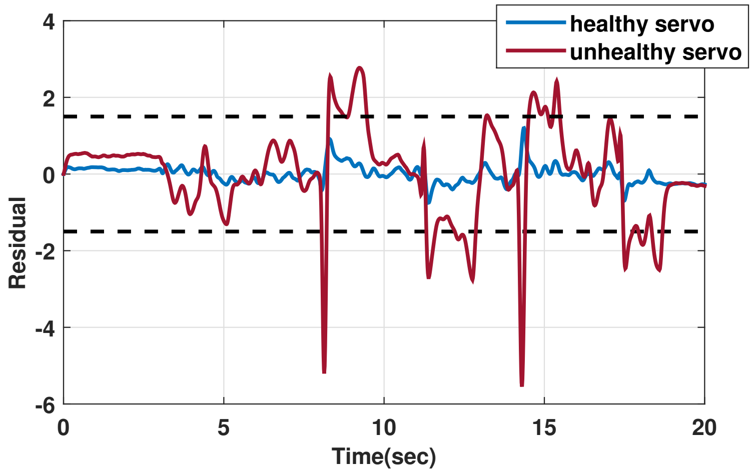

Figure 4.2: Residuals generated for a healthy and unhealthy HITEC HS-225BB servo for *FASER* flight commands.

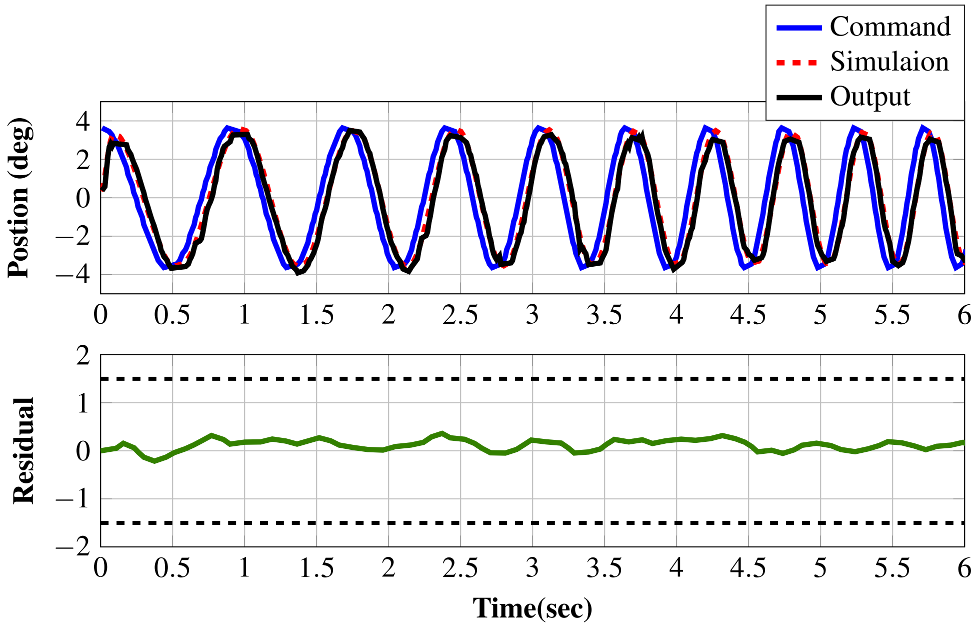

Figure 4.3: Residual generated by a healthy HITEC HS-225BB for a chirp command. The thresholds for fault detection are represented by black dotted lines.

Figure 4.3 depicts the actuator fault detection filter run on a fully functional or healthy actuator. The ’Command’ or the input is a chirp signal. The ‘Simulation’ signal is the output generated by the SIMULINK actuator model or the expected output. The ’Output’ signal is the true output obtained from experiments. For a frequency sweep command, the generated residual has a mean 0.00 and variance 0.02 units. Hence the residual is well within the set threshold limit. Next we will see the variation of the residual for the same input command on broken actuators.

4.2.1 Critical fault type: Increased deadband/stiction

As stated in Section 2.2 this failure is caused by slippage of gears or damaged servo driveshaft. Note that a UAV can still fly with an increased deadband type of actuator fault but with reduced maneuverability leading to loss of motion. Hence this is a critical fault type. Figure 4.4 shows its response for a chirp command. The movement of the servo shaft is constrained and the servo appears to be stuck at certain shaft positions. It is also accompanied by an increased current intake and higher servo noise levels. As seen in the lower plot of Figure 4.4, the residual is outside the threshold limits.

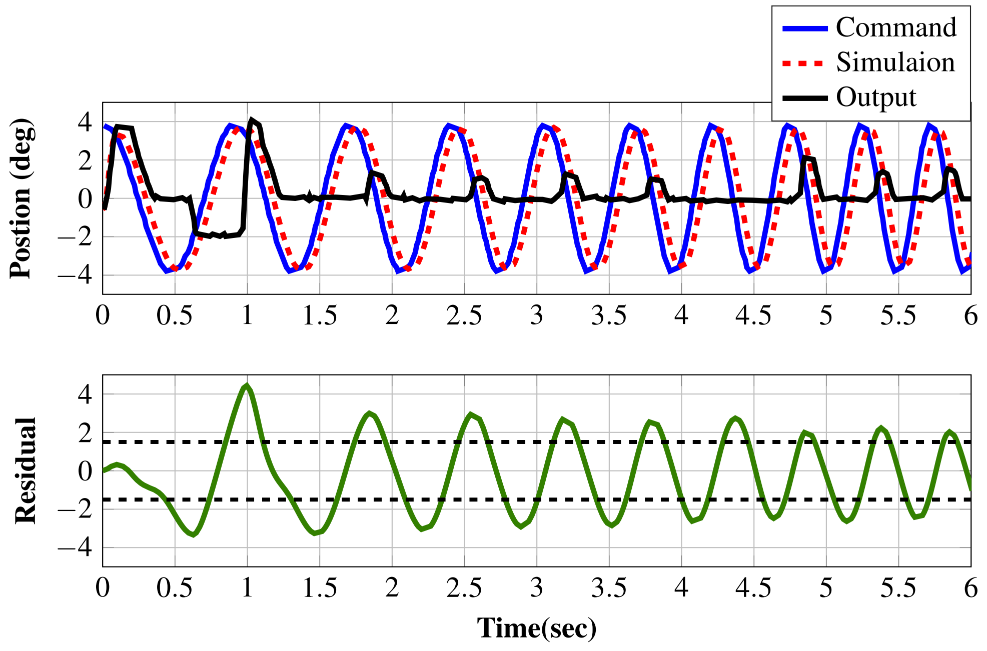

Figure 4.4: The response and residual of a HITEC HS-225BB servo with an increased deadband fault for chirp input.

Figure 4.5 represents the response and residual for the same fault for *FASER* commands. The fault is sensed by the filter at 8.05 seconds after the first large scale dynamic maneuver performed at 7.98 seconds.

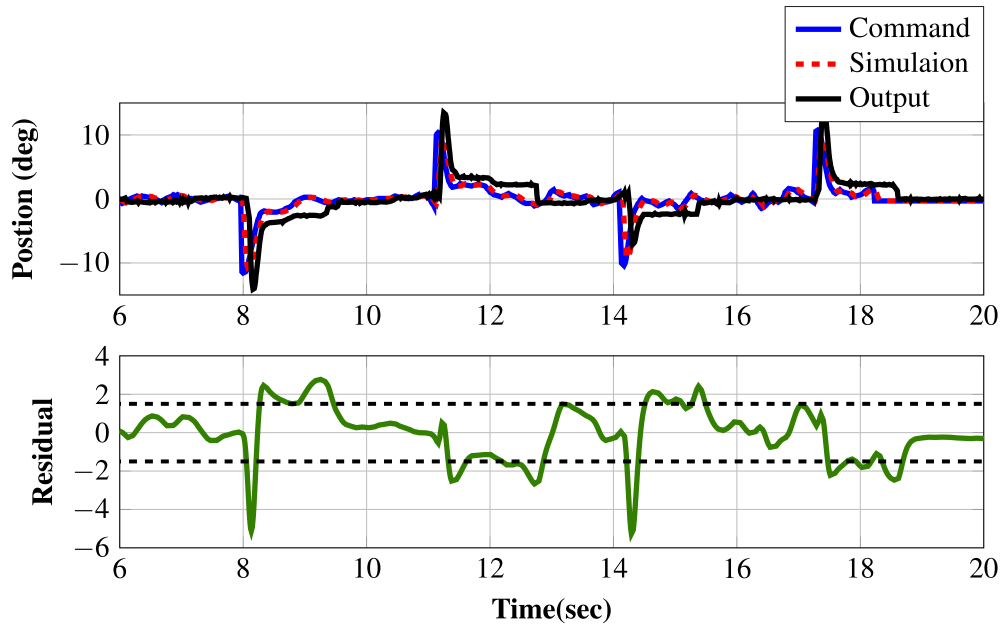

Figure 4.5: The response and residual of a HITEC HS-225BB servo with an increased deadband fault for *FASER* flight commands.

4.2.2 Catastrophic fault type: Stuck fault

This failure is caused due to a damage in the servo drive shaft, linkage or an unbalanced surface. It is a catastrophic fault which can lead to loss of motion and loss of control and in severe cases, a loss of vehicle. This fault can also be interpreted as an extreme case of the increased deadband fault. Figure 4.6 represents the response of the actuator model and the fault detection filter for a sinusoidal input. Here the servo is stuck/immovable at all positions irrespective of the initial shaft position. The current intake is observed to be higher than the normal levels (specified by the manufacturer).

Figure 4.7 shows the response of the same actuator for flight commands. In this test, the first flight command to maneuver the UAV was sent after 2.97 seconds. As soon as the UAV is maneuvered, the fault detection filter immediately senses an actuator fault and raises a flag at

3.05 seconds.

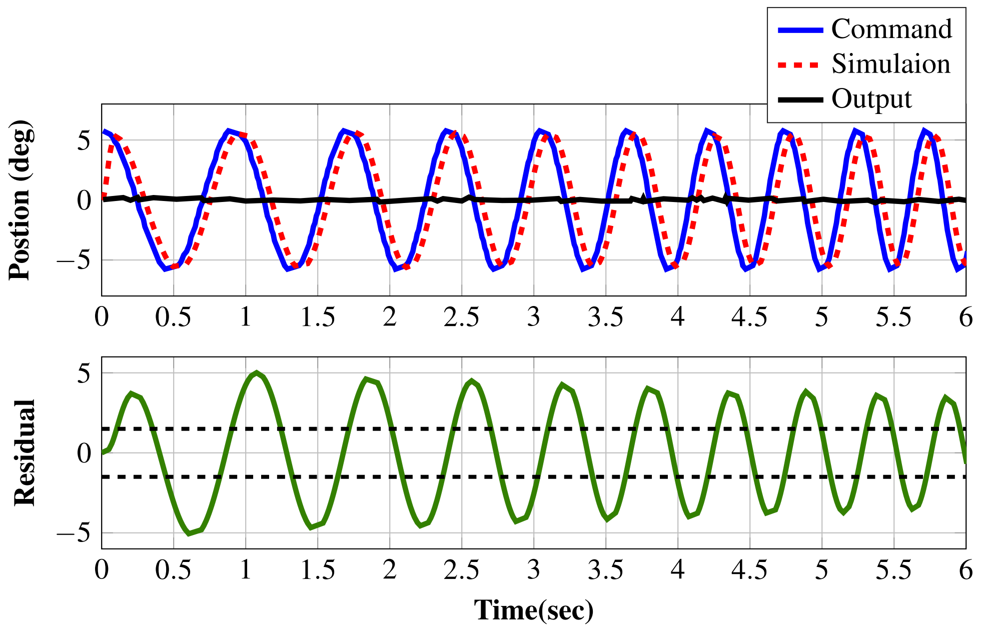

Figure 4.6: The response and residue of a HITEC HS-225BB actuator with a stuck fault for chirp input commands.

It must be noted that the fault detection algorithm proposed here is only one aspect of a complete fault tolerant approach. As soon as the fault detection filter raises a flag, the identified actuator needs to be isolated from the GNC system and a fault tolerant controller must take over to safely land the aircraft [18]. Hence a UAV can be made reliable by designing a fault detection filter followed by fault isolation and control system reconfiguration.

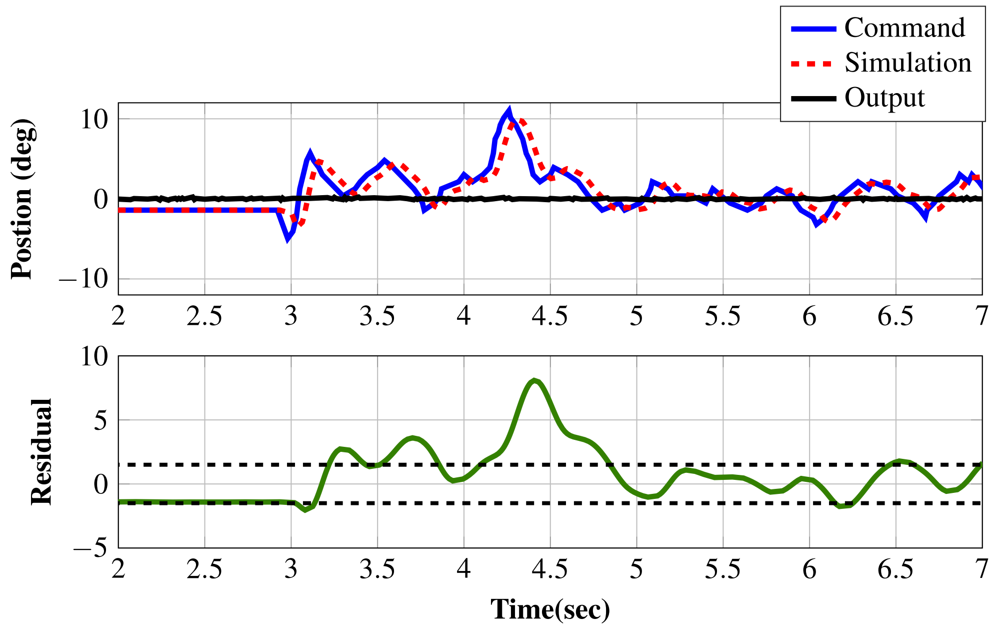

Figure 4.7: The response and residue of a HITEC HS-225BB actuator with a stuck fault for *FASER* flight commands.

Chapter 5

Conclusion

This purpose of this study was to improve safety and reliability of low-cost Unmanned Aerial Vehicles using model based fault diagnosis. The test vehicle used is *FASER* with the Ultrastick 120 airframe owned by the UAV laboratory of the University of Minnesota. From FMEA and FTA conducted on this aircraft it was evident that the rudder and elevator servos have the severest and highest failure rate of 2.17 failures for every 100 hours. Hence this thesis focused on modeling of servo actuators and design of fault detection filters for servos. In this process, an Arduino based experimental testbed was designed to perform offline analysis of servos used on UAVs. Various types of data were collected using this testbed which was used to model actuators using frequency domain system identification techniques in MATLAB. A robust second or third order model fits were obtained for five actuators (manufactured by HITEC or Futaba) used in low-cost UAVs. The bandwidth, rate limit and time delay were also determined for these servos.

Using the identified actuator model, a linear fault detection filter was designed based on polynomial basis vectors [26] to generate a residual proportional to fault. A threshold of ± 1.5 units was set for fault declaration. If the residual is within ± 1.5 at all times then the performance of the servo is said to be normal/healthy. The fault detection algorithm was tested on actuators with two types of faults: increased deadband/stiction (critical fault) and stuck fault (catastrophic fault). These tests were conducted on the actuator experimental testbed using flight data obtained from *FASER* flight tests. For both fault types, the filter was able to raise a flag indicating occurrence of actuator fault within 0.08 seconds of the first dynamic maneuver.

Hence the results from model based fault detection filter are satisfactory and confirm the adequate modeling of the underlying system, an adequate design of the fault detection filter, and the possibility to detect faults during flight, while trying to minimize the probability of false alarms.

A more comprehensive analysis of actuator fault detection can be accomplished by upgrading the laboratory experimental testbed. Currently, the testbed is being expanded to replicate aerodynamic loads experienced in flight to conduct load tests on actuators. Also there is scope for a better actuator calibration based on precise angular measurements which can aid in mapping unhealthy actuators. Besides, monitoring the servo current intake will help in achieving a more accurate actuator model. The fault detection algorithm can also be modified to include adaptive thresholds. The fault detection technique put forward in this thesis based on analytical redundancy can be further validated and supported by conducting a probabilistic analysis such as probability of false alarms and missed detections. Finally, all of this can be implemented and tested in real-time by conducting flight tests on low-cost UAVs.

References

[1] Federal Aviation Administration. *Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap*. US Department of transportation, first edition edition, 2013.

[2] Y.C Yeh. Triple-triple redundant 777 primary flight computer. In *Proceedings on the 1996 aerospace applications conference*, pages 293–307, 1996.

[3] R.P.G. Collinson. *Introduction to Avionics Systems*. Springer, 2003.

[4] University of Minnesota. www.uav.aem.umn.edu, 2015.

[5] J. Amos, E. Bergquist, J. Cole, J. Phillips, S. Reimann, and S. Shuster. UAV for reliability, December 2013. http://www.aem.umn.edu/∼SeilerControl/Papers/Seiler_All.html.

[6] J. Gertler. *Fault detection and dagnosis in engineering systems*. Marcel Dekker, Inc., New York, USA, 1998.

[7] P. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy-a survey and some new results. In *Automatica, Vol 26, No 3, pp. 459-474*, 1990.

[8] J. Chen and R.J Patton. *Robust Model-Based Fault diagnosis for dynamic systems*. Kluwer academic publishers, 1999.

[9] P. Goupil. AIRBUS State of the Art and Practices on FDI and FTC. *Control Engineering Practice*, pages 524–539, 2011.

[10] P. Freeman, R. Pandita, N. Srivatsava, and G. Balas. Model-based and data-driven fault detection performance for a small UAV. In *IEEE/ASME Transactions on Mechatronics*, volume 18, pages 1300–1309, 2013.

[11] R. Pandita, J. Bokor, and G. Balas. Closed-loop performance metrics for fault detection and isolation filter and controller interaction. *International Journal of Robust and Nonlinear Control*, pages 419–438, 2013.

[12] X. Yang, M. Warren, B. Arain, B. Upcroft, F. Gonzalez, and L. Mejias. A UKF-based estimation strategy for actuator fault detection of UASs. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS), USA*, pages 516–525, 2013.

[13] L. Ma and Y. Zhang. DUKF-based GTM UAV fault detection and diagnosis with nonlinear and LPV models. In *Mechatronics and Embedded Systems and Applications (MESA), IEEE/ASME International Conference*, pages 375–380, 2010.

[14] F. Bateman, H. Noura, and M. Ouladsine. Actuators fault diagnosis and tolerant control for an unmanned aerial vehicle. In *16th IEEE International Conference on Control Applications*, 2007.

[15] A. Zolghadri. The challenge of advanced model-based FDIR techniques for aerospace systems: the 2011 situation. In *Proc. of 4th European Conference for Aerospace Sciences*, 2011.

[16] A. Varga and D. Ossmann. LPV-techniques based robust diagnosis of flight actuator faults. *Control Engineering Practice*, 31:135–147, 2014.

[17] B. Vanek, A. Edelmayer, Z. Szab, and J. Bokor. Bridging the gap between theory and practice in LPV fault detection for flight control actuators. *Control Engineering Practice*, 31:171–182, 2014.

[18] R. Venkataraman and P. Seiler. Safe flight using one aerodynamic control surface. In *AIAA Science and Technology Forum and Exposition*, pages Paper No. AIAA–2016–0634, 2016.

[19] P. Freeman and G. Balas. Actuation failure modes and effects analysis for a small UAV. In *American Control Conference (ACC)*, pages 1292–1297, 2014.

[20] F.A. Lie, A. Dorobantu, B. Taylor, D. Gebre-Egziabher, P. Seiler, and G. Balas. An Airborne Experimental Test Platform: From Theory to Flight (Part 1). In *InsideGNSS*, pages 44–58, March/April 2014.

[21] F.A. Lie, A. Dorobantu, B. Taylor, D. Gebre-Egziabher, P. Seiler, and G. Balas. An Airborne Experimental Test Platform: From Theory to Flight (Part 2). In *InsideGNSS*, pages 40–47, May/June 2014.

[22] G. How, D. Owens, and C. Denham. Forced oscillation wind tunnel testing for FASER flight research aircraft. In *AIAA Atmospheric Flight Mechanics Conference*, August 2012.

[23] D. Owens, D.E Cox, and E.A Morelli. Development of a low-cost sub-scale aircraft for flight research: The FASER project. In *25th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*, June 2006.

[24] https://www.pjrc.com/teensy/.

[25] J. Gertler. Analytical redundancy methods in fault detection and isolation. In *IFAC/IMACS Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS*, pages 9–21, 1991.

[26] E. Frisk and M. Nyberg. A minimal polynomial basis solution to residual generation for fault diagnosis in linear systems. *Automatica*, 37:1417–1424, 2001.

Appendices

Appendix A

System Identification in MATLAB

```
%% ServoAnalysis
% This file constructs a second order model for the Futaba S9254 servo.
% The model is constructed via frequency-domain identification techniques
% using a chirp input signal.  Validation is performed in the frequency
% domain using a second set of data with an input chirp at a higher
% voltage.  Finally, step response data is used to estimate the rate
% limit and to provide a time-domain validation for the model.
%
% History:
% Initial Coding: I. Lakshminarayan and P. Seiler  5/29/2015
%

%% Load time domain data and convert to degs
% 'u': Input command, degs
% 'y' : Output position, degs

% Sample time and frequency
dt = 1e-2;                  % Sample time, sec
Fs = 1/dt;                  % Sample frequency, samples/sec

% Chirp input data for creating transfer function fit
fname = 'HITEC_225BB_Bandwidth_4_8V';
[u,y,t,cmd,pos] = load_Hitec_225BB(fname,dt);
u = u-mean(u);
y = y-mean(y);
Nt = length(t);

% Plot time domain response of input/output signals
figure(1)
plot(t,u,'b',t,y,'r--');
xlabel('Time (sec)');
ylabel('u and y');
```

```
legend('u','y');

% Make thick line types, etc for nice plots
if exist('garyfyFigure','file')
    garyfyFigure
end

%% Compute FFTs of input and output
NFFT = 2^nextpow2(Nt); % Next power of 2 from length of y
f = Fs/2*linspace(0,1,NFFT/2+1)';       % Frequency vector, Hz
w = f*2*pi;                             % Frequency vector, rad/sec
Y = fft(y,NFFT)/Nt;
U = fft(u,NFFT)/Nt;

% Single-sided amplitudes
idx = 1:NFFT/2+1;
Ymag = 2*abs(Y(idx));
Umag = 2*abs(U(idx));

% Plot single-sided amplitude spectrum for input and output
figure(2);
semilogx(w,Umag,'b',w,Ymag,'r--');
xlim([1 20]*2*pi);
title('Single-Sided Amplitude Spectrum')
xlabel('Frequency (rad/sec)')
ylabel('|U(w)| and |Y(w)|')
legend('U','Y');

% Make thick line types, etc for nice plots
if exist('garyfyFigure','file')
    garyfyFigure
end

%% Compute empirical transfer functions (raw / smooth) and fit

% Raw frequency response from FFTs of input and output
% An additional angle offset is added to Graw_ph to account for factors
% of 360degs that arise due to noise / low coherence at low frequencies.
Graw = Y(idx)./U(idx);
Graw_mag = 20*log10(abs(Graw));
Graw_ph = unwrap(angle(Graw))*180/pi+720;

% Smoothed (windowed) frequency response
% This uses the function SPA but ETFE could also be used here.
% Nwin is the window width. Smaller values give a more smooth estimate
% (lower variance) of the frequency response but with higher bias.
% The value was selected to be as large as possible (to minimize bias)
% while still giving a smooth estimate.
yudat = iddata(y,u,dt);
Nwin = 100;
Gwin = spa(yudat,Nwin,w);
```

```
[Gwin_mag,Gwin_ph] = bode(Gwin);
Gwin_mag = 20*log10(Gwin_mag(:));
Gwin_ph = Gwin_ph(:);

% Coherence
Cuy = mscohere(u,y,[],[],NFFT);

% Optimal transfer function fit:
% Use FITMAGFRD to fit the magnitude data. FITMAGFRD
% tries to minimize the peak error (in DB). This seems to provide
% better fits for data just beyond the bandwidth.  Alternative
% fits using least squares (FITFRD) provide very accurate fits
% at low frequencies at the expense of poor fits just above the
% bandwidth.  The drawback with FITMAGFRD, in general, is that it
% assumes the system is stable, minimum phase. That is fine for the
% servo actuator except that the time delay needs to be fit by hand
% in order to get the phase to match between the fit and experiment.
% The time delay is chosen mainly to fit the phase. However, it was
% also tweaked slightly to improve the fit with the time-domain, step
% repsonse data below (at the expense of less accurate fit to the phase
% in the frequency domain).
wfit = [1 20]*2*pi;       % Frequency band used for fit, rad/sec
iodel = 0.03;
Gfr = freqresp(Gwin);
widx = w>=wfit(1) & w<=wfit(2);
Gfr = frd(Gfr(widx),w(widx));
Gfit = fitmagfrd(Gfr,2,2);
Gfit.InputDelay = iodel;
tf(Gfit)

[Gfit_mag,Gfit_ph] = bode(Gfit,w);
Gfit_mag = 20*log10(Gfit_mag(:));
Gfit_ph = Gfit_ph(:);

%% Plot empirical transfer function and fit
figure(3);
subplot(311);
semilogx(w,Graw_mag,'b',w,Gwin_mag,'r',w,Gfit_mag,'g--');
xlim(wfit);
ylabel('|G(jw)| in dB')
title('Experimental Data and Fit')

subplot(312);
semilogx(w,Graw_ph,'b',w,Gwin_ph,'r',w,Gfit_ph,'g--');
xlim(wfit);
ylabel('\angle G(jw) in degs')
legend('Graw','Gwin','Gfit','Location','Best');

subplot(313)
semilogx(w,Cuy,'b');
xlim(wfit);
```

```
ylabel('Coherence');
xlabel('Frequency (rad/sec)')
set(3,'Pos',[680   115   560   863]);

% Make thick line types, etc for nice plots
if exist('garyfyFigure','file')
    garyfyFigure
end

%% Frequency-Domain Validation

% Chirp input data for validating transfer function fit
fname = 'HITEC_225BB_Bandwidth_6V';
[u,y,t,cmd,pos] = load_Hitec_225BB(fname,dt);
u = u-mean(u);
y = y-mean(y);
Nt = length(t);

% Smoothed (windowed) frequency response
yudat = iddata(y,u,dt);
Nwin = 100;
Gwin = spa(yudat,Nwin,w);
[Gwin_mag,Gwin_ph] = bode(Gwin);
Gwin_mag = 20*log10(Gwin_mag(:));
Gwin_ph = Gwin_ph(:);

% Coherence
Cuy = mscohere(u,y,[],[],NFFT);

% Plot fit and validation data
figure(4);
subplot(311);
semilogx(w,Gwin_mag,'r',w,Gfit_mag,'g--');
xlim(wfit);
ylabel('|G(jw)| in dB')
title('Validation Data and Fit')

subplot(312);
semilogx(w,Gwin_ph,'r',w,Gfit_ph,'g--');
xlim(wfit);
ylabel('\angle G(jw) in degs')
legend('Gwin','Gfit','Location','Best');

subplot(313)
semilogx(w,Cuy,'b');
xlim(wfit);
ylabel('Coherence');
xlabel('Frequency (rad/sec)')
set(4,'Pos',[680   115   560   863]);

% Make thick line types, etc for nice plots
```

```
if exist('garyfyFigure','file')
    garyfyFigure
end

%% Time-Domain Validation

% Load step-response data
fname = 'HITEC_225BB_Rate_4_8V';
%fname = 'Futaba_S9254_Rate_6V';
[u,y,t,cmd,pos] = load_Hitec_225BB(fname);

% Extract coefficients from transfer function fit.
% Note: The coefficient of the fit is tweaked to give a unity DC gain,
% i.e. a0 = b0.  This unity DC gain matches the step response data.
% However, all the units conversions, etc that appear from input to
% output must be recalibrated when the servo is installed on the a/c.
[a,b]=tfdata(Gfit);
b1 = b{1}(2);
b0 = b{1}(3);

a0 = a{1}(3);
a0 = b0;            % Enforce unity DC gain

% Set Rate and Position Limits
RateLimit = 332;       % Rate Limit at 4.8 V
%RateLimit = 417;       % Rate Limit at 6V
PositionLimit = inf;

% Simulate step response
ydot0 = 0;      % Initial velocity, deg/sec
y0 = y(1);      % Initial position, degs
uin = [t(:) u(:)];
sim('ServoModel',[0 t(end)]);

% Plot results
figure(5);
plot(t,u,'b',t,y,'r',tsim,ysim,'g--');
xlabel('Time, sec');
ylabel('Postion, degs');
legend('Ref','Exper','Sim','Location','Best')

% Make thick line types, etc for nice plots
if exist('garyfyFigure','file')
    garyfyFigure
end
```

Appendix B

Fault detection filter in MATLAB

```
function [Qvecs,info] = getbasicQ(sys,p)
%% [QVECS,INFO] = getbasicQ(SYS,P) generates an FD filter system Qvecs
% assuming input faults on all inputs of sys.
% Each row of the output Qvecs is a potential FD filter and generates a
% residual:
%                                        [ y ]
%                       r_i = QVECS(i,:) [ u ]
% using the system command inputs u and measurements y to generate the
% resiudal r_i. The filter decouples the system inuts from the residual
% while it couples the faults. The order of Qvecs is n+1, where n is the
% order of the common denumerator of sys. This ensures strictly proper TFs
% from faults to resiudals. In general the filter will have n+1 stable
% poles at P.In case of a single input system the algorithm tries to
% remove (stable) zeros from fault to residual channels.The multiplicity
% of each pole p in the channel i is n+1-m_i, where m_i is the number
% zeros to beremoved in channel i. The output info contains the command
% input to resiudal and fault toresiudal systems as well as its dc gains.
%------------ Version Information --------------------------------------%
% V0.1. - D. Ossmann, UMN, 17.9.2015: Basic input decoupling (runs w/o
% robust control TB (no coprime factorization function used)
%%---------------------------------------------------------------------%

if p>=0
   warning('For a stable filter second input (pole) has been'...
   'multiplied by -1')
end
if strcmp(class(sys),'ss')
   sys = tf(sys);
end

%% Gernating raw filter
[out,in] = size(sys);
[num, den] = tfdata(sys);
```

```
ord = order(sys(1,1));

DL = eye(out)*tf(den{1,1},1);
NL = tf(num,1);
Q = [DL -NL];

%% Filter processing
% try cancelling zeros from the f->r transfer function input faults if
% SI-system
if in==1
    M(out,out)=tf(0,1); %init
    for i =1:out
        z = zero(NL(i));
        if ~isempty(z)
            iz = find(z<0);                 % stable zeros
            z  = z(iz);
            Mtemp   = zpk([],z,prod(z));    %filter to cancel stable zeros
            dord    = order(Mtemp);
            M(i,i)  = Mtemp *(zpk([],-abs(p),-abs(p)))^(ord+1-dord);
        else
            M(i,i)  = (zpk([],-abs(p),-abs(p)))^(ord+1);
        end
    end

else
    M = eye(out)*(zpk([],-abs(p),-abs(p)))^(ord+1);
end

% get System from control input to filter output
 rusys = Q*[sys;eye(in)];

 dcgain(Q*[sys;eye(in)]);
 % get System from fault input to filter output (assuming additive
 % input faults)
 rfsys = minreal(M*Q*[sys;zeros(in,in)]);

 % Scale to a dcgain of at least +1 for the smallest residual in each
 % row of the filter
 if in>1 && out>1
    DCmin    = dcgain(rfsys);
    % daignoal matrix with invers dcgains
    Mdc      = diag(min(abs(DCmin')).^-1);
    % get correct sign for dcgain of +1
    indx     = diag(min(abs(DCmin')))*ones(out,in)== abs(DCmin);
    Mdcsign = diag(sum(sign(indx .* DCmin)'));
 else
     % vector operation in case of single input OR single output
     if in == 1
        Mdc = diag(dcgain(rfsys).^-1);
        Mdcsign = 1;
      else
       DCmin    = dcgain(rfsys);
       Mdc      = min(abs(DCmin)).^-1;
       indx     = min(abs(DCmin))==abs(DCmin);
       Mdcsign  = sum(sign(indx .* DCmin));
     end
 end

 % Get filter together
 Qvecs = Mdcsign*Mdc*M*Q;

 % Generate faults to residals system
 rfsys = minreal(Mdcsign*Mdc*rfsys);

 %% Additional Outputs
 info.rusys = rusys;
 info.dc_rusys = dcgain(Mdcsign*Mdc*M*rusys);
 info.rfsys = rfsys;
 info.dc_rfsys = dcgain(rfsys);
```

Appendix C

Actuator testing in Arduino

```
// Taylor, Brian R
// brtaylor@umn.edu
//
// 20150109
// This code is for conducting servo rate limit tests. A step command is
// sent to the servo and the position is measured via a potentiometer and
// an analog input pin. Additionally, the pwm command is fed back with an
// analog input pin and stored along with the output command. The program
// runs at 10 kHz and outputs the data in tab seperated formated over the
// serial monitor following completion of the test.
//
// Updated by Inchara Lakshminarayan on Nov 29, 2015
//
// The code was modified to run tests at varying PWM update frequencies.

int servo=5;      //Pin no. of the microcontroller connected to the servo
int PotPin = 17; // analog pin that I'm reading
int pwmPin = 22; // analog pin for the pwm input

int meas[3000] = {0}; // value of the analog signal
int cmd[3000] = {0}; // value of the command
int pw[3000] = {0}; // value of the measured pwm
// position to command to the servo, 60 degrees (FUTABA S9251)
float poscmd[3000] = {0};

// timing variables to stabilize the frame
unsigned long start_frame;
unsigned long stop_frame;
unsigned long delta_frame;
float pwmFreq= 300.0;  //Set pwm frequency
unsigned long pwmTime=(1/pwmFreq)*1000000;
```

```
int k = 0; // used to stop the loop after one run

void setup()
{
  Serial.begin(115200); //  setup serial
  analogReadRes(16); // set A/D converter to 16 bit
  delay(1000);
  analogWriteResolution(16);
  analogWriteFrequency(servo,pwmFreq);
}

void loop()
{
  delay(1000); // wait before starting the test
  if (k < 1) { // only run the test once, then we'll set k = 1
    for (int i = 0; i < 3000; i++) {
      if (i < 1000) {
        poscmd[i] = 500; //60 deg for Futaba S9254
      }
      else {
        poscmd[i] = 900; //120 deg for Futaba S9254
      }
      start_frame = micros(); // get the start of the frame
      analogWrite(servo,poscmd[i]/pwmTime*65535.0);
      meas[i] = analogRead(PotPin);    // read the input pin
      pw[i] = analogRead(pwmPin); // read the pwm signal
      stop_frame = micros(); // get the end of the frame
      delta_frame = stop_frame - start_frame;
      if (delta_frame < 1000) {
      // sleeping to stabilize the frame at 1 kHz
        delayMicroseconds(1000 - delta_frame);
      }
    }
    // switch flag so we don't run this again
    k = 1;
    delay(1000); // wait to get the serial monitor open
    // printing out the stored data so we can import to MATLAB
    for (int j = 0; j < 3000; j++) {
      Serial.print(meas[j]);
      Serial.print("\t");
      Serial.print(pw[j]);
      Serial.print("\t");
      Serial.println(poscmd[j]);
    }
  }
}
```