Implementation, Testing and Application of PROTEST: Simulator Motion Expert Tuning Software

by

Opale M. Hullen

A thesis submitted in conformity with the requirements
for the degree of Master's in Applied Science
Graduate Department of Aerospace Studies
University of Toronto

Pour mon père,

Tes yeux ne verront jamais ses lignes.
Et pourtant, elles n'auraient existé sans toi.

Papa, tu me manques énormément.

ABSTRACT

The tuning of motion base simulators has always been a necessity in achieving reliable motion response. Regardless of the scheme used to transform forces and rates into hardware motion response the key always remains in a set of coefficients. In the past it has been necessary to employ a tuning expert and a pilot's subjective opinion of motion to tune motion bases. However, developed in 1996, PROTEST sought to simplify the tuning process by capturing, in a computer software package, the expertise of the motion-tuning specialist.

Designed to work within the UTIAS Flight Simulator Laboratory environment of the time, PROTEST was found to tune the simulator to acceptable levels. Since that time however, the laboratory hardware and software configuration has changed resulting in the need to upgrade PROTEST.

The goal of this project was to adapt PROTEST to the current UTIAS Flight Simulation Laboratory environment and report the steps taken in this document. This work identified the primary adaptation requirements to lie in the communication required between the PROTEST and host system modules and in incorporating PROTEST components into the host simulator code.

The above tasks completed the entire system was tested for the correct response. This was achieved by controlling the input to the washout filters and observing the response obtained by PROTEST. Once the results were deemed correct a sample tuning procedure was carried out to illustrate that the adaptation was a success. Indeed, this goal was fulfilled in the last of three test experiments when the flight conditions obtained were deemed satisfactory by the test pilot.

The outcome of the project has led to a functioning version of PROTEST and a comprehensive document to aid any further adaptations of PROTEST.

ACKNOWLEDGEMENTS

I would first like to thank Dr. L. Reid for giving me the opportunity to work on this project. This project began with the daunting task of reconstituting someone else's computer code, and his help proved invaluable in this. I want to truly thank him for his help and most importantly for his patience

The completion of this project would have been impossible without the help of Mr. Wolfgang Graf. His knowledge of every detail of the UTIAS Flight Simulator Laboratory made each step less daunting. I want to thank him for the long hours he has put into helping me with this project, especially those surprise solutions to the problems I would sometimes find waiting for me in the morning.

Also invaluable was Capt. Dan Sattler who flew all my test flights. I want to thank him for the expertise his has shared with me from the very start of this project and his seemingly bottomless patience with my many problems during test flights. Most of all I want to thank him for fitting me into his very busy schedule that has him travelling around the country.

I would also like to thank all the support staff at UTIAS. I believe no degree could be completed without these amazing people to help us along. Although not directly involved

in my research, my stay would not have been as rewarding an experience without you. Gail, I want to thank you for the support you gave me both personally and professionally, it was truly invaluable.

I wish to thank my friends, both the ones I left in Winnipeg and the new friends I made in Toronto. The latter made Toronto a friendlier place we could discover together, thus adding a colourful life outside of my studies.

Finally I wish to thank my family who have never ceased to believe in my abilities and have encouraged me everyday of my life. To my mother and sister who saw me leave Winnipeg and venture out on my own, and yet whose love and support I have never stopped feeling. Lastly, I wish to thank my father whose strength I feel to this day. I wish only to become half the person he was and to make him proud in the attempt.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

The symbols used throughout this document are listed below in the order in which they appear.

$\underline{f}$	specific force
$\underline{a}$	acceleration
$\underline{\underline{L}}_{IS}$	rotation matrix that transforms vector component from the simulator frame to the inertial frame
$\underline{g}_I$	ravity vector expressed in the inertial frame
$\underline{\beta}_=$	Euler angles fo frame X $[\phi_X, \theta_X, \psi_X]^T$
s	Laplace operator
$\omega_{hp\_}$, $\zeta_{hp\_}$	2nd order high-pass break frequency and damping ratio
$\omega_{lp\_}$, $\zeta_{lp\_}$	2nd order low-pass break frequency and damping ratio
$\omega_{b\_}$	1ST order high-pass break frequency
$\underline{\underline{T}}_S$	transformation from angular velocity to Euler angles rates
$\underline{\omega}$	angular rates
$\underline{\underline{C}}$	tilt co-ordination matrix

CHAPTER 1

INTRODUCTION

The development of high fidelity flight simulators has been a goal of researchers for many decades. From the first simulators with fixed bases to the more sophisticated multi-degree of freedom full motion simulators, the goal has always remained to provide the pilot with the most accurate perception of flight. To this end, researchers and designers have employed a variety of both software and hardware techniques. These have included reality based simulator cockpits and virtual reality visual displays. The increased understanding, and hence complexity, of flight equations and simulations has created the demand for highly sophisticated computer models and powerful processors. The increased computing abilities have made moving base simulators produce highly reliable motion cues. For these systems, computer models are used to represent the response of the aircraft to pilot control inputs in a given circumstance. This response takes the form of a set of translational specific forces and angular rates that represent the motion cues a pilot would experienced under 'real' conditions. It is the goal of the simulator motion base to reproduce these motion cues. However, because the motion of the simulator is constrained by its mechanical limits the response from the flight equations must be altered to prevent it from attaining its bounds. This task is fulfilled by implementing what is commonly called the washout algorithm. This

algorithm transforms the aircraft motions into commanded simulator motion [1]. Its performance thus significantly impacts the fidelity of the simulator motion. Currently there exist a number of washout filter schemes including, but not limited to, the classical algorithm, the adaptive algorithm, the optimal control approach, quasioptimum control technique, and the subliminal scheme. [1] However different these approaches are to one another, at their core is a set of coefficients that determine the behaviour of the washout filter. To achieve the best possible motion it is therefor necessary to determine the best combination of coefficients. This task, called tuning, has been accomplished by having a pilot fly a given manoeuvre and then relaying the fidelity of the motion to a tuning expert who modifies the coefficients to attempt to improve the motion. Although it has given acceptable results, this method introduces the subjectivity of both a pilot and a tuning expert. Not only does this technique not ensure the best possible coefficients have been achieved but it also prohibits the testing of pilot variability.

Dr. P. Grant, in his Ph.D. thesis *The Development of a Tuning Paradigm for Flight Simulator Motion Drive Algorithms*, addressed these issues. His research led to the development of PROTEST, expert tuning software for the UTIAS flight simulator. This software removed the need for an expert tuner by replacing him with a software package. Using this package, tuning is accomplished by entering pilot comments into PROTEST that then uses a set of rules to modify the coefficients. Test flights are repeated until a satisfactory set of coefficients is found.

Completed in 1996 Dr. Grant's work has remained unused until now. Recent simulator projects have rekindled the need to tune the UTIAS simulator effectively. However, the UTIAS simulator facilities have changed since 1996 and consequently PROTEST was not configured to function in the new computing environment.

It is the task of this thesis to describe the method used to reconfigure PROTEST to function within the computing environment of the present UTIAS flight simulation laboratory. A detailed description of the original and final configurations will be given as well as the reasoning behind each change. The main components of PROTEST will be outlined so that future implementation can be achieved more easily. Finally, the results from a set of tuning experiments will be presented with a description of their relevance.

CHAPTER 2

UTIAS FLIGHT LAB CONFIGURATION

The task of re-adapting PROTEST to the UTIAS computing environment first requires a comprehensive understanding of the different components of the simulator laboratory and their inherent interdependence. These components include both hardware and software configurations as well as the user interfaces. The implementation of PROTEST by no means impacts every aspect of the simulator configuration; however, a description of the current system as a whole is given for completeness and future reference. In addition, a description of the system components as they were at the completion of PROTEST in 1996 is given.

2.1 Hardware

2.1.1 Simulator

The UTIAS flight simulator is comprised of a DC8 simulator cab mounted on a CAE series 300 motion base. This configuration is used to produce six-degree-of-freedom (6 DoF) synergistic motion driven by six hydraulic actuators. Each actuator has a stroke of

91.4 cm and bore of 8.9 cm. Figure 2.1 gives a picture of the motion base/cab system at UTIAS.

2.1.2 Computing

The numerous cues required in producing a realistic aircraft simulation and the limited computing abilities of any single processor have created the need for multiple computing systems. These include the generation of motion, visual and auditory cues as well as user control interfaces. Below is a basic description of the functions of each computer available in the laboratory (a more detailed description is given by Tai [2]). Figure 2.2 gives a graphical description of the flow of information between computers.

The most basic requirement of the simulator is the computing of the flight model. This is achieved with an IBM RISC 6000 Model 390 Power 2 computer that samples the control inputs and solves the flight equations using a 60Hz iteration cycle.[2] Within the framework of the CAE Simulation Management Utility (SIMex-PLUS), the various modules required to generate the simulation are controlled on this machine. Terminals connected to the RISC 6000 (also called VUOT) are used to modify simulation modules and for real-time user interfaces. An Intel 186/03 Single Board Computer (SBC) provides the interface to the motion and sound systems from the RISC. At the time of the PROTEST development the host computer was a Perkin-Elmer 3250 computer with its own simulation management system.

The motion of the simulator is controlled by an analogue control system called the N1 cabinet. This system takes the digital inputs from the RISC 6000 and creates the appropriate response of the servo-valves to extend the actuators. [3]

The inputs for a live simulation are the movements of the various control devices in the simulator cab. In the case of a helicopter simulation, these are the collective, pedals, and cyclic. A McFadden System Inc. Model 292B Universal Variable Digital Cockpit Control Force Loading System provides the longitudinal and lateral cyclic forces.

The UTIAS laboratory uses virtual reality as its primary source for visual cues. To generate the virtual world for the pilot the graphic display is generated using a CAE MaxVue Enhanced B Image Generator (IG) that sends the images to the CAE Fiber-Optic Helmet-Mounted Display. Databases are created on two Silicon Graphics Indigo 2 IMPACT

10000 workstations. The secondary sources of visual cues are the various instrument displays provided to the pilot. The current configuration includes options for an Electronic Flight Information System (EFIS) and a Head-Up Display (HUD). The former is generated by a Silicon Graphics IRIS 3130 workstation (IRIS2) while the latter is generated by a Silicon Graphics 4D/310 workstation (IRIS5). A course map is generated by a Silicon Graphics personal IRIS workstation (HAL).

The user interface and statistics display is run during a simulation on an IBM RISC PowerStation 355, which is used for its capability in running the X11 windowing systems. The auditory cues are generated by an E-mu System Inc. E-max digital sampling keyboard connected to the RISC 6000. Finally a Polhemus Model 3SF0002 3SPACE FASTRAK Magnetic Head Tracker is used to track pilot head movement.

2.2 Software

There are a number of software routines used to run the simulator. As is the case for the hardware, the software is spread over the various computers to perform each of their required tasks. It is not the purpose of this thesis to delve into the details of each of these. However, those relevant aspects of the software environment will be discussed in as much detail as is necessary. It should be noted that the SIMex environment was not present at the time of PROTEST's initial development. As well, the original code that ran on the host computer (Perkin Elmer 3250) is no longer available, negating any discussion of its configuration.

2.2.1 SIMex

The amount of computing required to run the simulator requires software designed to manage the many connections between the various components of the code (herein called modules). As mentioned above, this task is fulfilled by SIMex-Plus that manages revision control and program integrity. It loads and launches the real time programs (also called the synchronous module SPOC0.exe) and the non-real time programs (asynchronous module APOC0.exe). Finally, it sets up the user control program with the initial conditions and the start/stop flags for the simulation.

2.2.2 The Bell 205 Fly-By-Wire Helicopter

The UTIAS flight simulation laboratory can be configured to represent various aircraft. These include both fixed wing and helicopter simulations. At the time of PROTEST's development the laboratory was predominately simulating fixed wing aircraft, specifically large commercial aircraft. Research demands, however, have shifted to helicopter simulation, with one example being the Bell 205. This section describes the software configuration used by the UTIAS flight simulation laboratory at the start of this thesis, and as such forms the baseline for this project. The modifications made to the code will be discussed in Chapter 5.

The flight model used for the Bell 205 helicopter is a modified version of the ARMCOP computer code developed by the National Aeronautics and Space Administration (NASA) and the U.S. Army Aviation Systems Command. The simulator code is divided into logical modules, each filling a specific task, with some variables shared by all modules through the common data block (CDB). Those modules relevant to PROTEST are discussed below by stepping through their hierarchy of calls. As has been mentioned, the simulator code can be divided into two categories: real and non-real time. Components of PROTEST will ultimately appear in both parts, and as such a brief description of relevant modules is in order. Figure 2.3 graphically represents the relevant real-time modules in their hierarchy and a break down of the three parts of the non-real time module.

2.2.2.1 Real-Time

The module *FLGHTEQ* serves to call the ARMCOP model, visual and washout modules. These modules work together to provide the motion and visual cues for the simulation. To accomplish this, the modules are run at 60Hz. The output from the helicopter ARMCOP model modules, such as the powertrain and gear modules, provides the input for the WASH module. This output takes the form of specific forces and angular rates. The WASH module then calculates the jack extensions that will drive the simulator. As its name indicates the WASH module houses the washout filters.

2.2.2.2 *Non-Real Time*

The relevant non-real time module is *NRT*. The module takes the user through a number of questions that set up the initial conditions, environment, and variable flags for the upcoming run. *NRT* provides user control for 'during flight' variable changes such as sky cover, timers, freeze, and start/stop toggles. User control for PROTEST will be implemented in this module. From the *NRT* module, the *SETWASH* module that serves to read the coefficients from a data file is called.

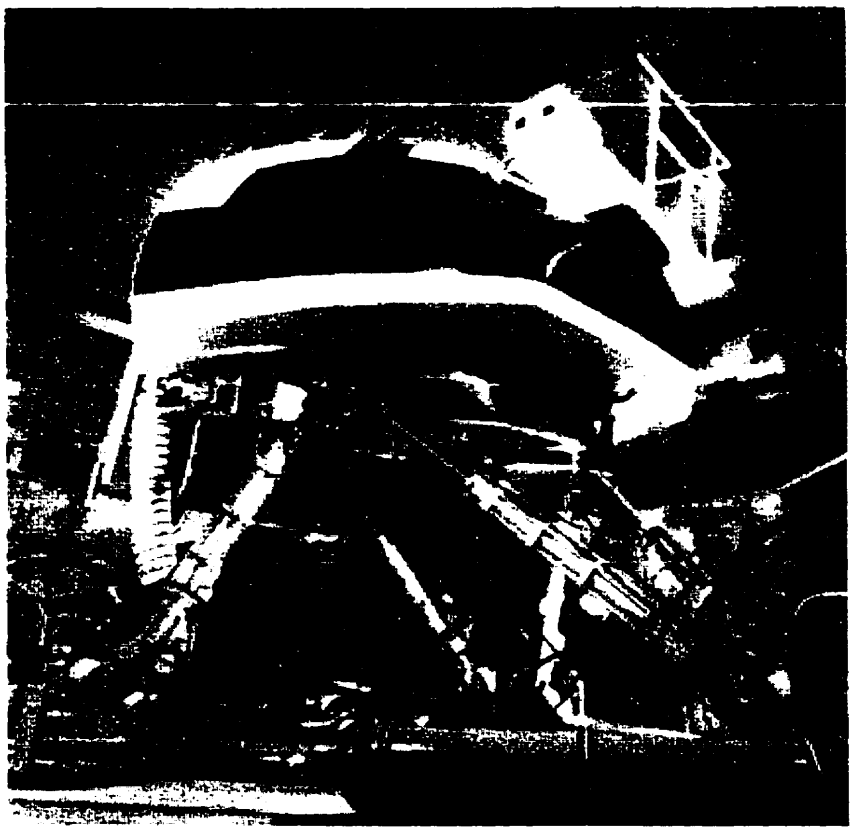

Figure 2.1 UTIAS Flight Research Simulator

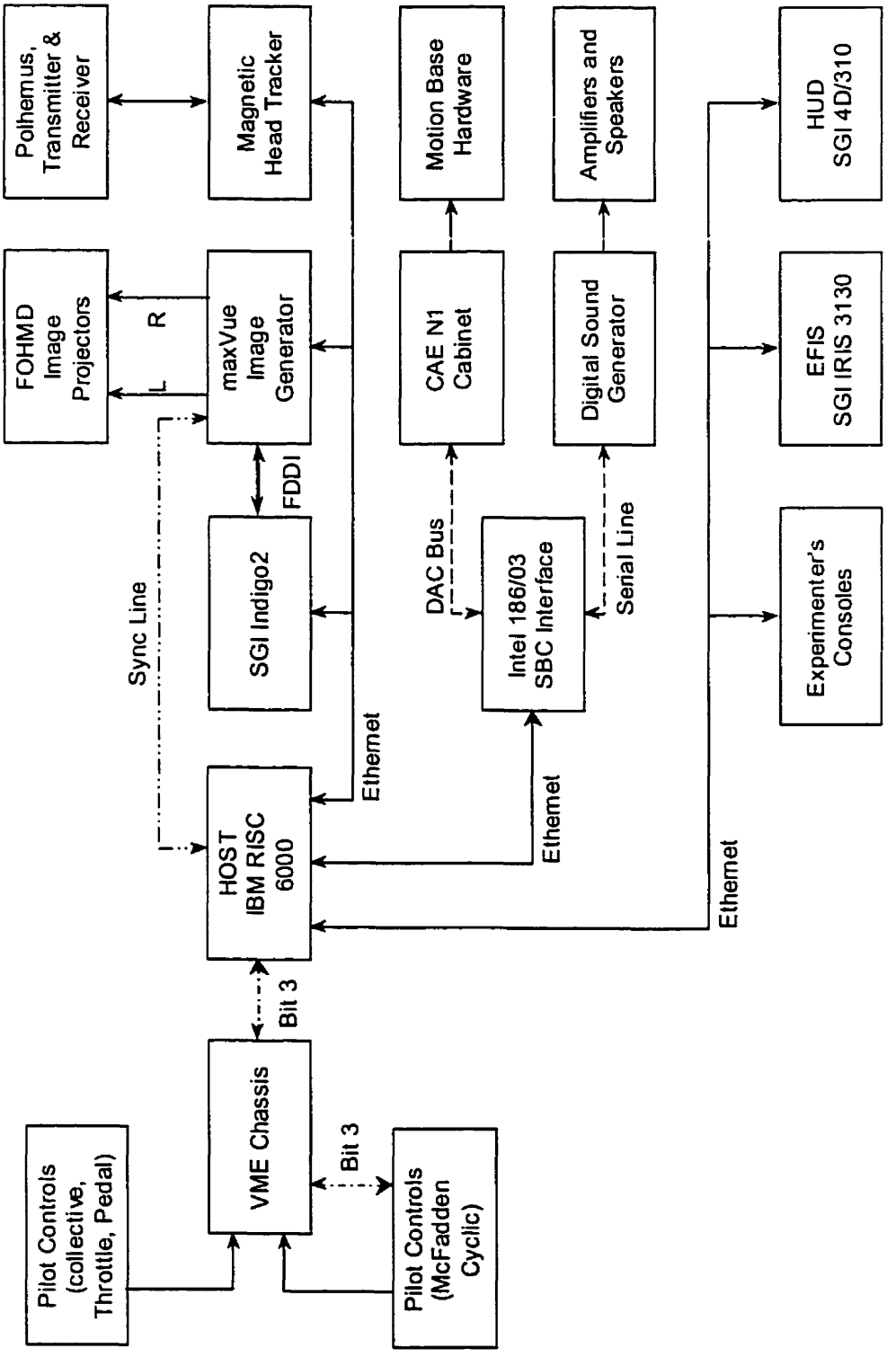

Figure 2.2 Simulator Hardware Interconnections [7]

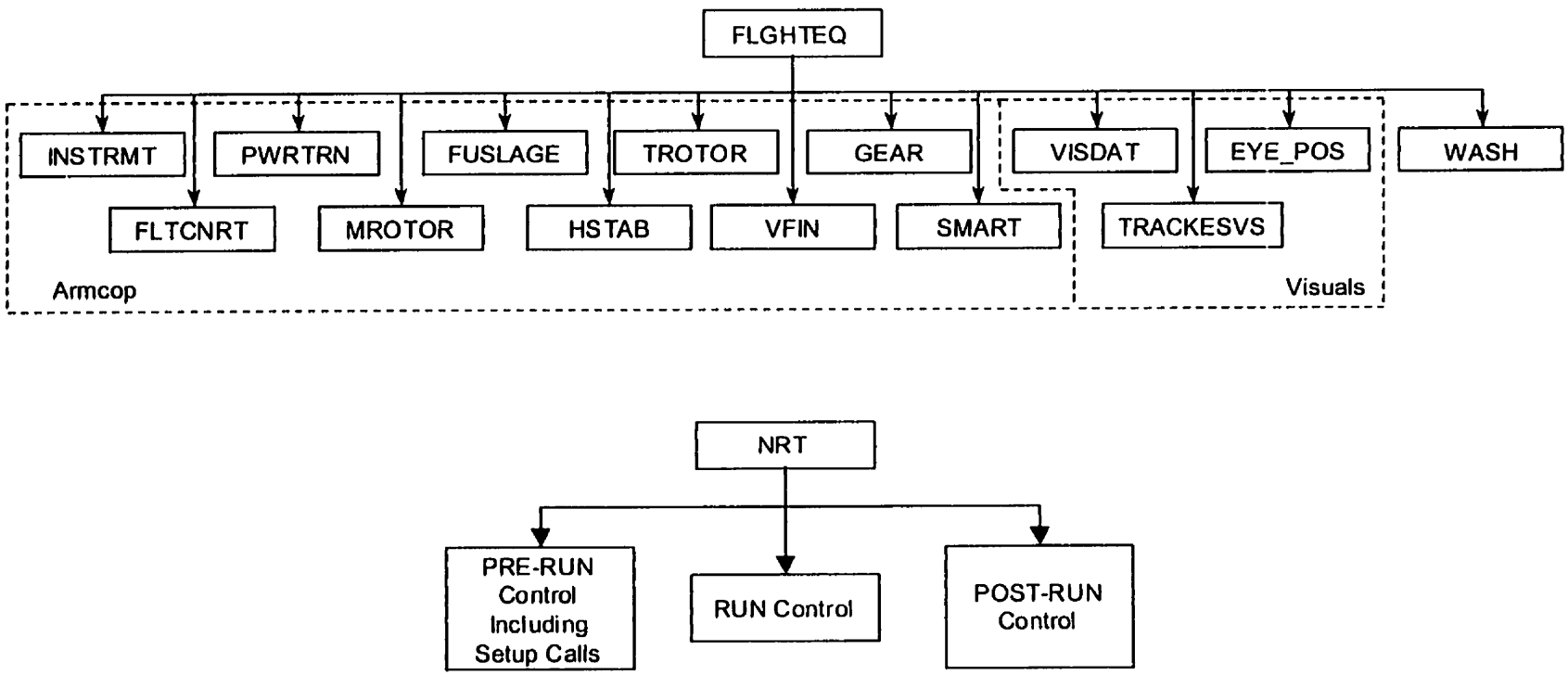

Figure 2.3 Partial Representation of the UTIAS Flight Simulator code

CHAPTER 3

UTIAS CLASSICAL WASHOUT ALGORITHM

Since the purpose of PROTEST is the tuning of the washout filters, it is crucial to have a firm understanding of how these filters work. Reid and Nahon describe the original development of the UTIAS washout filters in 'Flight Simulation Motion-Bas Drive Algorithms: Part 1 - Developing and Testing the Equations' [4]. This document provides an in-depth description of the development process of the washout filter equations. An improved version of these filter equations exists in the subsequent document, 'Flight Simulation Motion-Base Drive Algorithms: Part 2 - selecting the system parameters' also by Reid and Nahon [5]. Grant used this latter version in the initial development of PROTEST. However, as has been previously mentioned, the starting point for this project was the version of the ARMCOP Bell 205-helicopter simulation code in current use at the UTIAS Flight Simulation Laboratory. This version, in this documented called the baseline, of code uses a slightly modified form of the washout filters. The first section of this chapter will provide a description of the washout filters as they existed at the beginning of this project. The version of the algorithm used for PROTEST will then be discussed. Figure 3.1 and Figure 3.2 give block diagrams of the two different washout filter configurations.

3.1 Baseline Algorithm

The baseline algorithm, described by Figure 3.1, was developed to simulate a Bell 205 helicopter. Although the basic principles behind the washout filters is the same as that described by Reid and Nahon [5] the implementation is slightly different. Since this was the form of the washout filters used at the beginning of this project it will be described first.

3.1.1 High Pass Specific Force Channel

There are three translational specific force channels: surge, sway and heave. Each receives specific force input in the simulator body frame that represents what the pilot should feel in the cab. This input is first limited in order to protect the simulator from reaching its limits. This limit is typically set at $\pm 15 m/s^2$ for all channels with the heave channel set at ($g \pm 15 m/s^2$). The input to the limiting block is $\underline{fca}$ with the output being $\underline{fl}$. This input is then transformed into the inertial frame from the simulator body frame using the rotation matrix L_{IS}. The output from this block is $\underline{a}_C$ where,

$$\underline{a}_C = \underline{L}_{IS}\,\underline{fl} + \underline{g}_I \tag{1.1}$$

and

$$\underline{L}_{IS} = \begin{bmatrix} \cos\theta_S \cos\psi_S & \sin\phi_S \sin\theta_S \cos\psi_S - \cos\phi_S \sin\psi_S & \cos\phi_S \sin\theta_S \cos\psi_S + \sin\phi_S \sin\psi_S \\ \cos\theta_S \sin\psi_S & \sin\phi_S \sin\theta_S \sin\psi_S + \cos\phi_S \cos\psi_S & \cos\phi_S \sin\theta_S \sin\psi_S - \sin\phi_S \cos\psi_S \\ -\sin\theta_S & \sin\phi_S \cos\theta_S & \cos\phi_S \cos\theta_S \end{bmatrix} \tag{1.2}$$

with ϕ_S, θ_S, and ψ_S representing the Euler angles, as described by Etkin [6], to rotate F_I into F_S. a_C is then passed through the scaling block and the high pass filters. For the surge and sway channels the filters are second-order and have, in the Laplace domain, the form:

$$\overline{a_{SI}^x} = \overline{a_C{}^x}\left(\frac{s^2}{s^2 + 2\zeta_{hpx}\omega_{hpx} + \omega_{hpx}^2}\right) \tag{1.3}$$

$$\overline{a_{SI}^{y}} = \overline{a_C{}^{y}}\left(\frac{s^2}{s^2 + 2\zeta_{hpy}\omega_{hpy} + \omega_{hpy}^2}\right) \tag{1.4}$$

while the heave channel uses a third-order filter of the form (in the Laplace domain):

$$\overline{a_{SI}^{z}} = \overline{a_C{}^{z}}\left(\frac{s^2}{s^2 + 2\zeta_{hpz}\omega_{hpz} + \omega_{1hpz}^2} \cdot \frac{s}{s + \omega_{2hpz}}\right) \tag{1.5}$$

This third-order filter is formed by multiplying second and first-order filters. Setting ω_{2hpz} to zero reduces the system to the second-order filter of the form used for surge and sway. The outputs to the filters are double integrated to produce S_I which is the inertial displacement of the simulator reference point. Although present in the above filter equations, the values of the damping coefficients, ζ, are implicitly set to 1 by their omission in the computer code.

3.1.2 High-Pass Angular Rate Channel

Like the specific force filters there are three angular rate channels, roll, pitch and yaw. The input into the washout filter system are commanded angular rates that represent what would be felt by the pilot during a normal flight. Equivalently to the specific forces the inputs are limited to prevent the actuators from reaching their bounds. For the angular rates the limit is 34.4 °/s. Following limiting, the angular rates are transformed to Euler angle rates, with $\underline{T}_S$ given by Etkin [6], such that

$$\underline{\dot{\beta H}} = \underline{T}_S\,\underline{\omega 1} \tag{1.6}$$

where $\underline{\omega 1}$ is the input to the limiting with,

$$\underline{\omega 1} = [p1 \quad q1 \quad r1]^T \tag{1.7}$$

$$\underline{T}_S = \begin{bmatrix} 1 & \sin\phi_S \tan\theta_S & \cos\phi_S \tan\theta_S \\ 0 & \cos\theta_S & -\sin\phi_S \\ 0 & \sin\phi_S \sec\theta_S & \cos\phi_S \sec\theta_S \end{bmatrix} \tag{1.8}$$

$$\underline{\dot{\beta H}} = [\dot{\phi}_C \quad \dot{\theta}_C \quad \dot{\psi}_C]^T \tag{1.9}$$

It should be noted that, as described by Reid and Nahon[5], the following process is only an approximation. The total simulator Euler angles (those contributed to by both the high and

low pass filters) are used to transform $\underline{\omega 1}$, which itself is only used in the high-passed angular rates. The Euler angle rates are then scaled and independently passed through a set of high pass filters that vary for each channel. The pitch and roll high pass filters are both first-order systems, in the Laplace domain, of the form given below.

$$\overline{\dot{\phi}_{HP}} = \overline{\dot{\phi}_C}\left(\frac{s}{s+\omega_{hp\phi}}\right) \tag{1.10}$$

$$\overline{\dot{\theta}_{HP}} = \overline{\dot{\theta}_C}\left(\frac{s}{s+\omega_{hp\theta}}\right) \tag{1.11}$$

For yaw, however, a second-order filter is used. Unlike the motion in pitch and roll, helicopter flight may engender sustained motion in yaw. A second-order filter will give the simulator a greater ability to return the simulator to neutral. The filter takes the form of two first-order filters in cascade, represented in the Laplace domain by,

$$\overline{\dot{\psi}_{HP}} = \overline{\dot{\psi}_C}\left(\frac{s}{s+\omega_{1hp\psi}}\frac{s}{s+\omega_{2hp\psi}}\right) \tag{1.12}$$

Finally, the output from each filter is integrated once to give the Euler angles $\underline{\beta SH}$ that will be subsequently added to the low-pass filter output Euler angles.

3.1.3 Low-Pass Specific Force Channel

The low-pass filters serve to simulate the low frequency surge and sway specific forces. This is achieved by crossfeeding the surge and sway to the pitch and roll channels respectively. The input to these channels is the limited specific force $\underline{f1}$. The Euler input angles are first found using

$$\underline{\beta_{1L}} = \underline{C}\,\underline{f1} \tag{1.13}$$

where $\underline{C}$ is (as described by Reid and Nahon [5])

$$\underline{C} = \begin{bmatrix} 0 & -1/g & 0 \\ 1/g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.14}$$

The components of $\underline{\beta_{IL}}$ are then scaled and passed through the low-pass filters to obtain $\underline{\beta \ddot{L}}$. These filters are second-order and, in the Laplace domain, take the form:

$$\overline{\ddot{\theta}_{LP}} = \overline{\ddot{\theta}_C}\left(\frac{\omega^2_{\theta lp}}{s^2 + 2\zeta_{\theta lp}\omega_{\theta lp} + \omega^2_{\theta lp}}\right) \tag{1.15}$$

$$\overline{\ddot{\phi}_{LP}} = \overline{\ddot{\phi}_C}\left(\frac{\omega^2_{\phi lp}}{s^2 + 2\zeta_{\phi lp}\omega_{\phi lp} + \omega^2_{\phi lp}}\right) \tag{1.16}$$

$$\overline{\ddot{\psi}_{LP}} = 0 \tag{1.17}$$

The output of the filters is limited and then integrated. Following the integration $\underline{\beta \dot{L}}$ is obtained and its components limited. This limiting is conducted to keep the tilt rate below the human perception threshold. A second integration will yield $\underline{\beta SL}$. The total Euler rates and angles are found by summing the high and low pass filter output, yielding:

$$\beta_S = \beta SL + \beta SH \tag{1.18}$$

3.2 Modifications for the PROTEST Washout Algorithm

The washout filter algorithm used for PROTEST, although not exactly the same, more closely resembles that described by Reid and Nahon [5]. Grant [7] provides an excellent explanation of this algorithm as well as the frequency response of the difference systems. Figure 3.2 is a block diagram representation of the washout filters. This section will examine the difference between the two algorithms and identify the implications of these with respect to the implementation of PROTEST which, as is discussed in Chapter 5, required the baseline washout filters to be converted to those originally designed to work with PROTEST. The washout filters below are thus the version ultimately used with PROTEST.

3.2.1 High Pass Specific Force Channel

The primary difference between the two filters is the order. The algorithm used for PROTEST uses third order filters for each of the three specific force high pass channels. In the Laplace domain these are described by:

$$\overline{a_{SI}^{x}} = \overline{a2^{x}}\left(\frac{s^2}{s^2 + 2\zeta_{hpx}\omega_{hpx}s + \omega_{hpx}^2} \cdot \frac{s}{s+\omega_{bx}}\right) \tag{1.19}$$

$$\overline{a_{SI}^{y}} = \overline{a2^{y}}\left(\frac{s^2}{s^2 + 2\zeta_{hpy}\omega_{hpy}s + \omega_{hpy}^2} \cdot \frac{s}{s+\omega_{by}}\right) \tag{1.20}$$

$$\overline{a_{SI}^{y}} = \overline{a2^{y}}\left(\frac{s^2}{s^2 + 2\zeta_{hpy}\omega_{hpy}s + \omega_{hpy}^2} \cdot \frac{s}{s+\omega_{by}}\right) \tag{1.21}$$

The difference in the order of the filters results in the base code lacking a coefficient in the surge and sway channels, ω_{bx} and ω_{by}. In addition, although the form of the second order component is the same, the original code sets the damping coefficients to one, thus eliminating the variability required by PROTEST. The form of the equations above can be transformed into second order systems by setting the ω_b to zero, thus achieve the form of the initial code. This option gives the PROTEST code an additional level of flexibility.

3.2.2 High-Pass Angular Rate Channel

Similarly to the translational channels, the algorithm for PROTEST requires the order of the pitch and roll filters to be augmented by one. The form of the yaw filter remains second order but is changed slightly. The three filters in the Laplace domain, as described by Grant [7], are

$$\overline{\phi_{SH}} = \overline{\dot{\phi}_{H}}\left(\frac{s^2}{s^2 + A_\phi s + B_\phi}\right) \tag{1.22}$$

$$\overline{\theta_{SH}} = \overline{\dot{\theta}_{H}}\left(\frac{s^2}{s^2 + A_\theta s + B_\theta}\right) \tag{1.23}$$

$$\overline{\psi_{SH}} = \overline{\dot{\psi}_{H}}\left(\frac{s^2}{s^2 + A_\psi s + B_\psi}\right) \tag{1.24}$$

where for 1st order filters,

$$A_{\phi,\theta,\psi} = \omega_{\phi,\theta,\psi} \text{ and } B_{\phi,\theta,\psi} = 0 \tag{1.25}$$

and for 2nd order filters,

$$A_{\phi,\theta,\psi} = 2\zeta_{\phi,\theta,\psi}\omega_{\phi,\theta,\psi} \text{ and } B_{\phi,\theta,\psi} = \omega^2_{\phi,\theta,\psi} \qquad (1.26)$$

Again, the initial washout algorithm lacks the coefficient, in this case damping, for the higher order filters in pitch and roll. The initial algorithm uses two first order filters in cascade whereas the algorithm for PROTEST uses a second order high pass filter. A simple equation can be used to equate the two sets of coefficients. As with the translational filters, the scaling block is placed prior to the transformation matrix.

3.2.3 Low-Pass Specific Force Channel

The presence of the scaling block prior to the co-ordinate transformation removes the need to incorporate it in the low-pass branch of the washout filter algorithm. This feature is present in the PROTEST version of the code, whereas the original code uses a different scaling block in each of the high and low pass filters. The result is the elimination of two variables.

The PROTEST code places the tilt-coordination block after the low pass filter whereas the original code has it previous to this block. The original code also introduces an added integration block that, in the PROTEST algorithm, had been included into the filter block.

3.3 Coefficients

The importance of the washout filter algorithm differences resides primarily in their impact on the set of coefficients. Because PROTEST's goal is to tune the washout algorithm, three choices are available. The first option is to change the washout algorithm used in the simulator code. The second option is to alter PROTEST's reasoning and logic rules to reflect the different frequency response. The third choice is to develop an equivalency system between the two washout filter algorithms. The second option involves an in depth analysis of the frequency response and the results to be integrated into the PROTEST code. In addition to being time consuming and clearly beyond the scope of this work, it demands a deep knowledge of the PROLOG language. The last option would allow the washout filter code to remain in its base state, which ensures its fidelity since it is already

known to work. However, the PROTEST washout filter essentially provides a higher level of complexity than the base code and can, by setting certain variables to zero provide the same response.

It was thus decided to replace the baseline simulator code with one equivalent to that designed for PROTEST. This option presented fewer difficulties and promised exact compatibility with the PROTEST code. The aforementioned decision does engender the need for comprehensive testing of the washout filters to ensure their proper functioning. This testing will be the topic of Chapter 6. Table 3.1 gives the list of coefficients used in both the baseline and PROTEST washout algorithms. The baseline damping coefficients, although present in the baseline theory, are implicitly set to one in the code. As well, an equivalence equation is given for the high pass yaw coefficients to reflect the slightly different nature of the filter equation in the Laplace domain.

Table 3.1 Coefficient Equivalencies between the baseline and PROTEST washout filters

	PROTEST		Baseline
	COEFFICIENT NUM.	VARIABLE	Variable
ROLL	1	$\omega_{hp\phi}$	$\omega_{hp\phi}$
	2	$\zeta_{hp\phi}$	1
Pitch	3	$\omega_{hp\theta}$	$\omega_{hp\theta}$
	4	$\zeta_{hp\theta}$	1
Yaw	5	$\omega_{hp\psi}$	$\sqrt{\omega_{hp1\psi}\omega_{hp2\psi}}$
	6	$\zeta_{hp\psi}$	$\frac{\omega_{hp1\psi}+\omega_{hp2\psi}}{2\sqrt{\omega_{hp1\psi}\omega_{hp2\psi}}}$
Surge	7	ω_{hpx}	ω_{hpx}
	8	ζ_{hpx}	1
	9	ω_{bx}	n/a
Pitch Low pass	10	ω_{lpx}	$\omega_{lp\theta}$
	11	ζ_{lpx}	$\zeta_{lp\theta}$
Sway	12	ω_{hpy}	ω_{hpy}
	13	ζ_{hpy}	1
	14	ω_{by}	n/a
Roll Low Pass	15	ω_{lpy}	$\omega_{lp\phi}$
	16	ζ_{lpy}	$\zeta_{lp\phi}$
Heave	17	ω_{hpz}	ω_{hpz1}
	18	ζ_{hpz}	ζ_{hpz}
	19	ω_{bz}	ω_{hpz2}
Gains	20	k_x	ghpx
	21	k_y	ghpy
	22	k_z	ghpz
	23	k_p	gphihp
	24	k_q	gthehp
	25	k_r	gpsihp
Limiters	26	flxmax	thedmax
	27	flymax	phidmax

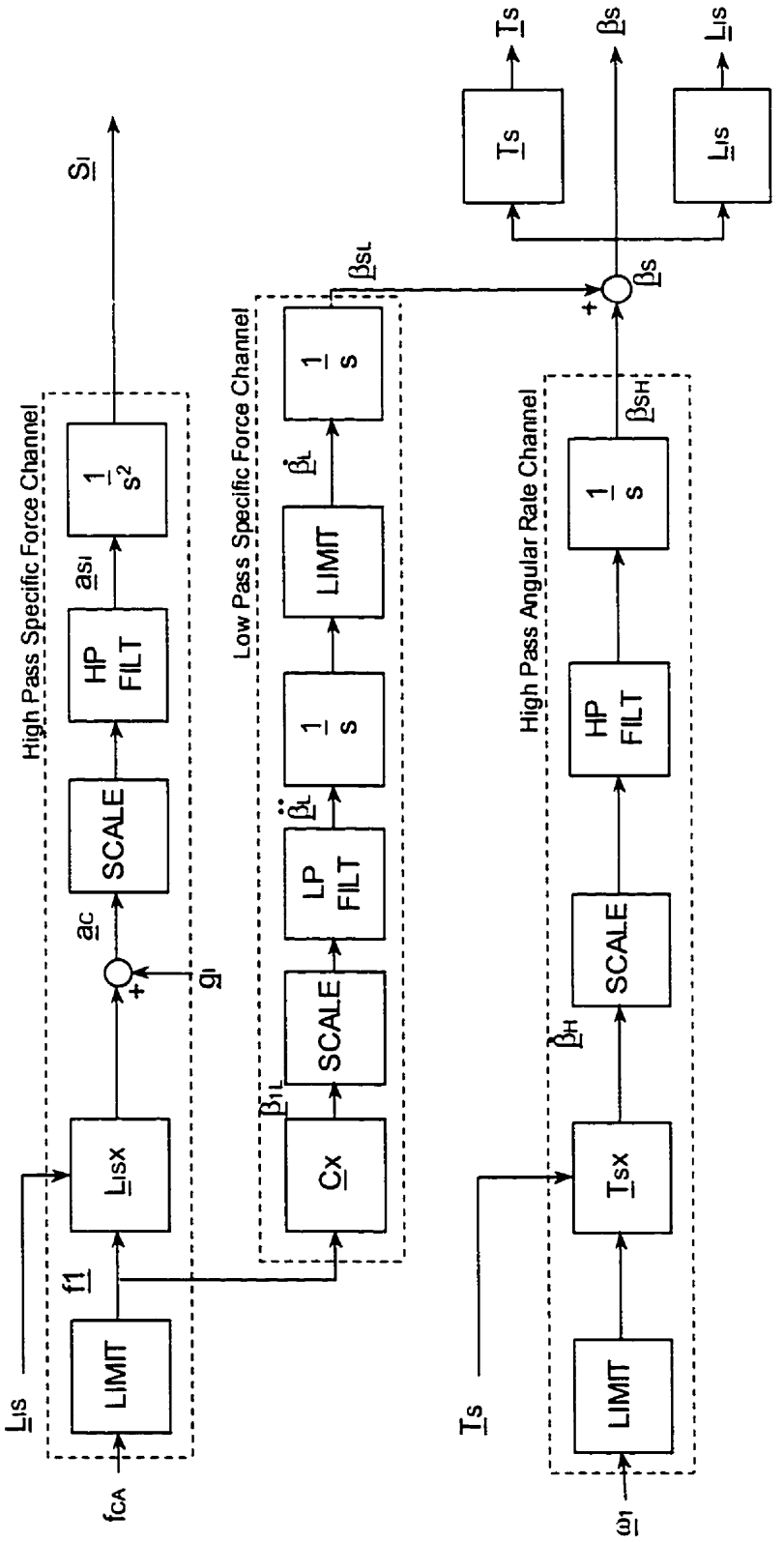

Figure 3.1 Baseline UTIAS Classical Washout Algorithm

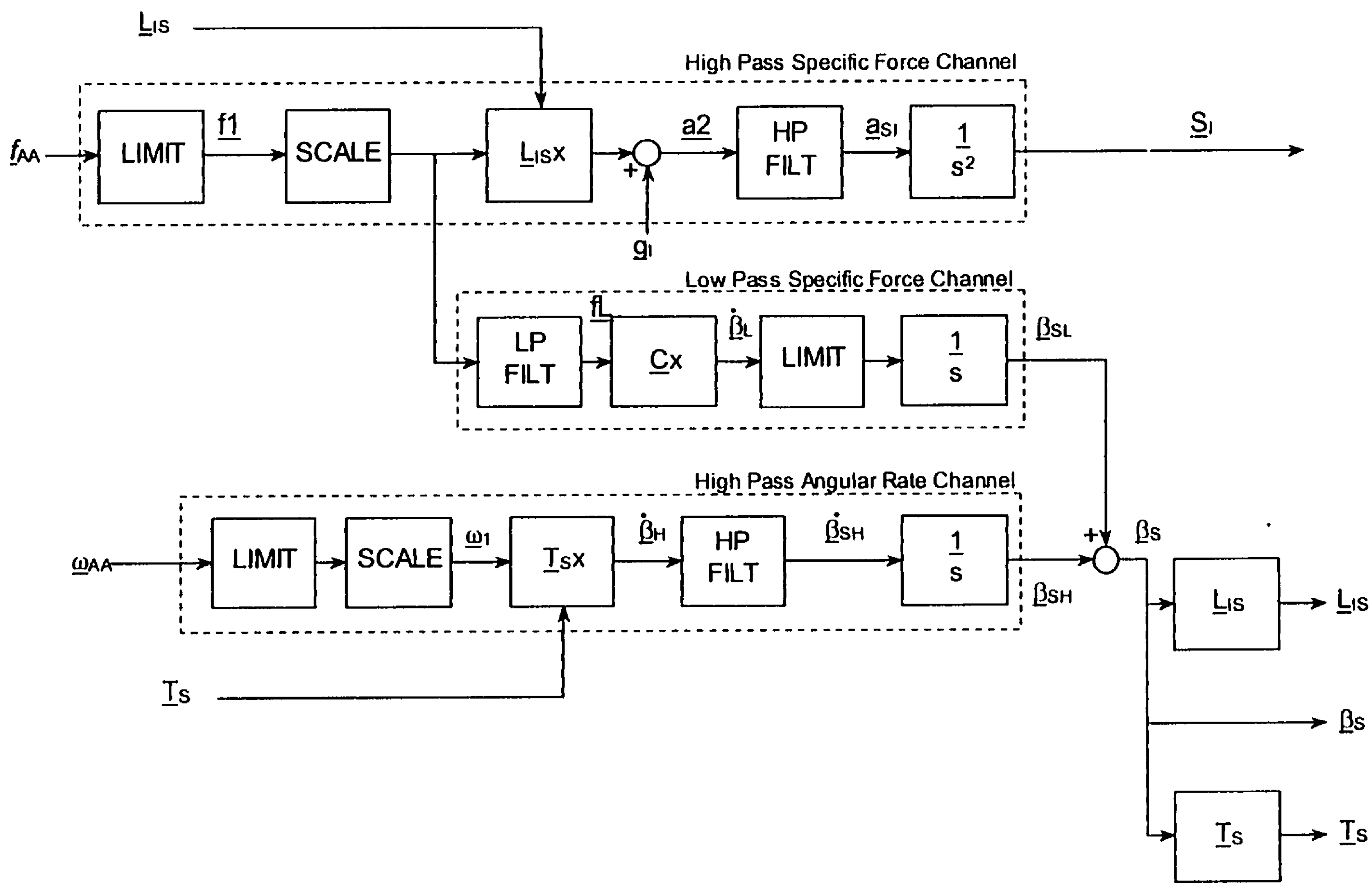

Figure 3.2 UTIAS Classical Washout Algorithm used for PROTEST

CHAPTER 4

PROTEST

4.1 Expert Systems

The goal of an expert system, as described by Walker et al [8], is to have the problem solving capabilities of an expert or, at the very least, a high level of performance. This statement as it applies to PROTEST suggests that the expert system tuning performance should match that of an expert tuner. In other words the final motion characteristics obtained from using PROTEST should be as satisfactory, if not more so, than that obtained from regular tuning strategies.

The expert system has the unique ability to reach the above goal by basing its logic, to a certain degree, on the knowledge captured from an expert tuner. This information is stored in a knowledge base of rules and facts. In the case of PROTEST this knowledge base not only encompasses the empirical knowledge from the expert tuner, but also includes analytically based rules and facts. Together these form a solid basis for tuning the washout filters. Grant presents the complete development of PROTEST's reasoning. [7]

4.1.1 *Tasks for Tuning*

Although an expert system can be applied to a vast number of tasks, PROTEST applies only four distinct tasks: diagnosis, repair, control, and monitoring. The diagnosis algorithm is the first step in tuning. It is used to analyse data obtained from a test flight, in combination with pilot input, to hypothesise the correct coefficient to adjust. In the second step, repair, PROTEST determines the magnitude of the adjustment to fix the problem while remaining within the specified limits of the simulator. The closed loop nature of the tuning process (by virtue of pilot input) allows the repair sequence to be equated to a control task. The final task is monitoring. This identifies when tuning should end, when to try a secondary coefficient or cycle, and similar such decisions.

A detailed description of each task, how they are encoded, and efficiency issues are presented by Grant [7].

4.2 Structure of PROTEST

The Expert System requires three software components to function. The first is the code for PROTEST written in PROLOG. It includes all the rules, logic, and user interface described above. The second component is a program called ANALYZE. It is used to calculate the required statistics of each run and pass its results to PROTEST. The last component is the simulator code whose purpose it is to run the flight equations and calculate the simulator motion properties via the washout filters, as discussed in previous chapters.

Figure 4.1 shows the original interconnections of the three components as designed by Grant [7]. To adapt PROTEST to the current computing environment of the UTIAS Flight Simulator Laboratory it will be necessary to alter the connection pathways between the different components as well as adapt the components themselves.

It is also important to acknowledge the presence of the pilot, whose input needs must not be forgotten. Any interface changes must provide the user with equivalent or added control in the functioning of the Expert System. This topic will be discussed in Chapter 5, when the adaptation scheme is presented. The current section will discuss the primary needs and functions of each of the components required by the Expert System.

4.2.1 PROTEST Algorithm

The PROTEST algorithm component holds the logic and reasoning tools the Expert System needs to function. As discussed at the beginning of this Chapter this code tries to capture in its rules the thought process and procedural aspects of a professional tuning expert.

It is not the mandate of this thesis to alter the PROLOG code in anyway since this would require knowledge of the PROLOG language. A basic understanding was gained, however, in order to comprehend the logic and input/output modules. Figure 4.2 presents the lowest level of complexity for the functioning of PROTEST. Figures 4.2 to 4.7 give five progressively deeper levels of code description with subsequent figures describing the bolded boxes of the previous figure. In these flowcharts, ellipses indicate the entry and exit points, diamonds represent decision points, circles indicate points of input from the evaluation pilot, and finally dotted circles outside a diamond indicate a decision point based solely on 'raw' information supplied by the pilot (such as a yes or no answer). These figures give the user a quick reference guide to the procedure followed by PROTEST during tuning, and should be understood before any tests are run. Identified in the figures are the locations of pilot required input and the decision blocks employed by PROTEST. They also show the termination locations when an error has occurred. An important aspect of PROTEST, of which the user should be aware, is its decision procedure when limiting has occurred. In this situation, PROTEST first tunes to correct the limiting and, only once this has been rectified, does it ask for pilot input. This is shown in Figure 4.3, where one can see that there are two main branches present: one if limiting has occurred and one where it has not. During a tuning run the option is given for PROTEST to echo its reasoning procedure. A sample of such output is presented in Appendix B. Using this output and the flowcharts one can determine the steps taken by PROTEST to tune the simulator.

Appendix C provides a brief user manual for PROTEST. In a general sense PROTEST begins when the initial set of coefficients are sent from ANALYZE. It then goes into 'wait' mode while the evaluation pilot completes a run after which it receives statistical data from ANALYZE. If no limiting has occurred the pilot is prompted to identify the PROBLEM using a list built into PROTEST. Following this PROTEST generates a corrective set of

coefficients that are sent back to ANALYZE. All sets of coefficients are updated and another run is flown. Appendix A gives a summary of the general behaviour of PROTEST for each given PROBLEM or multiple PROBLEMS. Although exact adjustments are not given, this table is useful in predicting (and testing) the response of PROTEST.

The inputs to PROTEST are first the initial set of coefficients and, after each subsequent run, the statistics calculated by ANALYZE. The exact nature of these statistics is discussed in Section 4.2.2. It is important to note the format required by PROTEST for incoming data. This necessitates that the data to be sent as one long data string with each item separated by a colon and each group of items by a semicolon. This will be discussed further within the context of ANALYZE.

4.2.2 ANALYZE

The purpose of ANALYZE is to calculate the statistical properties of each run as required by PROTEST. These serve to give PROTEST the information it uses to perform the tuning tasks. Of importance are the absolute displacement of the simulator in each degree of freedom, the intensity of the motion on each degree of freedom, the severity of any limiting, and the importance of each coefficient to a degree of freedom. The mathematical derivations of these equations are the subject of Appendix D in Grant's thesis.

Figure 4.1 shows the relationships ANALYZE has with the other components of the Expert System. For adaptation into the new computing environment this relationship scheme had to be altered. This process is discussed in Chapter 5. This section describes the different computing tasks assigned to ANALYZE, and how they are fitted into code.

Figure 4.8 is a flowchart representation of the original version of ANALYZE. Not shown in this flowchart is the flag variable assignment that came as a result of options set when the executable is run. This allowed the user to identify, when starting ANALYZE, what the input and output conditions would be. For example, one such flag set the source for input, either live or from a data file. These flags will not be necessary in the adapted version of the code since these options are prompted to the user within the run. The first step shown in the flow chart is then the initialisation of a counter flag. What ANALYZE does is determined to a large extent by the variable IANLZE whose value is received from

the host at the same time as the specific force and angular rate inputs. This flag, in conjunction with the IANLZE_OLD variable, guides ANALYZE through its different stages: first-run initialisation, regular run progression, and termination. Next, ANALYZE sets up the ethernet socket ports that will enable communication between computers. It then receives the initial set of coefficients from the host and initialises the filters. To calculate the coefficient effectiveness the washout filters are run 28 times. Once to obtain the real response and 27 times with a small change in each coefficient. In the original code this was achieved by calling the subroutine partial_ev, the structure of which is shown in Figure 4.9. Limit flags are used to determine if any limiting has occurred and the total time spent at the limits. The motion statistics are then calculated via calls to two subroutines: *p_stats* and *s_stats*. At the end of a run, collected data is normalised and sorted. This is achieved by calling a subroutine called *pturb*. The data is formatted for output to both a file and communication to the PROTEST modules. The program is put on hold while PROTEST calculates the new coefficient set that is returned to ANALYZE. After updating its coefficient set ANALYZE forwards the data to the host. The option for another run is given and ANALYZE is either run again or the sockets are closed and ANALYZE terminates.

ANALYZE can thus be divided into two parts: the real and non-real time components. Of relevance to the adaptation procedure are those components in each part and their communication requirements. Specifically, it is important to be aware of the statistical variables that must be passed to PROTEST since it does not only use the numerical values generated by ANALYZE. Instead, ANALYZE produces arrays of characters with matching indices. One such array describes the relative displacement, defined as the ratio of the maximum displacement of a degree of freedom to the sum of the displacements of all degrees of freedom. This will constitute an array with the names of each degree of freedom and an integer array identifying the order of displacement of each degree of freedom. Together the two arrays list the degrees of freedom from the one having had the largest displacement to that with the smallest. The same method is used to produce the ordered list of degrees of freedom for the sensed motion. For each degree of freedom a character array is also generated listing, in order of importance to a particular degree of freedom, the name of each coefficient. Following these lists are the actual displacement

and partial derivative numerical values. In both the previous cases only those coefficients used in the calculations are included. If, for example a second order filter is used for surge, then the ω_{bx} is not included in the lists. Finally, a character string is transmitted to PROTEST indicating if limiting has occurred and a numerical value representing the duration of that limiting

4.3 Ethernet

The communication between computers is achieved using ethernet sockets. These are opened and closed using C coded algorithms. The most important of these is the 'send_es.c' module that sends data to the Expert System. Table 4.1 gives the list of variables that are sent with a description of each. This module, in addition to sending the data, formats it in the form to be read by PROTEST. The form requires the data to be sent as a single text string containing all the information. In this text string a colon separates each array item while a semicolon separates every array. This is the format the expert system is designed to receive and produces an error if the format does not conform.

Table 4.1 Statistical variables passed from ANALYZE to PROTEST

Variable name	Description
send_msgsock	ethernet variable
cdisp	displacement arrays
idisp	
csensed	sensed motion arrays
isrms	
partial	coefficient list of partial derivatives
limit_string	character string indicating limiting
disp	actual displacement numerical values
ii	partial derivative counter
time_limit	time at stops
fnrms	normalized numerical values of specific force
wnrms	and angular rate partials derivatives

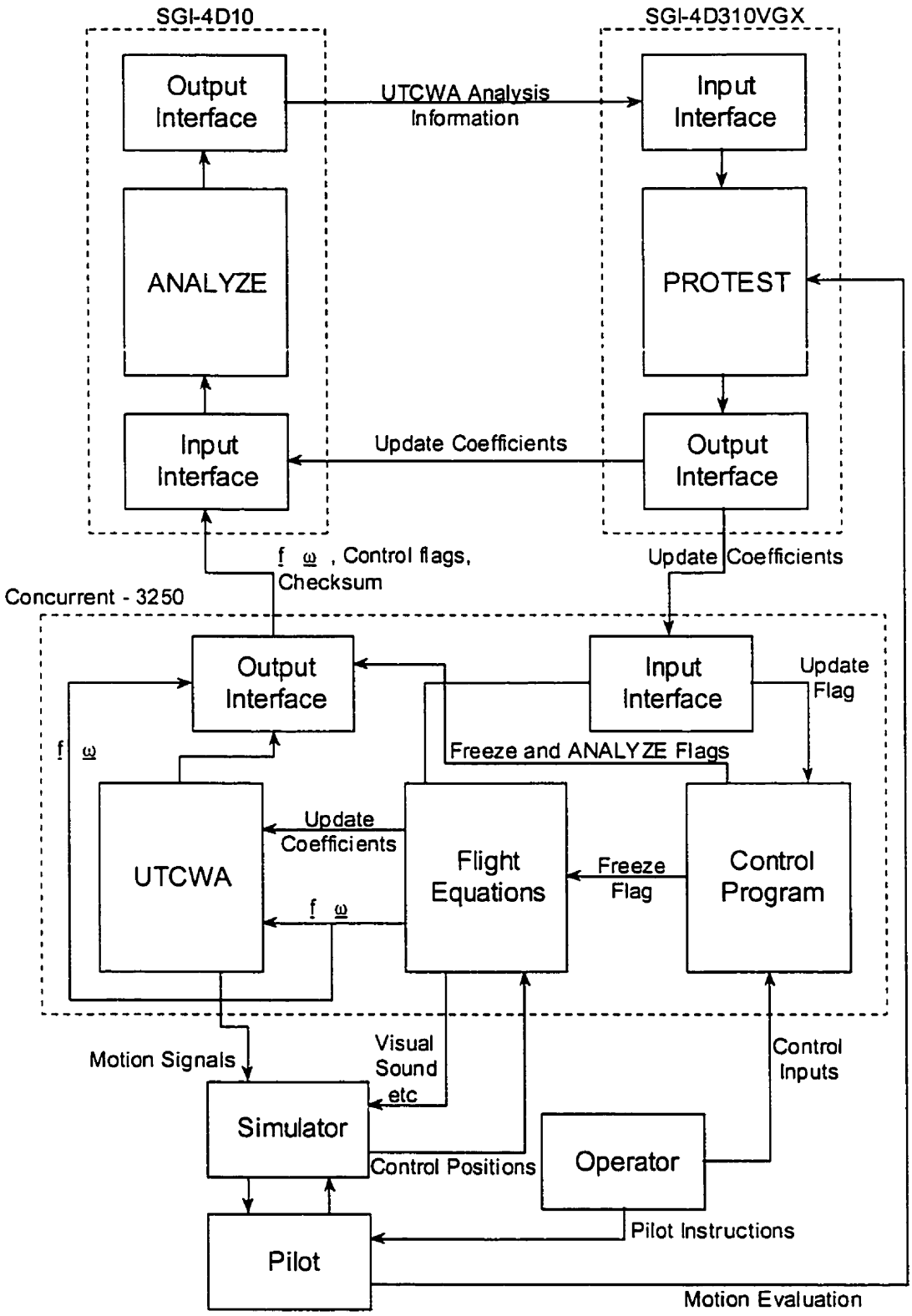

Figure 4.1 Distributed Computing Resources [7]

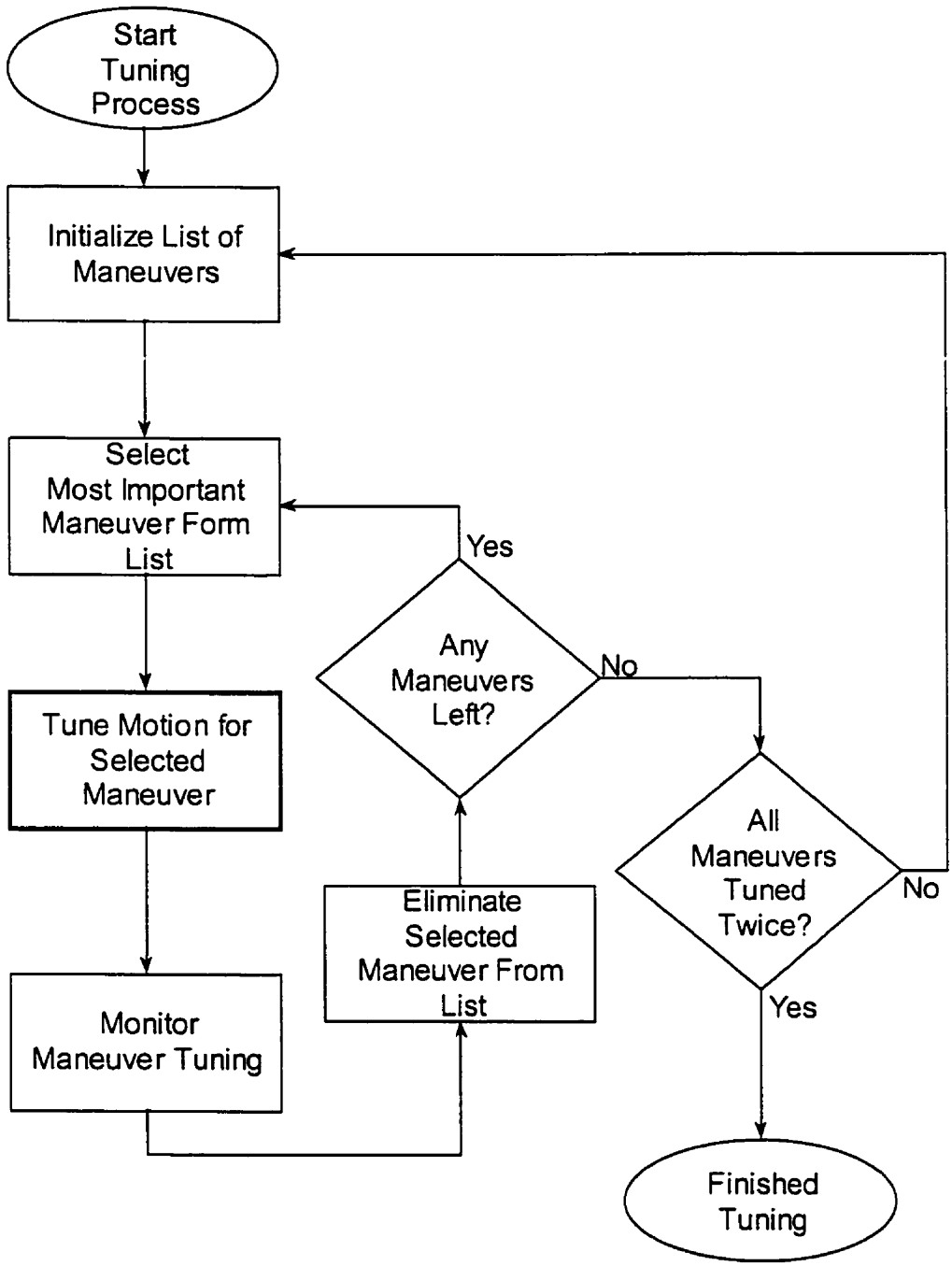

Figure 4.2 Tuning All Manoeuvres [7]

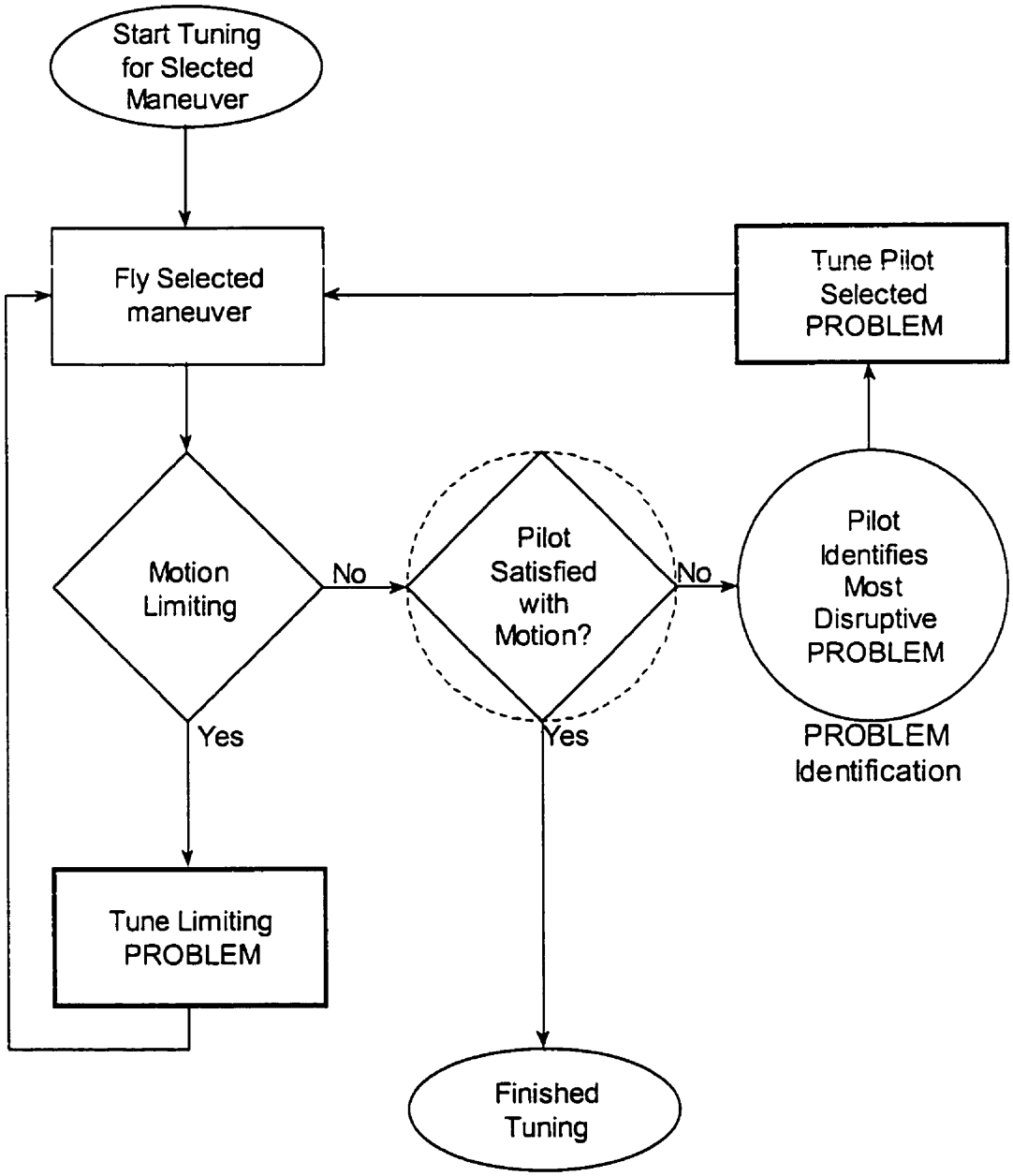

Figure 4.3 Tune Motion for Selected Manoeuvre [7]

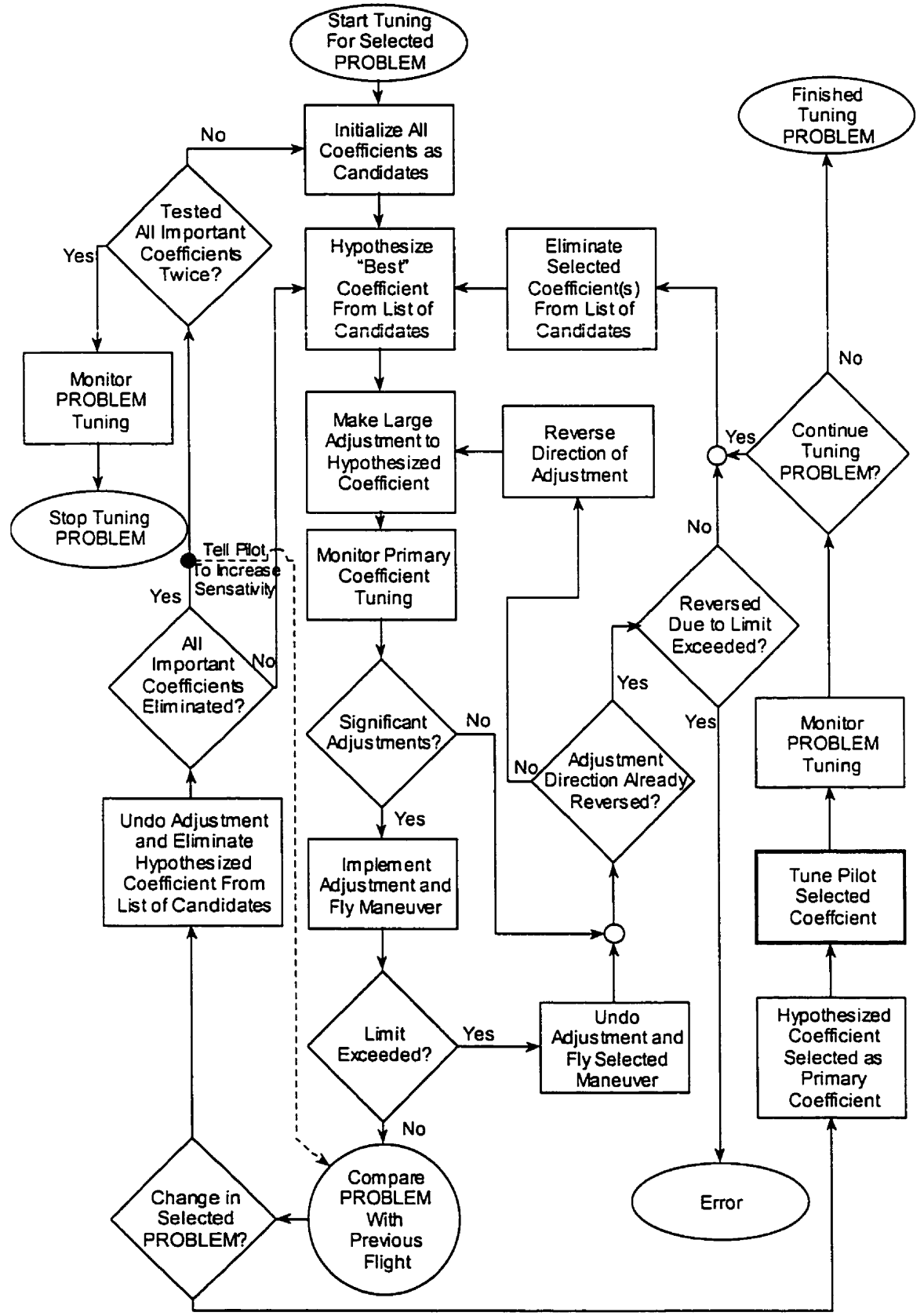

Figure 4.4 Tune Pilot Selected PROBLEM [7]

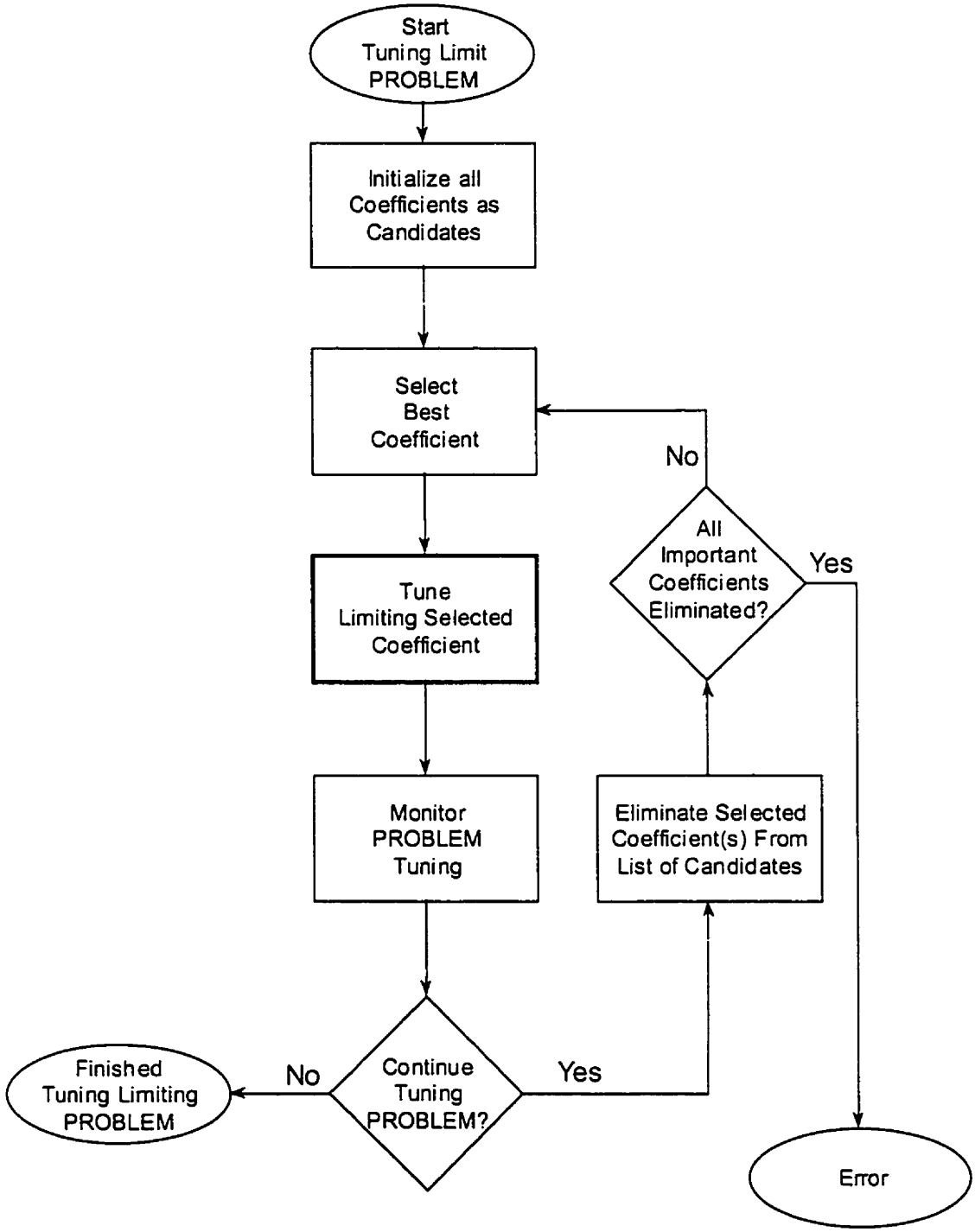

Figure 4.5 Tune Limiting PROBLEM [7]

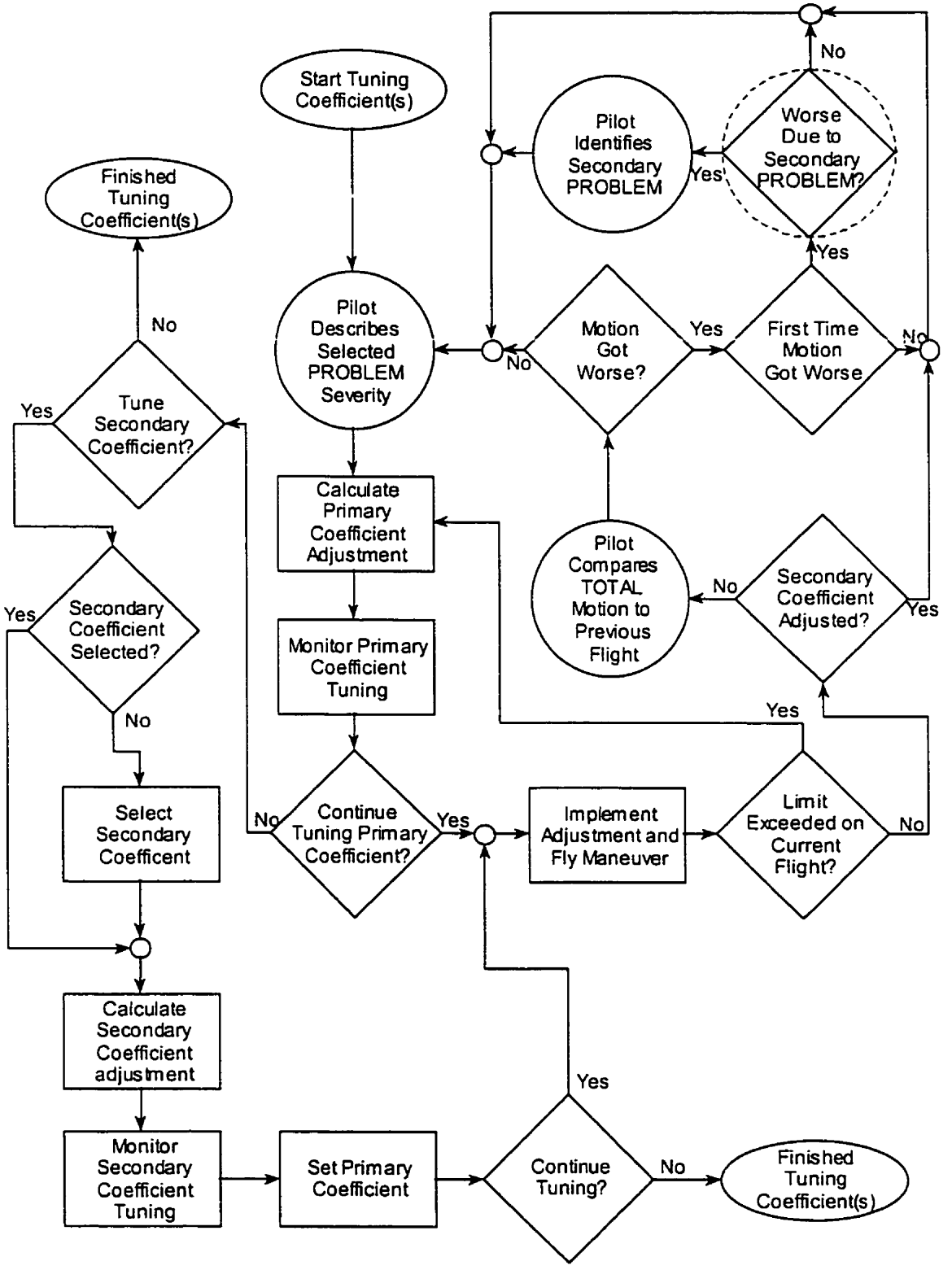

Figure 4.6 Tune Pilot Selected Coefficient [7]

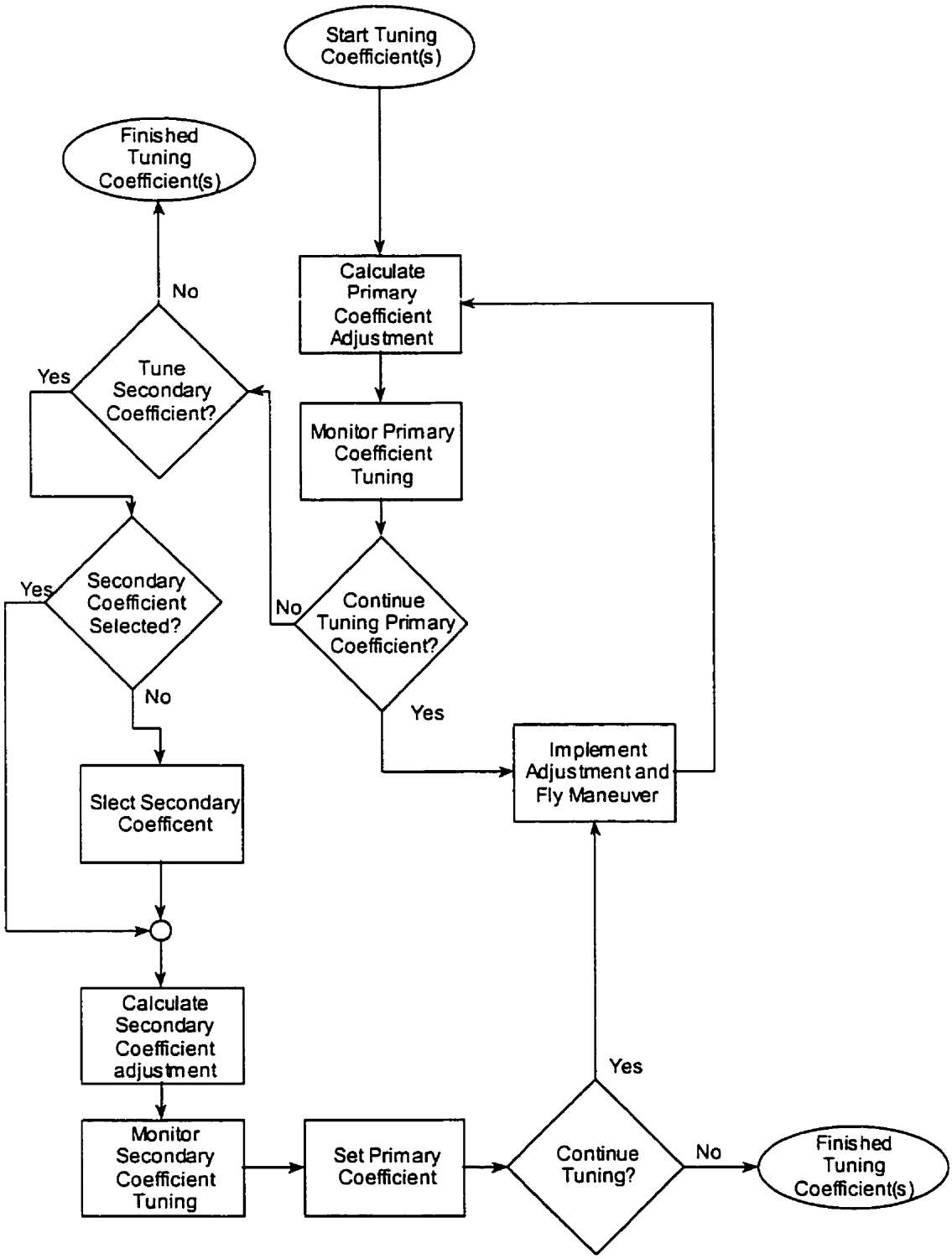

Figure 4.7 Tune Limiting Selected Coefficient [7]

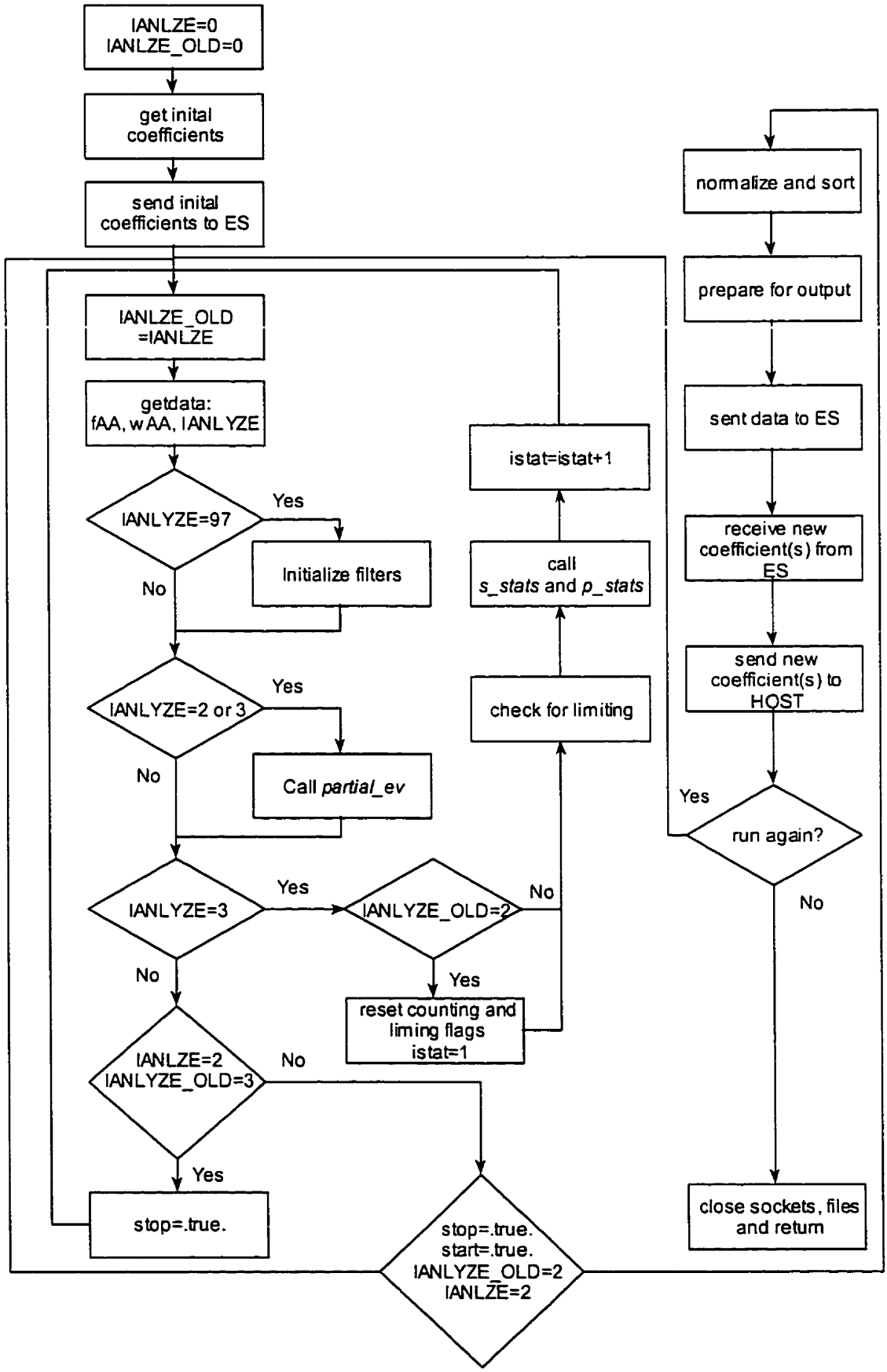

Figure 4.8 ANALYZE Flowchart

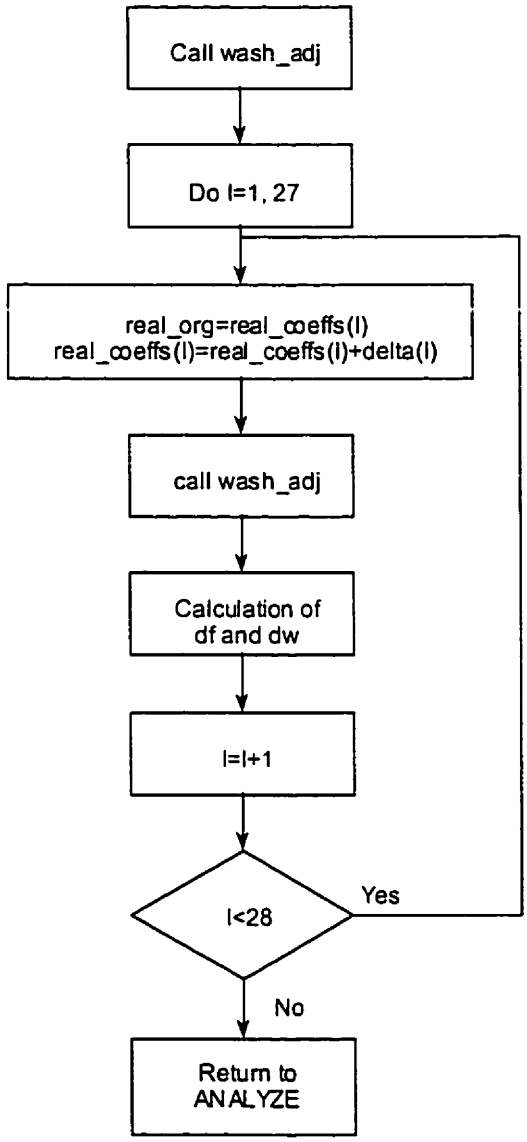

Figure 4.9 Flowchart of partial_ev code

CHAPTER 5

ADAPTATION

This chapter describes the way various components of PROTEST and the simulator code were modified so that they could work together. The previous chapters identified the many components of each system and have given a few glimpses into some of the changes that are required. The steps involved in obtaining a fully functional system are the topic of this chapter. A top down presentation method will be used, starting from the more global changes and working down to specific modules. This technique was the one used in performing the modifications.

As can be seen from Figure 4.1, PROTEST originally ran on three computers: one for the PROTEST code, one for ANALYZE, and one for the simulator code. This division of code was largely due to the lack of computing power of the host computer. The requirement of ANALYZE to run the washout algorithm 28 times per iteration being too demanding for the host computer. The result was a need for multiple checksums and flags to ensure synchronisation of the host washout calculations with those of ANALYZE. The current hardware, however, is powerful enough to run both the simulator software and ANALYZE simultaneously. Thus the ethernet communication requirements between ANALYZE and the

host were eliminated. The added advantage to this configuration is the removal of real-time ethernet communication previously required for PROTEST.

The consequence of the above decision was the integration of ANALYZE into the host code. This task comprises the bulk of the adaptation requirements. This integration requires ANALYZE to be divided into the real and non-real time components. The module called ANALYZE in the adapted configuration now refers only to the non-real time sorting, normalisation and communication of data. Figure 5.1 gives a breakdown of the modules that were called from ANALYZE in the original configuration while Figure 5.2 identifies how these have been divided in the new code. As was discussed earlier, FLGHTEQ resides as a module in the real-time code and thus will house ANANLYZE's real-time components while NRT will call the non-real time items.

5.1 Non-Real Time

The non-real time module NRT is used to provide the user interface, initial conditions set up, and post-run control. The baseline code contains a number of run options that determine flight conditions. These initial conditions do not impact PROTEST but must be consistent between runs to allow the pilot to focus on the changes due to coefficient manipulation by PROTEST. Added to the pre-run set up is the option of whether or not PROTEST will be run. This flag (variable IPROON) determines if the ethernet communication is initialised or not.

During the pre-run set up the SETWASH module is called. This module reads in the initial values of the coefficients. As is discussed in Section 3.3 the number of coefficients has changed from the baseline code. As a consequence the SETWASH file has been modified. Figure 5.3 gives a sample of the input file format to be used. SETWASH prompts for input of the initial washout file name on the first run but will use the data from a previous run for every subsequent run. If PROTEST is to be run (IPROON=1) ANALYZE is called to initiate ethernet communication with PROTEST by opening sockets.

Run control gives the user the opportunity to change flight characteristics during a run. The option is thus given to start and stop the collection of data for PROTEST. This flag determines if the washout filters are run once per simulator cycle or twenty eight

times. In this manner data collection can be started when conditions are stabilised and the pilot is ready, and stopped when the appropriate amount of data has been collected. The flag which is toggled is IRANAL.

In the post run ANALYZE is called again using an entry point. In this part of ANALYZE the obtained data is sorted and normalised, formatted for output, and written to a file. The option is then given to keep the flight results. If they are rejected the user is asked if another run should be made. If the results are accepted the data is sent to PROTEST which then returns the new coefficient(s). Appendix C provides a user manual for PROTEST. At this point the option is now to continue tuning. If the answer is 'no' the sockets are closed and the current set of coefficients are written to a user defined file. Otherwise the current set of coefficients are written to the default file (final_coeff.dat) for use by the next run. This file is written in the correct format to be read by SETWASH.

5.2 Real-time

The change to FLIGHTEQ involves the addition of calls to p_stats and s_stats. These calls are contingent on the value of IRANAL, which is a flag set by the user during the run to turn data collection on and off. Following the two subroutine calls is an increase by 1 of the ANALYZE counter. This value is used by ANALYZE in its calculations.

The bulk of the real-time adaptation is found in the WASH module, where a number of ANALYZE's components have been moved. Figure 5.4 shows how these components have been included in the WASH module. The box containing the term 'WASHOUT' represents the washout filter calculations as described by Chapter 3. The IRANAL flag is used again to establish the number of times the code will be run: once if data collection is not running and 28 times otherwise. In the former case the washout filter will run normally, bypassing all the ANALYZE components. In the latter case the first step is to initialise variables and arrays. In both cases the LHS array is initialised with the washout filter intermediary variables so that each subsequent partial derivative run uses the correct data from its previous time step. Then the first run is completed that calculates the true filter outputs and jack extensions. During this first run a number of variables needed by ANALYZE are

also stored. These include the inertial displacement of the simulator reference point for each degree of freedom, the total angular commands for each degree of freedom and the output simulator specific forces and angular rates. As well, any time spent against the limits is added to a running sum. Each subsequent run will change one coefficient by a value 'delta' and run through the washout filters again. At the end of these runs the difference values df and dw, which are the differential specific force and angular rate respectively, are calculated for the given coefficient and each degree of freedom. Control is then returned to FLGHTEQ.

5.3 CDB and Include Files

For variables to be shared amongst modules they must be entered into the CDB (common data block). For ease of calling, these variables are generally grouped, with the first two characters being common to the group followed by an underscore. Following this pattern the 'ES_' group was created for the Expert System variables. They are predominantly those variables used by ANALYZE. The coefficients were also added to the CDB in the MO_ group. A number of flags have also been added. Table 5.1 provides the list of all variables added to the CDB.

Two include files were added to the code; es_equiv.inc and real_coeffs.inc. They are both equivalence files and are given in Figure 5.5 and Figure 5.6.

5.4 PROTEST

The only modifications made to the PROTEST code are in the communication modules, where the computer name where ANALYZE resides had to be changed and in the manoeuvres list. The former is due to the incorporation of ANALYZE in the host computer and the change of the host name since the original development of PROTEST. In addition a line was added at both ends of the communication to write and read the numbers of bytes sent to ensure complete reception of a given message.

Since PROTEST was originally configured to tune a large transport fixed wing aircraft the manoeuvre list needed to be changed to accommodate a set better suited to helicopters. This task was achieved by first replacing the manoeuvre names in the

'maneuver_importance.pl' and 'tune_ss_couple.pl' files. The second of these files requires changing only the list presented to the user when manoeuvre selection is made. The 'maneuvre_importance.pl' files however also include the manoeuvre importance descriptor. Following Grant's [7] instructions, manoeuvre importance was determined and entered in this file. The 'maneuvre_importance.pl' files are listed in Figure 5.7.

Table 5.1 Common Data Block Variables added for PROTEST

Variable Description		Variable Name
Washout filter Coefficients	High Pass Roll	MO_WPHIHP
		MO_ZPHIHP
	High Pass Pitch	MO_WTHEHP
		MO_ZTHEHP
	High Pass Yaw	MO_WPSIHP
		MO_ZPSIHP
	High Pass Surge	MO_WXHP
		MO_ZXHP
		MO_WBXHP
	Low Pass Pitch	MO_WTHELP
		MO_ZTHELP
	High Pass Sway	MO_WYHP
		MO_ZYHP
		MO_WBYHP
	Low Pass Roll	MO_WPHILP
		MO_ZPHILP
	High Pass Heave	MO_WZHP
		MO_ZZHP
		MO_WBZHP
	Gains	MO_GXHP
		MO_GYHP
		MO_GZHP
		MO_GPHIHP
		MO_GTHEHP
		MO_GPSIHP
	Tilt co-ordination Limits	MO_FLXMAX
		MO_FLYMAX
	True output specific forces and angular rates	MO_WS_ORG
		MO_FS_ORG
Statistical Information for ANALYZE	Displacements	ES_TRANS
		ES_ANGL
	partial derivatives	ES_DF
		ES_DW
	Sensitivity root mean square values	ES_FSRMS
		ES_WSRMS
	Max displacements	ES_P_TRANS
		ES_P_ANGL
	Specific forces and angular rates root mean square	ES_FRMS
		ES_WRMS
	Coefficient array	ES_R_COEFF
	Stats. on/off flag	ES_IRANAL
	Stats. counter	ES_ICNT

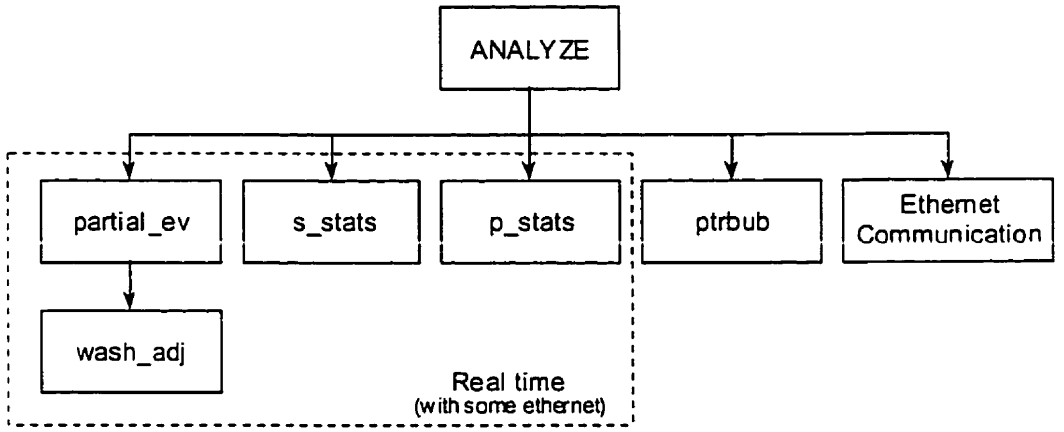

Figure 5.1 Original ANALYZE hierarchy

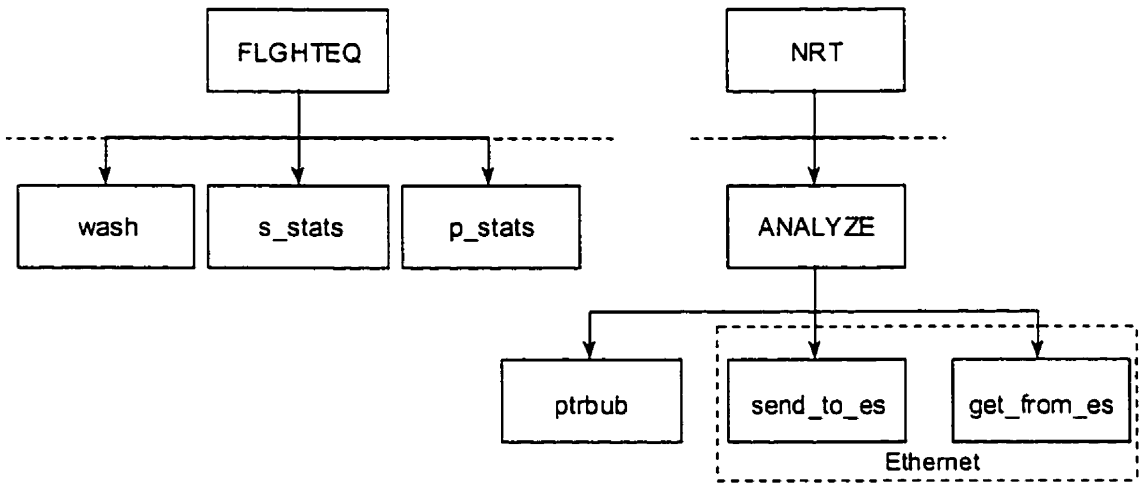

Figure 5.2 Implementation of ANALYZE components

```
*************************************************************
**                                                         **
**                    WASHC.DAT.2                          **
**          ROTATION @ SIMULATOR CENTROID                  **
**                                                         **
*************************************************************

-------------------------------------------
      HEADER
-------------------------------------------
   2          ; NWASHC extension/version #

-------------------------------------------
      SURGE-PITCH
-------------------------------------------
   0.5        ; GXHP
   2.3328     ; WXHP
   1.0        ; ZXHP
   0.         ; WBXHP
   0.5        ; GTHEHP
   0.978      ; WTHEHP
   0.         ; ZTHEHP
   0.5        ; GTHELP
   5.832      ; WTHELP
   1.0        ; ZTHELP

-------------------------------------------
      SWAY-ROLL
-------------------------------------------
    .5        ; GYHP
   5.832      ; WYHP
   1.         ; ZYHP
   0.         ; WBYHP
   0.5        ; GPHIHP
   7.824      ; WPHIHP
   0.         ; ZPHIHP
   0.5        ; GPHILP
   2.916      ; WPHILP
   1.         ; ZPHILP

-------------------------------------------
      HEAVE
-------------------------------------------
   1.25       ; GZHP
   3.912      ; WZHP
   1.         ; ZZHP
   0.1956     ; WBZHP
```

Figure 5.3 Format of Coefficient Input File Format

```
-----------------------------------------
      YAW
-----------------------------------------
    .75       ; GPSIHP
   0.1966     ; WPSIHP
   2.5884     ; ZPSIHP

-----------------------------------------
      LIMITERS
-----------------------------------------
  15.2439     ; FXMAX
  15.2439     ; FYMAX
  15.2439     ; FZMAX
   0.6108     ; PMAX
   0.6108     ; QMAX
   0.6108     ; RMAX
   0.228      ; FLXMAX
    .5137     ; FLYMAX

-----------------------------------------
      ACTUATORS
-----------------------------------------
 200          ; NFADIN
   3.14       ; WFADOUT          (rps)
    .4        ; LILIM            (m)
    .63       ; LIDLIM           (m/sec)
   5.         ; LIDDLIM          (m/sec2)

-----------------------------------------
      PRINTOUT
-----------------------------------------
  10          ; IPRINTF
  10          ; IPRINTR
  10          ; IPRINTD
  10          ; IPRINTDD
  10          ; IPRINTP
  10          ; IPRINTV
  10          ; IPRINTA
```

Figure 5.3 Format of Coefficient Input File Format cont'd

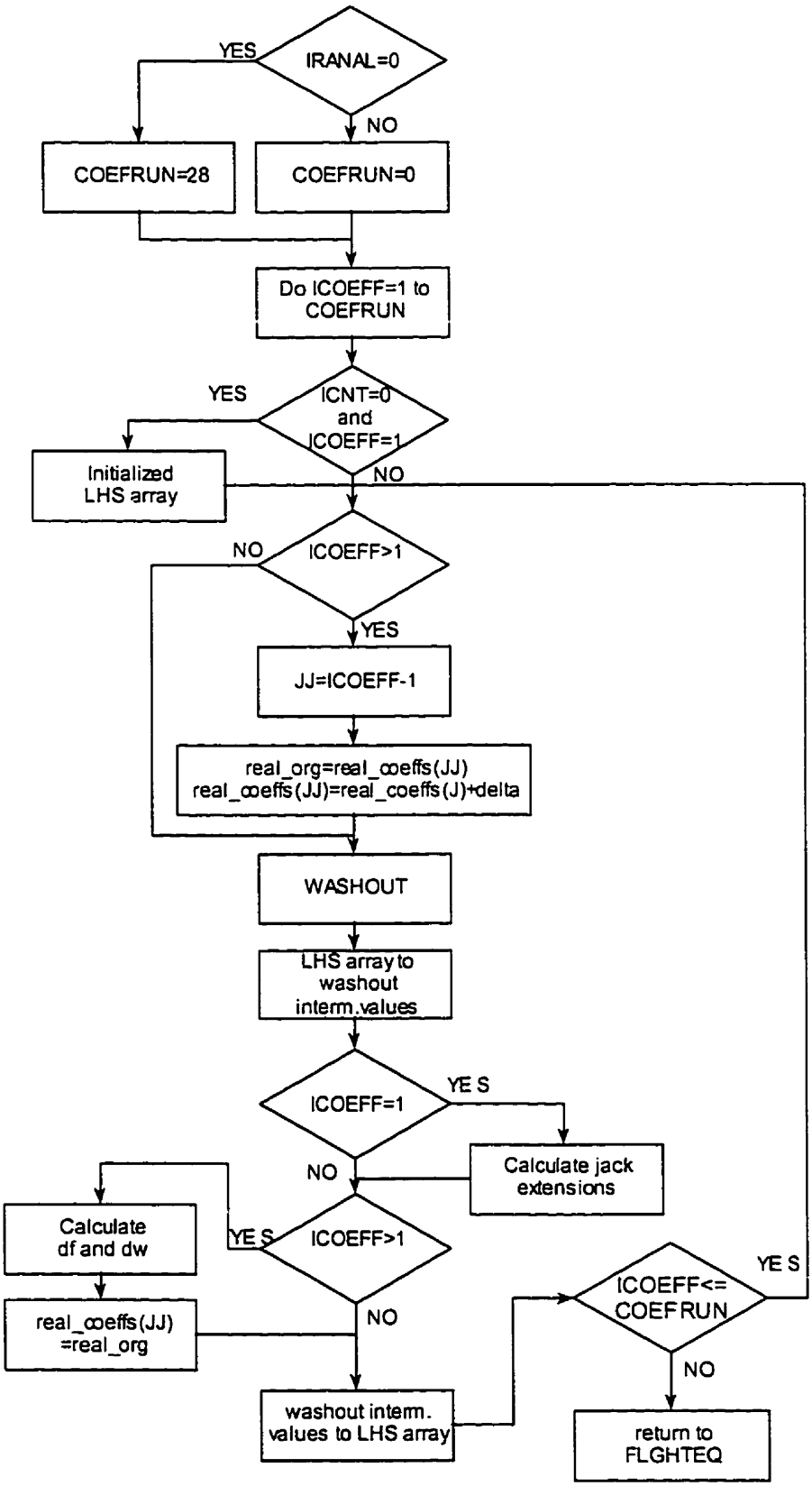

Figure 5.4 Washout Algorithm with integrated ANALYZE components

```
REAL*4 REAL_COEFFS(27)
C      EQUIVALENCE
C      -----------
       EQUIVALENCE (real_coeffs(1),MO_WPHIHP),
      &            (real_coeffs(2),MO_ZPHIHP),
      &            (real_coeffs(3),MO_WTHEHP),
      &            (real_coeffs(4),MO_ZTHEHP),
      &            (real_coeffs(5),MO_WPSIHP),
      &            (real_coeffs(6),MO_ZPSIHP),
      &            (real_coeffs(7),MO_WXHP),
      &            (real_coeffs(8),MO_ZXHP),
      &            (real_coeffs(9),MO_WBXHP),
      &            (real_coeffs(10),MO_WTHELP),
      &            (real_coeffs(11),MO_ZTHELP),
      &            (real_coeffs(12),MO_WYHP),
      &            (real_coeffs(13),MO_ZYHP),
      &            (real_coeffs(14),MO_WBYHP),
      &            (real_coeffs(15),MO_WPHILP),
      &            (real_coeffs(16),MO_ZPHILP),
      &            (real_coeffs(17),MO_WZHP),
      &            (real_coeffs(18),MO_ZZHP),
      &            (real_coeffs(19),MO_WBZHP),
      &            (real_coeffs(20),MO_GXHP),
      &            (real_coeffs(21),MO_GYHP),
      &            (real_coeffs(22),MO_GZHP),
      &            (real_coeffs(23),MO_GPHIHP),
      &            (real_coeffs(24),MO_GTHEHP),
      &            (real_coeffs(25),MO_GPSIHP),
      &            (real_coeffs(26),MO_FLXMAX),
      &            (real_coeffs(27),MO_FLYMAX)
```

Figure 5.5 Real_coeff.inc file

```
INTEGER*4 IRANAL, ICNT
       REAL*4 FS(3), WS(3), TRANS(3), ANGL(3),DF(27,3),
      &       DW(27,3),DEULER(27,3),EULERS(3),FSRMS(3),
      &       WSRMS(3),EULERSRMS(3),FSPEAK(3),WSPEAK(3),
      &       EULERSPK(3),P_TRANS(3),P_ANGL(3),FRMS(27,3),
      &       WRMS(27,3),EULERRMS(27,3),FPEAK(27,3),WPEAK(27,3),
      &       EULERPK(27,3)
       EQUIVALENCE (ES_FS,FS),(ES_WS,WS),(ES_TRANS,TRANS),
      &            (ES_ANGL,ANGL),(ES_DF,DF),(ES_DW,DW),
      &            (ES_DEULER,DEULER),(ES_EULERS,EULERS),
      &            (ES_FSRMS,FSRMS),
      &            (ES_WSRMS,WSRMS),(ES_EULSRMS,EULERSRMS),
      &            (ES_FSPEAK,FSPEAK),(ES_WSPEAK,WSPEAK),
      &            (ES_EULSPK,EULERSPK),(ES_P_TRANS,P_TRANS),
      &            (ES_P_ANGL,P_ANGL),(ES_FRMS,FRMS),(ES_WRMS,WRMS),
      &            (ES_EULRMS,EULERRMS),(ES_FPEAK,FPEAK),
      &            (ES_WPEAK,WPEAK),(ES_EULERPK,EULERPK),
      &            (ES_IRANL,IRANAL),(ES_ICNT,ICNT)
```

Figure 5.6 Es_equiv.inc file

```
fact(maneuver_importance(quick_stop,'',large)).
fact(maneuver_importance(side_step,'',large)).
fact(maneuver_importance(pedal_turn,'',medium)).
fact(maneuver_importance(collective_pull,'',small)).
```

Figure 5.7 Maneuvre_importance.pl file

CHAPTER 6

TESTING AND EXPERIMENTATION

There are three parts to testing the accuracy of the adaptation of PROTEST. These are the testing of the washout filters, the verification of the functioning of PROTEST and finally an example tuning run. This chapter will describe the method used for each of these and the results obtained.

6.1 Testing the washout filters

Before any progress can be made in implementing PROTEST the washout filters must be ensured to provide the correct response from which statistical data will be drawn for PROTEST. This process involves two steps. The first involves testing the washout filters for the correct response to a step input. For this purpose a test program was written that provides the washout filters with a unit step input and then plots the output. The first test using this program isolated each filter to test for the correct standard response to a step input. Once the correct response was achieved the transformation matrices $\underline{L}_{IS}$ and $\underline{T}_{S}$ were added with the step input being first transformed. Again, the code was debugged until the anticipated response was achieved. The appropriate response was determined by using the baseline washout filters that are known to have the correct response. For these two

washout filters to be equivalent the coefficients sets have to be made to match. This is achieved using Table 3.1. For the purpose of these tests all the gains were set to one.

Once the washout code was debugged it was integrated into the host code. Again the accuracy was tested by comparing the output of the adapted PROTEST washout filters to the response of the baseline. To achieve this the option is given to the user to choose which washout to use for a given run. Both filters are then used with a pre-recorded flight. The difference in outputs is presented in Figure 6.1. In these figures the degree of freedom, in the body frame, is indicated as x, y and z for the specific forces and p, q and r for the angular rates; roll, pitch and yaw. The slight differences can be attributed to the variation in coding and the different integration method used. The differences are small and do not greatly alter the motion response of the simulator. The response of the PROTEST washout filter is thus assumed acceptable and the testing can proceed to the next step.

6.2 Verification of Statistical Data

Once the washout filters are known to respond correctly it is necessary to ensure the adapted components of PROTEST are working properly. The items of importance are those that provide PROTEST with the statistical data it requires to tune the motion. Using the output to the washout filters these values were manually calculated using a Microsoft Excel Spreadsheet. Thus the values calculated during a run in PROTEST are compared to the manually calculated data. Appendix B gives a sample of the information echoed during a PROTEST run. In addition to this information a clearer format of the statistical data is provided in an additional output file. Figure 6.2 gives a sample of this file.

In analysing the statistical values it is important to consider the nature of the manoeuvre being tuned. Although the magnitude of the values themselves at first glance do not necessarily say anything about their accuracy, the order in which these values rank the degrees of freedom can provide a first appreciation of accuracy. Since these lists represent part of the input to PROTEST it is relatively easy to judge if the correct information is being sent. The logic for calculating the statistical data is provided by Grant

[7] and forms the basis for the manual calculations. In all cases the code was debugged until the expected results were achieved.

The first, and easiest, item to check is limiting. For this there are two important values, the limit string and the time at the limits. The former sends the character string 'limit_exceeded' when the limits have been reached while the latter sends the actual time spent at the limits. Checking if these values are accurate simply means timing the limiting during a run and ensuring that this value is the same as that received by PROTEST and that the character string is received by PROTEST. Indeed, this method was used to test accuracy and it was determined that the limiting statistical data was functioning correctly.

Tested next was the translation and angular simulator displacement. The time histories for the normalised translation displacements and angular displacements for the quickstop manoeuvre are given in Figure 6.3 and Figure 6.4. From these plots the maximum displacement values for each channel could be determined. These values are normalised by dividing each element by the single degree of freedom displacement limit of the UTIAS motion base as described by Grant [7] to be,

$$\underline{D}_{\text{lim}} = \begin{bmatrix} 0.69m & 0.59m & 0.51m & 0.37rad & 0.37rad & 0.42rad \end{bmatrix} \quad (6.1)$$

In this list the first three values represent the x, y and z displacements in the inertial frame while the last three the represent the simulator ϕ, θ, and ψ displacements.

The order of the degrees of freedom can then be determined from the magnitude of the resulting values. As was mentioned earlier one could obtain qualitative approximation of accuracy by first looking at the ordered list. It is evident that the degrees of freedom most excited during a given manoeuvre should appear at the top of the resulting ordered list. Similarly, the sensed motion is communicated to PROTEST in the form of an ordered list. The sensed motion is given as the root mean square of the output specific forces and angular rates. These values are then normalised with respect to the human motion perception critical amplitudes, described by Greig [9] and given as,

$$\underline{CA} = \begin{bmatrix} 0.026 & 0.026 & 0.026 & 0.0059 & 0.0059 & 0.0059 \end{bmatrix} \quad (6.2)$$

The first three elements represent the approximate specific force critical amplitude in m/s^2 while the last three represent the approximate angular rate critical amplitude in rad/s. Again, it is clear that the channel most excited during a manoeuvre should appear at the top

of the list. However, attention should be paid to the trimmed attitude of the aircraft that may be sustained during the flight and, because of its sustained nature, would appear at the top of the sensed motion list.

The final, yet maybe most important statistic, is the partial derivative results. These are obtained when a small change, delta, is made to each coefficient and its effect on the six degrees of freedom is observed. The result is the numerical value of the effect and an ordered list of the importance of each coefficient to a particular washout filter. As with the other statistics a qualitative observation of the results can give a good indication of their accuracy. For example a degree of freedom will clearly be more sensitive to those coefficient that directly affect its related washout filter. In addition greater sensitivity should be observed in the degree of freedom being excited. To test this latter point a run was performed where only the heave channel was excited. As expected coefficient sensitivity was only observed in the heave channel. A similar test was run where the surge channel was excited revealing sensitivity in the surge and pitch channels. Again, this is an expected result due to the tilt co-ordination. Equivalent results were obtained when the sway and yaw channels were each individually excited. It is clear that if a certain channel is not excited a change in the coefficients for its related filter will result in no effect on the output. And, if no tilt limiting is occurring an increase in these coefficients should also result in no change in the outputs. These phenomenon were qualitatively tested with the expected result being achieved.

6.3 Tuning Experiment

Once the complete expert system was determined to work a tuning run was performed. This run was based on the complete tuning of the washout filters. It is important to note that the aircraft model used was the Bell 205 ARMCOP in TRC mode. The TRC mode identifies the flight control method as Translational Rate Command. This helicopter model was employed because of its extended use in the UTIAS Flight Simulator Laboratory.

A single pilot was used to conduct the experiments. A captain in the Canadian Armed Forces, this individual is a qualified helicopter pilot with many years of experience. He also

has previous experience with UTIAS Flight Simulator including knowledge of the washout filters and the tuning process. The pilot has also logged a number of hours in other flight simulators. This section will discuss the specifications of the experiment and the results obtained.

6.3.1 The Manoeuvres

To perform the tuning four manoeuvres were selected. These are a quickstop, sidestep, pedal turn, and collective pull. Grant [7] explains that tuning should proceed from the manoeuvre of greatest importance to that of least importance. He also gives the criteria by which importance can be determined. In deciding the order of tuning it is important to keep in mind the capabilities of the simulator. For example, the manoeuvres that are most demanding on the motion of the simulator, and are hardest to tune, should be given less importance since obtaining satisfactory tuning is unlikely anyway. This is the case for the pedal turn and collective pull that are both characterised by sustained motion. For these reasons the quickstop was tuned first, followed by the sidestep, and finally the pedal turn the collective pull. The goal in executing the manoeuvres was to produce a moderately aggressive response. The criteria for the manoeuvres were determined from Tai [2] and the advice of the pilot. The specifications for each manoeuvre are given below.

The test course for the four manoeuvres is a recreation of the NRC Flight Research Laboratory in Ottawa. Tai [2] gives the course layout, which consisted of cone markers in various formations for different manoeuvres and a number of 2D and 3D objects to provide a better representation of the real environment.

The quickstop manoeuvre consists in starting from a stabilised position at 20 ft in altitude, accelerating to a nominal airspeed of 30 kt, and immediately decelerating to a hover at a given reference point. As stated by Tai [2] the maximum nose-down attitude should occur immediately after initiating the manoeuvre and the peak nose-up pitch attitude should occur just before reaching the final stabilised hover. The flight begins in a hover above a line of cones parallel to the x-axis of the helicopter. The pilot is instructed to fly the lateral manoeuvre keeping the same heading and while remaining above a line of cones. The pilot then must stop on the line of cones opposite the initial set. It is of little

importance which direction the test flight is conducted, but both directions should be exercised during the test.

The sidestep manoeuvre starts in a stabilised hover at 20 ft. The pilot is instructed to perform a moderately aggressive lateral translation with a bank angle of approximately 20°. The aircraft should then be brought back to a hover over the finish line of cones. The peak bank angle during deceleration should also be approximately 20°. During the manoeuvre there should be minimal change in altitude. The manoeuvre was flown starting with the aircraft on the left line of cones and flying to right.

Both the pedal turn and collective pull were started from a stabilised hover at the position labelled 'Starting position for hover' at an altitude of 20 ft. The former consisted in performing a controlled pedal turn at an average rate of 7°/sec for 360°. During this manoeuvre there should be no change in altitude. The collective pull consists in climbing at a rate of 1ft/sec until 100 ft is reached, stabilising the aircraft and descending at the same rate to the starting position.

6.3.2 Flight Test Procedure

Each run was begun with a stabilised hover at the designated position. The pilot was then reminded of the desired manoeuvre to be flown and given any special instructions. For all flights the pilot is required to report on the status of the flight, such as the start and finish of a run. This last point is crucial for the tester to know when to start and stop the statistical components of PROTEST. If the run has met the particular flight criteria the pilot is asked if the run was satisfactory. If this is the case the statistics of the flight are sent to PROTEST for the run evaluation to begin. The instructions for PROTEST are given in Appendix C. PROTEST will return the new coefficients and instruct the pilot to fly again. The manoeuvre are flown in the order described above, with each manoeuvre tuned independently.

6.3.3 The Results

The success of the experiment is determined by whether or not the UTIAS simulation of the Bell 205 helicopter can be acceptably tuned using PROTEST. This goal is

characterised by a set of coefficients that produce motion that satisfies the test pilot. It is hoped that this level of tuning can be achieved through a minimum number of steps.

To this end a series of experiments was conducted. The initial and final sets of coefficients are given in Table 6.1. At the end of the test the pilot was satisfied that the motion obtained was acceptable given the physical limits of the UTIAS flight simulator. However, through the entire experimentation procedure the pilot complained of jerkiness in the roll channel. This problem was first hypothesised as a washout filter tuning issue and consequently PROTEST was used to attempt to rectify the motion. However, the ultimate correction of the 'motion jerkiness' in roll resulted in the suppression of regular motion cueing. It appears now that the erroneous motion cue could be a result of the flight model itself and not a tuning problem. The correct tuning of all other channels and the successful testing of the washout filters support this hypothesis. Tuning all four manoeuvres took 6 hours with the most time being spent on the quickstop and sidestep manoeuvres. Appendix B provides the PROTEST output for the quickstop manoeuvre.

Table 6.1 PROTEST Experiment Tuning results

	Coefficient		Value	
	NUM.	VARIABLE	INITIAL	FINAL
ROLL	1	$\omega_{hp\phi}$	1.5	3.384
	2	$\zeta_{hp\phi}$	0.0	0.0
Pitch	3	$\omega_{hp\theta}$	1.5	0.513
	4	$\zeta_{hp\theta}$	0.0	0.0
Yaw	5	$\omega_{hp\psi}$	4.0	0.303
	6	$\zeta_{hp\psi}$	1.0	1.296
Surge	7	ω_{hpx}	4.0	1.934
	8	ζ_{hpx}	1.0	0.642
	9	ω_{bx}	0.0	0.0
Pitch Low pass	10	ω_{lpx}	4.0	2.254
	11	ζ_{lpx}	1.0	1.0
Sway	12	ω_{hpy}	4.0	2.417
	13	ζ_{hpy}	1.0	1.0
	14	ω_{by}	0.0	0.0
Roll Low Pass	15	ω_{lpy}	4.0	1.95
	16	ζ_{lpy}	1.0	1.0
Heave	17	ω_{hpz}	4.0	3.59
	18	ζ_{hpz}	1.0	1.0
	19	ω_{bz}	0.0	0.18
Gains	20	k_x	0.5	0.5
	21	k_y	0.5	0.5
	22	k_z	1.0	1.0
	23	k_p	0.5	0.5
	24	k_q	0.5	0.5
	25	k_r	1.0	1.0
Limiters	26	flxmax	0.228	0.228
	27	flymax	0.5137	0.5137

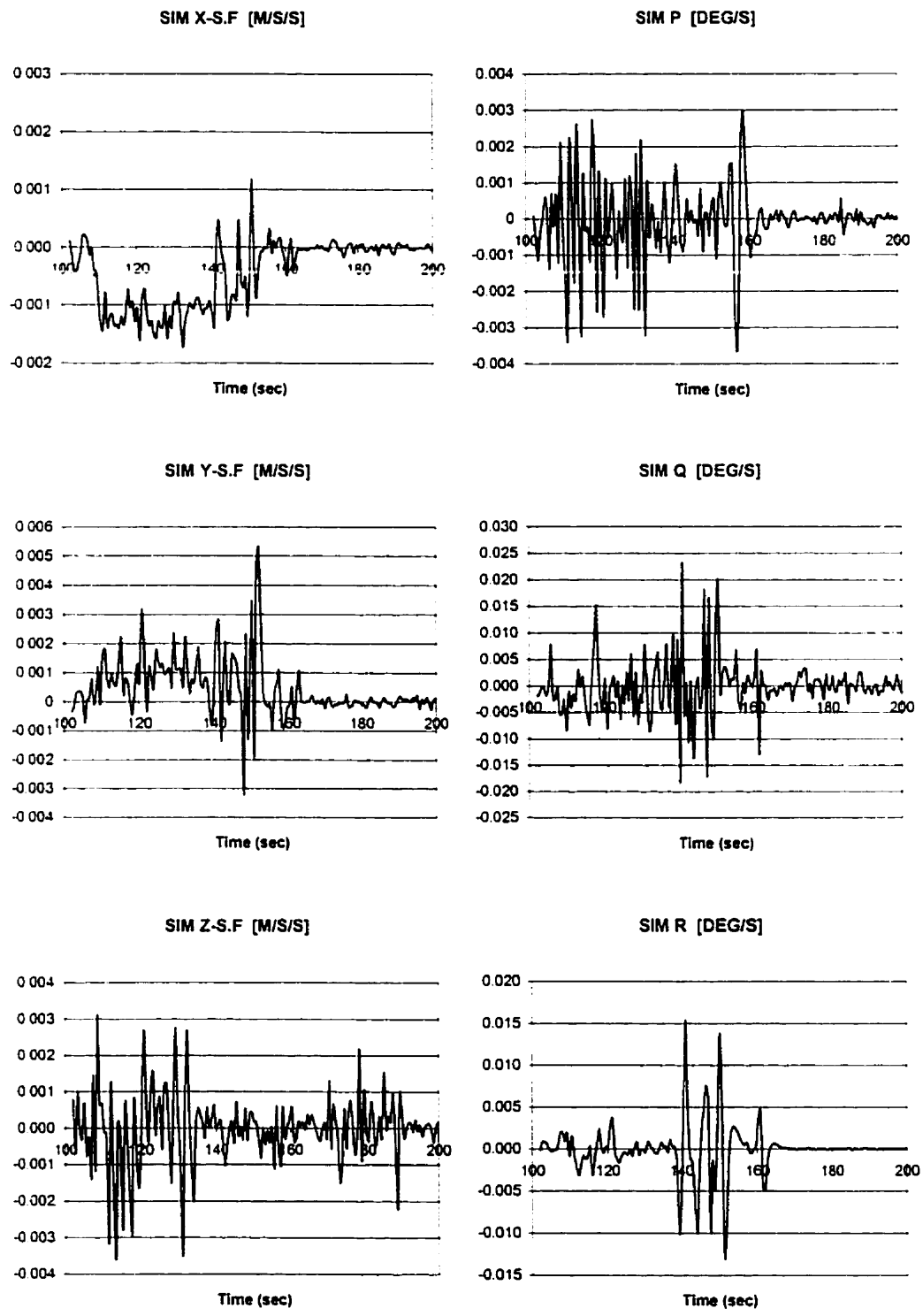

Figure 6.1 Specific Force and Angular rate compared output

Figure 6.2 Sample of Statistical Output File

```
sorted displacements
   pitch
   surge
   heave
   roll
   sway
   yaw
  sorted sensed motion
   surge
   pitch
   sway
   heave
   roll
   yaw
 ** Axis #            1
gain-roll
gain-surge
whp-pitch
zhp-surge
whp-surge
zlp-surge
wlp-surge
zhp-heave
gain-heave
whp-heave
gain-sway
gain-pitch
gain-yaw
whp-roll
zlp-sway
wlp-sway
zhp-yaw
zhp-sway
whp-yaw
whp-sway
tilt_rate_limit-surge
tilt_rate_limit-sway
 ** Axis #            2
gain-sway
gain-pitch
whp-roll
zhp-sway
zlp-sway
whp-sway
wlp-sway
gain-surge
whp-pitch
gain-yaw
gain-roll
gain-heave
zhp-heave
```

```
whp-heave
zhp-yaw
zlp-surge
whp-yaw
zhp-surge
wlp-surge
whp-surge
tilt_rate_limit-surge
tilt_rate_limit-sway
 ** Axis #            3
gain-roll
gain-heave
gain-surge
whp-pitch
zhp-heave
whp-heave
gain-sway
gain-pitch
whp-surge
zhp-surge
zlp-surge
whp-roll
wlp-surge
zhp-sway
zlp-sway
wlp-sway
whp-sway
gain-yaw
zhp-yaw
whp-yaw
tilt_rate_limit-surge
tilt_rate_limit-sway
 ** Axis #            4
gain-pitch
gain-sway
whp-roll
zlp-sway
wlp-sway
gain-roll
gain-surge
whp-pitch
gain-yaw
zhp-yaw
whp-yaw
zlp-surge
wlp-surge
whp-surge
zhp-surge
whp-sway
zhp-sway
whp-heave
zhp-heave
gain-heave
tilt_rate_limit-surge
tilt_rate_limit-sway
```

```
 ** Axis #           5
gain-roll
whp-pitch
gain-surge
zlp-surge
wlp-surge
gain-sway
gain-pitch
gain-yaw
whp-roll
zlp-sway
zhp-yaw
wlp-sway
whp-yaw
whp-surge
zhp-surge
whp-sway
zhp-sway
whp-heave
zhp-heave
gain-heave
tilt_rate_limit-surge
tilt_rate_limit-sway
 ** Axis #           6
gain-yaw
zhp-yaw
gain-roll
gain-sway
whp-yaw
gain-pitch
whp-pitch
gain-surge
whp-roll
zlp-sway
wlp-sway
zlp-surge
wlp-surge
whp-surge
zhp-surge
whp-sway
zhp-sway
whp-heave
zhp-heave
gain-heave
tilt_rate_limit-surge
tilt_rate_limit-sway
  *** sensed motions ***
  fsrms  0.5926664472      0.8624271303E-01  0.6226050481E-01
  wsrms  0.8573954925E-02  0.2450641058E-01  0.9324681596E-03
  *** partial derivatives ***
  ** fs first **
        .00012          .03433          .00033
        .00430         1.68085          .02693
       1.01950          .00263          .09915
       9.62864          .03358          .49199
```

```
        .00001          .00002          .00000
        .00001          .00003          .00000
        .01452          .00001          .00095
        .01768          .00001          .00094
        .01965          .00001          .00115
        .00334          .00001          .00030
        .00624          .00002          .00056
        .00000          .01138          .00008
        .00001          .02302          .00015
        .00000          .00512          .00005
        .00002          .00809          .00008
        .00004          .01280          .00011
        .00185          .00025          .03518
        .00323          .00046          .06642
        .00112          .00016          .01855
       1.59169          .00390          .10490
        .00140          .29214          .00321
        .00189          .00063          .10846
       1.60967          .00258          .19152
        .00057          .15058          .00178
        .00052          .00262          .00003
        .00000          .00000          .00000
        .00000          .00000          .00000
** ws first **
        .00475          .00001          .00010
        .24250          .00061          .00315
        .00024          .04640          .00037
        .00209          .43444          .00360
        .00002          .00000          .00046
        .00005          .00000          .00103
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00000          .00051          .00000
        .00000          .00080          .00001
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00297          .00000          .00003
        .00326          .00000          .00004
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00000          .00000          .00000
        .00026          .02908          .00023
        .01826          .00006          .00068
        .00000          .00000          .00000
        .00033          .09901          .00081
        .02496          .00006          .00042
        .00021          .00003          .00185
        .00000          .00000          .00000
        .00000          .00000          .00000
*** total motion/channel ***
trans   0.6466626376E-01  0.8345603012E-02  0.3879805654E-01
angular 0.2055826038E-01  0.1526836753      0.1197624719E-02
```

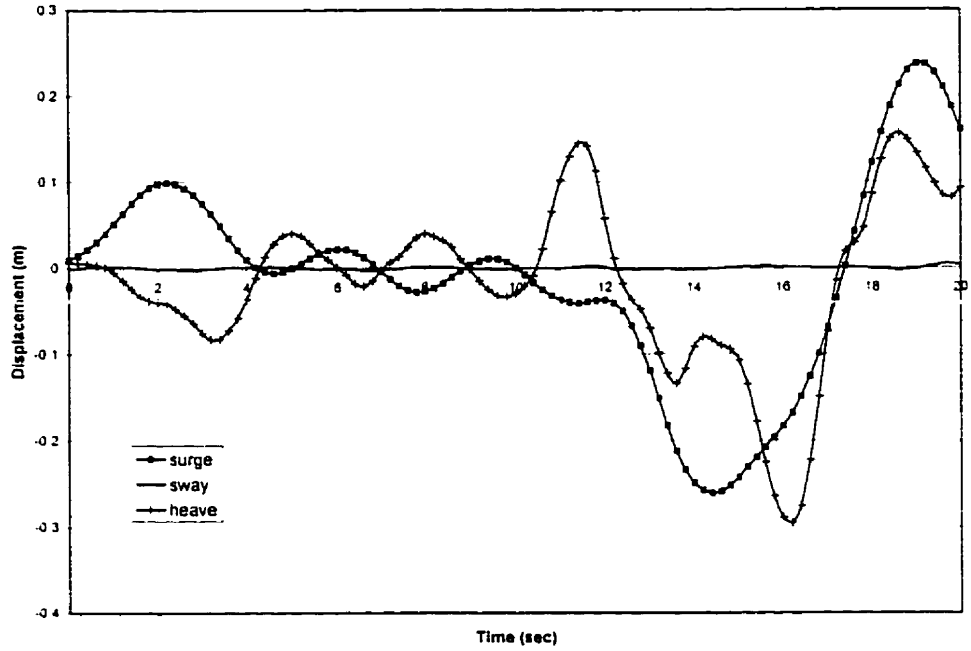

Figure 6.3 Normalised Translation Displacement Time History for Quickstop Maneuver

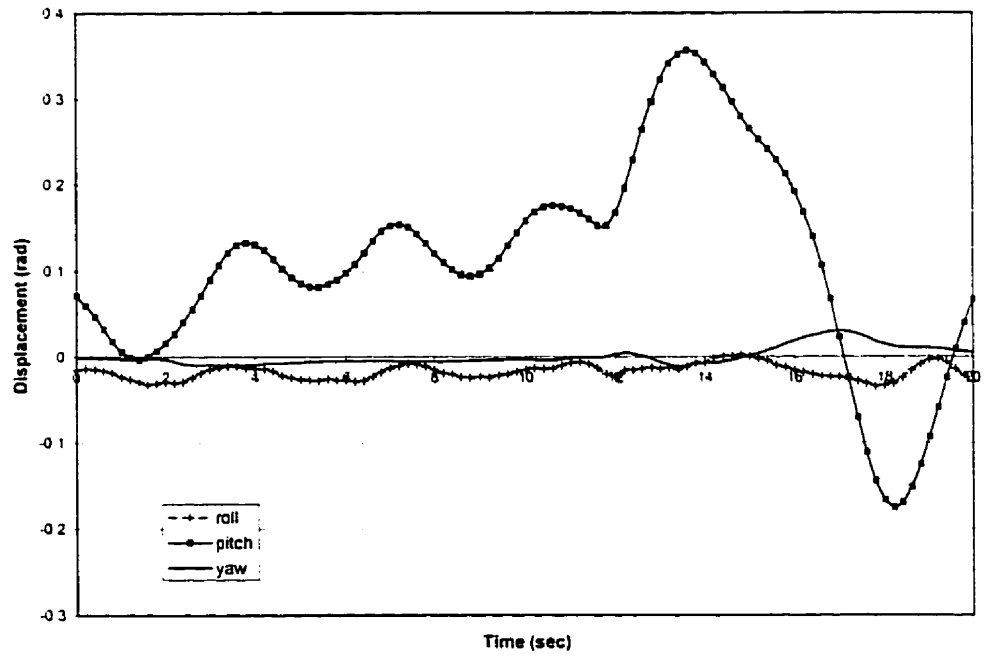

Figure 6.4 Normalised Angular Displacement Time History for Quickstop Manoeuvre

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

Although successful in its task of adapting the original version of the PROTEST Expert System to the current UTIAS Flight Simulator environment, the experience has led to a number of conclusions and recommendations. It has become evident that PROTEST is not, at this point, a self-contained software package that can be incorporated into any simulator system. This task requires an in depth knowledge of both the PROTEST software and the host code. For this reason it would be advantageous to develop a version of the PROTEST code that has a wider range of software compatibility.

Although Appendix C provides a user manual for PROTEST it does not give any information on the modification of the Prolog code, nor does it specify how new rules and logic can be added. These features could be useful for systems with vastly different washout filters or special features. An exploration into the method for adding rules would be beneficial in the future.

Grant [7] in his recommendations suggests that a better front end user interface could be developed for PROTEST. This recommendation is still valid. It would be an asset to PROTEST to have a more intuitive front-end interface.

In terms of the actual usage of PROTEST three recommendations can be made. The first would give the user more control to be able to redirect PROTEST. The second recommendation is to provide a more robust system in terms of user errors. In the current form it seems the Prolog code does not allow the user to make a mistake without the run having to be repeated. Both the above could be solved by giving the user backtracking control. This would take the form of the ability to easily return to a previous point in the tuning procedure. Finally, it would be an asset to PROTEST to allow the user to save the run at any point in the tuning procedure, not only at some predetermined locations.

Finally the tuning procedure using PROTEST may have revealed a fault in the roll/sway flight model. It would be interesting to investigate this characteristic of the UTIAS ARMCOP model in search of the exact cause of the erroneous motion cues.

The goal of this project was to implement, into the current UTIAS computing environment, a functioning version of PROTEST. This objective has been achieved and the steps in so doing have been presented in this document. It is hoped that by reproducing these steps in combination with customised adaptation, PROTEST can be fitted into any flight simulator system.

It would now be interesting to conduct experiments where the tuning procedure is begun from a number of different initial coefficient sets to investigate the path PROTEST will use to arrive at a valid set. Also the current configuration will allow pilot variability to be tested.

APPENDIX A

COEFFICIENT ADJUSTMENT TABLES

This Appendix replicates the coefficient adjustment tables found in Grant's thesis [7]. Grant provides a comprehensive description regarding these adjustments to which the reader is referenced if more detail is required. However, the tables provided give the information required during regular tuning operations.

False Cue in Heave or Yaw		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
⇑ $\zeta_{hpz,hp\psi}$	*Source*: Return to neutral false cue	*Likelihood* 4 - Only source of false cues on heave or yaw degrees-of-freedom
⇓ $k_{z,r}$	*Source*: All false cues types	*Likelihood* 4 - reduces all types

False Cue in Surge or Sway		
⇑ $\omega_{hp\theta,hp\dot{u}}$	*Source*: Co-ordinated angular input specific force false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue
⇓ $k_{p,q}$	*Source*: Co-ordinated angular input specific force false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue
⇑ $\omega_{lpx,lpy}$	*Source*: Tilt-coordination remnant, return to neutral specific force false cues	*Likelihood* 4 - very likely source, coefficient directly controls false cue if no tilt-rate limiting
⇓ $\omega_{lpx,lpy}$	*Source*: Tilt-coordination angular acceleration specific force false cue	*Likelihood* 1 - unlikely source, coefficient directly controls false cue, but this false cue usually is identified as motion jerkiness NOT as a false cue

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7]

False Cue in Surge or Sway, Continued		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
$\Uparrow k_{x,y}$	*Source*: Tilt-coordination remnant, return to neutral, and tilt-coordinaiton acceleration specific force false cues	*Likelihood* 4 - remnant very likely source, coefficient directly controls remnant false cue
$\Downarrow \dot{\phi}_{lim}, \dot{\theta}_{lim}$	*Source*: Tilt-coordination remnant specific force false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue if significant tilt-rate limiting
$\Downarrow \omega_{hpx, hpy}$	*Source*: Coordinated angular input, return to neutral specific force false cue	*Likelihood* 3 - coordinated angular input very likely source, coefficient controls translational coordination, ineffective on typical motion systems due to displacement limitations
$\Uparrow \zeta_{hpx,hpy}$	*Source*: Return to neutral false cue	*Likelihoo* 2 - unlikely source, coefficient directly controls false cue

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7](continued)

False Cue in Roll or Pitch		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
⇓ $\omega_{lpx,\ lpy}$	*Source*: tilt-coordination angular rate false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue, if insignificant tilt-rate limiting
⇓ $\dot{\phi}_{lim},\ \dot{\theta}_{lim}$	*Source*: Tilt-coordination angular rate false cue	*Likelihood* 4 - very likely source, coefficient directly controls maximum amplitude of false cue
⇓ $k_{x,y}$	*Source*: Tilt-coordination angular rate false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue

Jerkiness in Heave or Yaw		
⇓ $\omega_{hpz,hp\psi}$	*Source*: Missing high-frequency transient cues	*Likelihood* 4 - only source of jerkiness, coeffcient provides direct control
⇓ $\zeta_{hpz,hp\psi y}$	*Source*: Missing high-frequency transient cues	*Likelihood* 3 - only source of jerkiness, above coefficient provides more direct control

Jerkiness in Surge or Sway		
⇓ $\omega_{lpx,lpy}$	*Source*: Tilt-coordination angular acceleration specific force false cue	*Likelihood* 4 - verylikely source, coefficient directly controls motion jerkiness
⇓ $\omega_{hpx,hpy}$	*Source*: Missing high-frequency transient cues	*Likelihood* 4 - very likely source of jerkiness, coefficient provides direct control
⇓ $\zeta_{hpx,hpy}$	*Source*: Missing high-frequency transient cues	*Likelihood* 3 - very likely source of jerkiness, above coefficient provides more direct control
⇓ $k_{x,y}$	*Sourse*: Tilt-coordination angular acceleration specific force false cue	*Likelihood* 4 - verly likely source, coefficient will directly control jerkiness

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7] (continued)

Motion Jerkiness in Pitch or Roll		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
⇓ $\omega_{hp\phi, hp\theta}$	*Source*: Missing high-frequency transient cue	*Likelihood* 4 - very likely source, coefficient provides direct control
⇓ $\zeta_{hp\phi,hp\psi}$	*Source*: Missing high-frequency transient cue	*Likelihood* 3 - very likely source, but above provides more direct control

Too Small Motion Amplitude on Angular Degree-of-Freedom		
⇓ $\omega_{hp\phi, hp\theta, hp\psi}$	*Source*: Missing or scaled down high-frequency transient cue	*Likelihood* 4 - very likely source of motion amplitude PROBLEM
⇑ $k_{hp\phi, hp\theta, hp\psi}$	*Source*: Missing or scaled down cues	*Likelihood* 3 - very likely source, coefficient provides direct control

Too Small Motion Amplitude on Translational Degree-of-Freedom		
⇓ $\omega_{hpx, hpy, hpz}$	*Source*: Missing or scaled down high-frequency transient cue	*Likelihood* 4 - very likely source, coefficient provides direct control
⇑ $\omega_{lpx, lpy}$	*Source*: Missing or scaled down high-frequency transient cue	*Likelihood* 3 - likely source of motion amplitudes PROBLEM
⇑ $k_{hpx, hpy, hpz}$	*Source*: Missing or scaled down cues	*Likelihood* 4 - very likely source, coefficient provides direct control

Too Large Motion Amplitude on Angular Degree-of-Freedom		
⇑ $\omega_{hp\phi, hp\theta, hp\psi}$	*Source*: Scaled up high-frequency transient cue	*Likelihood* 4 - very likely source, coefficient provides direct control
⇓ $k_{hp\phi, hp\theta, hp\psi}$	*Source*: Scaled up cues	*Likelihood* 4 - very likely source, coefficient provides direct control

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7] (continued)

Too Large Motion Amplitude on Translational Degree-of-Freedom		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
⇑ $\omega_{hpx,\ hpy,\ hpz}$	*Source*: Scaled up high-frequency transient cue	*Likelihood* 4 - very likely source, coefficient provides direct control
⇓ $\omega_{lpx,\ lpy}$	*Source*: Scaled up high-frequency transient cue	*Likelihood* 3 - likely source of motion amplitudes PROBLEM
⇓ $k_{hpx,\ hpy,\ hpz}$	*Source*: Scaled down cues	*Likelihood* 4 - very likely source, coefficient provides direct control

Too Short Motion Duration on Heave, Pitch, Roll or Yaw		
⇓ $\omega_{hpz,\ hp\Phi,\ hp\Theta,\ hp\Psi}$	*Source*: High-frequency transient missing or scaled cue	*Likelihood* 4 - very likely source of Duration complaints, coefficient provides direct control
⇓ $\zeta_{hpz,\ hp\Phi,\ hp\Theta,\ hp\Psi}$	*Source*: High-frequency transient missing or scaled cue	*Likelihood* 3 - very likely source of Duration complaints, previous coefficient provides more direct control

Too Long Motion Duration on Surge or Sway		
⇑ $\omega_{lpx,\ lpy}$	*Source*: Tilt-coordination remnant, return to neutral false cue	*Likelihood* 4 - very likely source, coefficient directly controls false cue, if no tilt-rate limiting
⇑ $\dot{\phi}_{lim},\ \dot{\theta}_{lim}$	*Source*: Tilt-rate remnant	*Likelihood* 4 - very likely source, coefficient directly controls false cue if significant tilt-rate limiting
⇓ $k_{x,y}$	*Source*: Tilt-coordination remnant	*Likelihood* 4 - very likely source, coefficient directly controls false cue

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7] (continued)

Too Short Motion Duration on Surge or Sway		
Adjustment Used to Reduce Error	Cueing Error Source	Likelihood
$\Downarrow \omega_{hpx,\ hpy}$	*Source*: High-frequency transient missing or scaled cue	*Likelihood* 4 - very likely source of duration complaints, coefficient provides direct control
$\Downarrow \zeta_{hpx,\ hpy}$	*Source*: High-frequency transient missing or scaled cue	*Likelihood* 4 - very likely source of duration complaints, coefficient provides more direct control

Too Long Motion Duration on Surge or Sway		
$\Uparrow \omega_{lpx,\ lpy}$	*Source*: Missing or scaled low-frequency transient cues	*Likelihood* 4 - very likely source, if not significant tilt-rate limiting then this provides direct control
$\Uparrow \dot{\phi}_{lim},\ \dot{\theta}_{lim}$	*Source*: Missing or scaled low-frequency transient cues	*Likelihood* 4 - very likely source, if significant tilt-rate limiting then this provides direct control

Unknown PROBLEM in Heave or Yaw		
? $k_{z,\ r}$	*Source*: Unknown	*Likelihood* 4
? $\omega_{hpz,\ hp\psi}$	*Source*: Unknown	*Likelihood* 4
? $\zeta_{hpz,\ hp\psi}$	*Source*: Unknown	*Likelihood* 4

Unknown PROBLEM in Surge or Sway		
? $k_{x\ y}$	*Source*: Unknown	*Likelihood* 4
? $\omega_{hpx,\ hpy}$	*Source*: Unknown	*Likelihood* 4
? $\zeta_{hpx,\ hpy}$	*Source*: Unknown	*Likelihood* 4
? $\dot{\phi}_{lim},\ \dot{\theta}_{lim}$	*Source*: Unknown	*Likelihood* 4

Unknown PROBLEM in Heave or Yaw		
? $k_{p,\ q}$	*Source*: Unknown	*Likelihood* 4
? $\omega_{hp\phi,\ hp\theta}$	*Source*: Unknown	*Likelihood* 4
? $\zeta_{hp\phi,\ hp\theta}$	*Source*: Unknown	*Likelihood* 4

Table A.1 Coefficient Adjustments for Selected PROBLEMs [7] (concluded)

APPENDIX B

SAMPLE PROTEST PRINTOUT

This appendix gives the output of PROTEST for the complete tuning of motion jerkiness in the pitch channel. Also included is a sample of the statistical file produced after each run.

```
Setprolog
yes

 * consulting all_rule_all_couple_rn.pl *
 * consulting fuzzy_small_new *
 ** Debug rules ? (yes or no)
yes

 ** Maneuver Type: a, b, c, d, or all ?
    Quick Stop = a
    Side Step = b
    Pedal Turn = c
    Collective Pull = d
You have chosen: a
 ** interactive or offline ? **
You have chosen: interactive.  Please setup host sockets now

 ** restart or new ? **
You have chosen: new

 ** created sending socket: handle = 6 **
 ** created receiving socket: handle = 8 **
Current coefficients:
[1.5, 0.0, 0.513, 0.0, 4.0, 1.0, 1.934, 0.642, 0.0, 2.254, 1.0, 4.0, 1.0,
          0.0, 4.0, 1.0, 4.0, 1.0, 0.0, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 0.228,
          0.514]

 ** Message to pilot **
 Please fly quick_stop


 ** Performing Garbage collection **


 ** Getting New Run Data **

** getting message from vuot
surge:pitch:heave:roll:sway:yaw;surge:pitch:sway:heave:roll:yaw;;gain-roll:gai
          n-surge:whp-pitch:whp-surge:zhp-surge:gain-heave:zlp-surge:wlp-surge:z
          hp-heave:whp-heave:gain-sway:gain-pitch:gain-yaw:whp-roll:zlp-sway:wlp
          -sway:zhp-yaw:whp-yaw:zhp-sway:whp-sway:tilt_rate_limit-surge:tilt_rat
          e_limit-sway;gain-sway:gain-pitch:whp-roll:zhp-sway:zlp-sway:whp-sway:
          wlp-sway:gain-surge:whp-pitch:gain-yaw:gain-roll:gain-heave:zhp-heave:
          whp-heave:whp-surge:zhp-surge:zhp-yaw:wlp-surge:zlp-surge:whp-yaw:tilt
          _rate_limit-surge:tilt_rate_limit-sway;gain-roll:gain-heave:gain-surge
          :whp-pitch:zhp-heave:whp-heave:whp-surge:zhp-surge:gain-sway:gain-pitc
          h:wlp-surge:zlp-surge:whp-roll:zhp-sway:zlp-sway:wlp-sway:whp-sway:gai
          n-yaw:zhp-yaw:whp-yaw:tilt_rate_limit-surge:tilt_rate_limit-sway;gain-
          pitch:gain-sway:whp-roll:zlp-sway:wlp-sway:gain-roll:gain-surge:whp-pi
          tch:gain-yaw:zhp-yaw:whp-yaw:wlp-surge:zlp-surge:whp-surge:zhp-surge:w
          hp-sway:zhp-sway:whp-heave:zhp-heave:gain-heave:tilt_rate_limit-surge:
          tilt_rate_limit-sway;gain-roll:whp-pitch:gain-surge:wlp-surge:zlp-surg
          e:gain-sway:gain-pitch:gain-yaw:whp-roll:zlp-sway:zhp-yaw:wlp-sway:whp
          -yaw:whp-surge:zhp-surge:whp-sway:zhp-sway:whp-heave:zhp-heave:gain-he
```

```
        ave:tilt_rate_limit-surge:tilt_rate_limit-sway;gain-yaw:zhp-yaw:gain-r
        oll:gain-sway:whp-yaw:gain-pitch:whp-pitch:gain-surge:whp-roll:zlp-swa
        y:wlp-sway:wlp-surge:zlp-surge:whp-surge:zhp-surge:whp-sway:zhp-sway:w
        hp-heave:zhp-heave:gain-heave:tilt_rate_limit-surge:tilt_rate_limit-sw
        ay;47.506874:1.414788:9.464442:5.556846:41.579430:0.285027;0.000000;0.
        621734:0.000000:0.073298:0.065126:0.000000:0.005371:0.900740:0.000352:
        0.000000:0.123912:0.000000:0.041071:0.083041:0.000000:0.029204:1.00000
        0:0.009042:0.000000:0.167148:0.315551:0.000000:0.515897:0.190470:0.000
        000:0.000000:0.000000:0.000000:0.118796:0.733251:0.013079:0.000000:0.4
        68501:0.000000:0.000000:0.000000:0.000000:0.009123:0.221303:0.000581:0
        .000000:0.250906:0.557237:1.000000;


 Enter Complaint (One of following):
(motion_duration motion_amplitude motion_jerkiness false_cue motion_lag
        unknown none move_on)
 Enter Term (One of following):
(large medium small ok exit_and_undo)
 Enter Hedge (One of following):
(very  somewhat)
 Your Response :
very small motion_jerkiness
 Is this OK ? (yes or no)
 Enter Axis (One of the following list)
(surge sway heave roll pitch yaw)
 Your Response :
pitch
 Is this OK (yes or no)

 ** Trying to identify Parameter **


 ** Parameter whp-pitch Hypothesised **
 ** to tune : pitch
 ok (char + return) or select new (P-DOF)
 You have chosen: o


 *** Forward reasoning for hyp and test ***

 if tune_parameter(whp, pitch) and whp(_1, _2, _3, 0.513) ~ pitch and (coupled
        _dof(pitch, surge) or surge = pitch) and wlp(_4, _5, _6, 2.254)
        ~ surge and 0.227595 is 0.513 / 2.254
 then
whp_wlp(very_very, medium, [0, 0, 0, 0, 0.085402, 1.0, 0, 0, 0, 0, 0, 0,
        0], 0.227595) ~ pitch in default

 if hyp_and_test and not reverse_hyp_and_test and pilot_complaint([very,
        small, motion_jerkiness]) and (small \= ok)
 then
motion_jerkiness(very, large, [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0009,
        0.04, 0.25, 0.64, 0.9025, 1.0], 9.08914) in primary

 if tune_parameter(whp, pitch)
```

```
 then
motion_jerkiness -> adjustment ~ whp ~ pitch in primary
adjustment(very, large-ve, [1.0, 0.25, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 0.415936) ~ whp ~ pitch

 if hyp_and_test
 then
adjust_direction(unknown) in primary

 if adjust_direction(unknown)
 then
pilot_inputs(unknown) in primary

 if pilot_inputs(unknown) and not nonmonotonic_problem and adjustment(very,
        large-ve, [1.0, 0.25, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0], 0.415936) ~ whp ~ pitch
 then
m_adjustment(very, large-ve, [1.0, 0.25, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.415936) ~ whp ~ pitch in primary

 if tune_parameter(whp, pitch) and m_adjustment(very, large-ve, [1.0, 0.25,
        0.04, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.415936)
        ~ whp ~ pitch and less_than_one(0.415936)
 then
m_adjustment ~ whp ~ pitch and simulator_motion ~ pitch -> adjustment_1
        ~ whp ~ pitch in primary
adjustment_1(, medium-ve, [0.0, 0.5, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 0.542883) ~ whp ~ pitch

 if tune_parameter(whp, pitch)
 then
adjustment_1 ~ whp ~ pitch and whp ~ pitch -> adjustment_2 ~ whp ~ pitch
        in primary
adjustment_2(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch

 if tune_parameter(whp, pitch)
 then
adjustment_2 ~ whp ~ pitch and whp_wlp ~ pitch -> adjustment_3 ~ whp ~
        pitch in primary
adjustment_3(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch

 if maneuver(quick_stop) and maneuver_importance(quick_stop, , large)
 then
maneuver_importance(, large, [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03,
        0.2, 0.5, 0.8, 0.95, 1.0], 8.80268) in default

 if simulator_motion(_7, _8, _9, 5.55685) ~ roll and simulator_motion(very_ver
        y, medium, [0, 0, 0, 0, 0.010578, 1.0, 0, 0, 0, 0, 0, 0, 0], 41.5794)
        ~ pitch and simulator_motion(_10, _11, _12, 0.285027) ~ yaw and
        simulator_motion(_13, _14, _15, 47.5069) ~ surge and simulator_motion
        (_16, _17, _18, 1.41479) ~ sway and simulator_motion(_19, _20,
        _21, 9.46444) ~ heave and 105.807 is 5.55685 + 41.5794 + 0.285027
```

```
        + 47.5069 + 1.41479 + 9.46444
 then
total_motion(105.807) in default

 if tune_parameter(whp, pitch) and (whp \= wlp) and (whp \= tilt_rate_limit)
        and ((whp \= gain) or (pitch \= surge) and (pitch \= sway)) and
        total_motion(105.807) and simulator_motion(very_very, medium,
        [0, 0, 0, 0, 0.010578, 1.0, 0, 0, 0, 0, 0, 0, 0], 41.5794) ~ pitch
        and 39.2973 is 41.5794 / 105.807 * 100.0
 then
relative_motion(very_very, medium, [0, 0, 0, 0, 0.397285, 1.0, 0, 0, 0,
        0, 0, 0, 0], 39.2973) ~ whp ~ pitch in default

 if tune_parameter(whp, pitch)
 then
maneuver_importance and relative_motion ~ whp ~ pitch -> dof_importance
        ~ whp ~ pitch in primary
dof_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch

 if tune_parameter(whp, pitch) and tuning_dof(pitch)
 then
parameter_importance ~ whp ~ pitch ~ pitch and dof_importance ~ whp ~ pitch
        -> new_importance ~ whp ~ pitch in primary
new_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch

 if hyp_and_test and tune_parameter(whp, pitch) and adjustment_2(, medium-ve,
        [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0], 0.611079) ~ whp ~ pitch
 then
adjustment_3(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed in
        primary

 if tune_parameter(whp, pitch) and adjustment_3(, medium-ve, [0.0, 0.05,
        0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
        0.611079) ~ whp ~ pitch
 then
adjustment_4(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch in primary

 if tune_parameter(whp, pitch) and hyp_and_test and adjustment_3(, medium-ve,
        [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed
 then
adjustment_4(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed in
        primary

 if tune_parameter(whp, pitch) and (not total_tuned(whp, pitch, _22, _23)
        or simulator_limit_exceeded) and adjustment_4(, medium-ve, [0.0,
        0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0], 0.611079) ~ whp ~ pitch
```

```
 then
adjustment_5(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch in primary

 if tune_parameter(whp, pitch) and hyp_and_test and (not total_tuned(whp,
        pitch, _24, _25) or simulator_limit_exceeded) and adjustment_4(,
        medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed
 then
adjustment_5(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed in
        primary

 if tune_parameter(whp, pitch) and (not total_tuned(whp, pitch, _26, _27)
        or simulator_limit_exceeded) and adjustment_5(, medium-ve, [0.0,
        0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0], 0.611079) ~ whp ~ pitch
 then
adjustment_6(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch in primary

 if tune_parameter(whp, pitch) and hyp_and_test and (not total_tuned(whp,
        pitch, _28, _29) or simulator_limit_exceeded) and adjustment_5(,
        medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed
 then
adjustment_6(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed in
        primary

 if m_adjustment(very, large-ve, [1.0, 0.25, 0.04, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.415936) ~ whp ~ pitch and less_than_
        one(0.415936)
 then
parameter_direction(decrease) in primary

 if parameter_direction(decrease) and tune_parameter(whp, pitch)
 then
max_adjustment(very, large-ve, [1.0, 0.25, 0.04, 0.0025, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.416273) ~ whp ~ pitch in primary

 if parameter_direction(decrease) and tune_parameter(whp, pitch) and (whp
        = whp or whp = zhp or whp = tilt_rate_limit or whp = wlp)
 then
current_tuning_direction(increasing_motion) in primary

 if adjustment_6(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed
        and not parameter_interaction_bypassed and not new_adjustment(_30,
        _31, _32, _33) ~ whp ~ pitch ~ bypassed
 then
new_adjustment(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch ~ bypassed
        in primary
```

```
if adjustment_6(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch and not
        parameter_interaction_bypassed and not new_adjustment(_34, _35,
        _36, _37) ~ whp ~ pitch
then
new_adjustment(, medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch in primary


** suggested adjustment ** : whp DOF: pitch multiply by: 0.611079

Is this ok (char + return) or enter new value (float)
You have chosen: o
** actual adjustment ** : whp DOF: pitch multiply by: 0.611079

Sending Parameter whp-pitch Value = 0.313483
Sending Parameter null-null Value = 0.0

** Message to pilot **
Please fly quick_stop


** Performing Garbage collection **


** Getting New Run Data **

** getting message from vuot
surge:pitch:heave:roll:sway:yaw;surge:pitch:sway:heave:roll:yaw;;gain-roll:whp
        -pitch:gain-surge:zhp-surge:whp-surge:gain-heave:zlp-surge:wlp-surge:z
        hp-heave:whp-heave:gain-sway:gain-pitch:gain-yaw:whp-roll:zlp-sway:wlp
        -sway:zhp-yaw:whp-yaw:zhp-sway:whp-sway:tilt_rate_limit-surge:tilt_rat
        e_limit-sway;gain-sway:gain-pitch:whp-roll:zhp-sway:zlp-sway:whp-sway:
        wlp-sway:whp-pitch:gain-roll:gain-surge:gain-yaw:gain-heave:zhp-heave:
        whp-heave:zhp-surge:zhp-yaw:whp-surge:whp-yaw:wlp-surge:zlp-surge:tilt
        _rate_limit-surge:tilt_rate_limit-sway;gain-roll:whp-pitch:gain-surge:
        gain-heave:zhp-heave:whp-heave:zhp-surge:whp-surge:gain-sway:gain-pitc
        h:wlp-surge:zlp-surge:whp-roll:zhp-sway:zlp-sway:whp-sway:wlp-sway:gai
        n-yaw:zhp-yaw:whp-yaw:tilt_rate_limit-surge:tilt_rate_limit-sway;gain-
        pitch:gain-sway:whp-roll:zlp-sway:wlp-sway:gain-roll:whp-pitch:gain-ya
        w:gain-surge:zhp-yaw:whp-yaw:wlp-surge:zlp-surge:whp-surge:zhp-surge:w
        hp-sway:zhp-sway:whp-heave:zhp-heave:gain-heave:tilt_rate_limit-surge:
        tilt_rate_limit-sway;gain-roll:whp-pitch:gain-surge:wlp-surge:zlp-surg
        e:gain-sway:gain-pitch:gain-yaw:whp-roll:zhp-yaw:zlp-sway:wlp-sway:whp
        -yaw:whp-surge:zhp-surge:whp-sway:zhp-sway:whp-heave:zhp-heave:gain-he
        ave:tilt_rate_limit-surge:tilt_rate_limit-sway;gain-yaw:zhp-yaw:gain-r
        oll:gain-sway:whp-pitch:whp-yaw:gain-pitch:gain-surge:whp-roll:zlp-swa
        y:wlp-sway:wlp-surge:zlp-surge:whp-surge:zhp-surge:whp-sway:zhp-sway:w
        hp-heave:zhp-heave:gain-heave:tilt_rate_limit-surge:tilt_rate_limit-sw
        ay;42.253902:1.315877:7.410963:4.112922:39.419941:0.181099;0.000000;0.
        915272:0.000000:0.073074:0.074963:0.000000:0.004373:0.860723:0.000408:
        0.000000:0.109375:0.000000:0.035864:0.074348:0.000000:0.025295:1.00000
        0:0.015139:0.000000:0.164193:0.300465:0.000000:0.497061:0.198348:0.000
```

```
000:0.000000:0.000000:0.000000:0.127713:0.773247:0.022613:0.000000:0.5
56287:0.000000:0.000000:0.000000:0.000000:0.009738:0.209930:0.000622:0
.000000:0.293710:0.594727:1.000000;


How was the motion this time compared to last time ?
(Use one of the following list)
(improved no_change worse)
Enter a hedge (One of the following list)
(much somewhat)
Your Response :
much improved
Is this OK (yes or no)
** Still tuning motion_jerkiness on pitch **
** Last complaint you made was: very small **
Enter Term (One of following):
(large medium small ok exit_and_undo)
Your Response :
ok motion_jerkiness
Is this OK ? (yes or no)

** Parameter Identified as :whp-pitch **


*** Forward reasoning for primary parameter ***

** Warning in convert_to_set_binary : value < 1 **
if tune_parameter(whp, pitch) and whp(very_very, very_small, [1.0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0.313483) ~ pitch and (coupled_dof(
        pitch, surge) or surge = pitch) and wlp(_38, _39, _40, 2.254) ~
        surge and 0.139079 is 0.313483 / 2.254
then
whp_wlp(very_very, very_small, [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        0.139079) ~ pitch in default

if pilot_response([much, improved]) and (improved \= minimized) and (improved
        \= no_change)
then
pilot_response(much, improved, [1.0, 0.9801, 0.9409, 0.8464, 0.7056, 0.49,
        0.25, 0.09, 0.0256, 0.0064, 0.0009, 0.0001, 0.0], 3.45053) in
        primary

if not simulator_limit_exceeded and last_adjustment(, medium-ve, [0.0,
        0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0], 0.611079) ~ whp ~ pitch and pilot_response([much, improved])
        and less_than_one(0.611079)
then
adjust_direction(reduce) in primary

if simulator_motion(_41, _42, _43, 4.11292) ~ roll and simulator_motion(_44,
        _45, _46, 39.4199) ~ pitch and simulator_motion(_47, _48, _49,
        0.181099) ~ yaw and simulator_motion(_50, _51, _52, 42.2539) ~
        surge and simulator_motion(_53, _54, _55, 1.31588) ~ sway and
        simulator_motion(_56, _57, _58, 7.41096) ~ heave and 94.6947 is
```

```
        4.11292 + 39.4199 + 0.181099 + 42.2539 + 1.31588 + 7.41096
 then
total_motion(94.6947) in default

 if tune_parameter(whp, pitch) and (whp \= wlp) and (whp \= tilt_rate_limit)
        and ((whp \= gain) or (pitch \= surge) and (pitch \= sway)) and
        total_motion(94.6947) and simulator_motion(_59, _60, _61, 39.4199)
        ~ pitch and 41.6284 is 39.4199 / 94.6947 * 100.0
 then
relative_motion(very_very, medium, [0, 0, 0, 0, 0.004608, 1.0, 0, 0, 0,
        0, 0, 0, 0], 41.6284) ~ whp ~ pitch in default

 if tune_parameter(whp, pitch)
 then
maneuver_importance and relative_motion ~ whp ~ pitch -> dof_importance
        ~ whp ~ pitch in primary
dof_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch

 if tune_parameter(whp, pitch) and tuning_dof(pitch)
 then
parameter_importance ~ whp ~ pitch ~ pitch and dof_importance ~ whp ~ pitch
        -> new_importance ~ whp ~ pitch in primary
new_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch

 if not m_adjustment(_62, _63, _64, _65) ~ whp ~ pitch and last_adjustment(,
        medium-ve, [0.0, 0.05, 0.635854, 1.0, 0.2, 0.05, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 0.611079) ~ whp ~ pitch and less_than_one(0.6110
        79)
 then
parameter_direction(decrease) in primary

 if parameter_direction(decrease) and tune_parameter(whp, pitch)
 then
max_adjustment(very, large-ve, [1.0, 0.25, 0.04, 0.0025, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.416273) ~ whp ~ pitch in primary

 if parameter_direction(decrease) and tune_parameter(whp, pitch) and (whp
        = whp or whp = zhp or whp = tilt_rate_limit or whp = wlp)
 then
current_tuning_direction(increasing_motion) in primary

 if pilot_complaint([, ok, motion_jerkiness]) and tune_parameter(whp, pitch)
 then
new_adjustment(very_very, zero, [0, 0, 0, 0, 0, 0, 1.0, 0.0, 0, 0, 0, 0,
        0], 1.0) ~ whp ~ pitch in primary

 if pilot_complaint([, ok, motion_jerkiness]) and tune_parameter(whp, pitch)
        and make_pred(whp, [very_very, very_small, [1.0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0], 0.313483], whp(very_very, very_small, [1.0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0.313483)) and whp(very_very,
        very_small, [1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0.313483)
        ~ pitch and maneuver(quick_stop) and pilot_axis(pitch) and original_v
```

```
        alue(whp, pitch, 0.513) and (0.513 >= 0.313483 and decreasing =
         decreasing or 0.513 < 0.313483 and decreasing = increasing)
 then
current_temp_tuned(whp, pitch, 0.313483, quick_stop, motion_jerkiness,
         pilot_axis(pitch), pilot_satisfied, na, decreasing) in primary

 if pilot_complaint([, ok, motion_jerkiness]) @ primary
 then
stop_tuning_parameter in primary

 if (pilot_complaint([, ok, motion_jerkiness]) or pilot_complaint_ok_replaced)
         and not primary_loop_parameter_interaction_bypassed(_66)
 then
no_secondary_tuning in primary


 ** suggested adjustment ** : whp DOF: pitch multiply by: 1.0

 Is this ok (char + return) or enter new value (float)
You have chosen: o
 ** actual adjustment ** : whp DOF: pitch multiply by: 1.0


 ** finished primary tuning **


 ** Firing STOP CONDITIONS Expert System **

 if not tune_parameter(_67, _68) @ secondary and current_temp_tuned(whp,
         pitch, 0.313483, quick_stop, motion_jerkiness, pilot_axis(pitch),
         pilot_satisfied, na, decreasing)
 then
best_temp_tuned(whp, pitch, 0.313483, null, null, null, quick_stop, motion_jer
        kiness, pilot_axis(pitch), pilot_satisfied, na, decreasing, null,
         null, null, null, null) in default

 if pilot_complaint([, ok, motion_jerkiness]) or pilot_complaint_ok_replaced
         or best_temp_tuned(_69, _70, _71, _72, _73, _74, _75, _76, _77,
         pilot_satisfied, _78, _79, _80, _81, _82, _83, _84)
 then
stop_tuning_problem in default

 if stop_tuning_problem and best_temp_tuned(whp, pitch, 0.313483, null,
         null, null, quick_stop, motion_jerkiness, pilot_axis(pitch), pilot_sa
        tisfied, na, decreasing, null, null, null, null, null) and (pilot_sati
        sfied \= exit_and_undo)
 then
keep_tuning_and_eliminate(whp, pitch, 0.313483, null, null, null) in default

 if keep_tuning_and_eliminate(whp, pitch, 0.313483, null, null, null) and
         not total_importance(_85, _86, _87, _88) ~ whp ~ pitch and new_import
        ance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0, 0.0,
         0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch
 then
```

```
new_total_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch in default

 if keep_tuning_and_eliminate(whp, pitch, 0.313483, null, null, null) and
        best_temp_tuned(whp, pitch, 0.313483, null, null, null, quick_stop,
        motion_jerkiness, pilot_axis(pitch), pilot_satisfied, na, decreasing,
        null, null, null, null, null) and new_total_importance(, medium,
        [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0], 3.33663) ~ whp ~ pitch
 then
tuned(whp, pitch, 0.313483, primary, quick_stop, motion_jerkiness, pilot_axis(
        pitch), pilot_satisfied, na, decreasing, na, na, na, na, na, total_imp
        ortance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0], 3.33663)) in default

 if keep_tuning_and_eliminate(whp, pitch, 0.313483, null, null, null)
 then
tuned_dof(pitch) in default

 if (tune_parameter(whp, pitch) @ primary or tune_parameter(whp, pitch)
        @ secondary) and tuned(whp, pitch, 0.313483, primary, quick_stop,
        motion_jerkiness, pilot_axis(pitch), pilot_satisfied, na, decreasing,
        na, na, na, na, total_importance(, medium, [0.0, 0.03, 0.2,
        0.5, 0.8, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 3.33663))
 then
update_number(previously_tuned(whp, pitch, 1)) in default

 if new_total_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8, 1.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 3.33663) ~ whp ~ pitch and best_temp_t
        uned(whp, pitch, 0.313483, null, null, null, quick_stop, motion_jerkin
        ess, pilot_axis(pitch), pilot_satisfied, na, decreasing, null,
        null, null, null, null)
 then
replace_fact(total_tuned(whp, pitch, _89, _90), total_tuned(whp, pitch,
        0.313483, total_importance(, medium, [0.0, 0.03, 0.2, 0.5, 0.8,
        1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 3.33663))) in default


 ** ADJUSTMENT **  PARAMETER: whp DOF: pitch multiply by: 1.0


 *** retaining tuning and eliminating *** whp-pitch


 ** finished tuning current problem **


 c[ontinue], e[xit](saving state), a[bort]

 Enter pilots initials
 Closed Socket Connection, handle = 8
 Closed Socket Connection, handle = 6
```

APPENDIX C

PROTEST USER MANUAL

This appendix will give the instructions on how to use PROTEST. It does not provide any information on Prolog programming or how to add rules to the tuning database. This information can be obtained in Grant [7]. These instructions are for the system as it now functions in the UTIAS Flight Simulation Laboratory.

Host

1. Load the host code.
2. Start the simulation with the 'simnrt' command.
3. Set the desired initial conditions, including the option to run PROTEST.
4. When prompted choose IRIS5 as the computer for PROTEST.

The host will now wait for ethernet communication to be established with IRIS5 and PROTEST. Move to IRIS5 and follow the directions below. Note that once Prolog is started any commands entered must end with a period.

1. On IRIS5 change into the directory with the PROTEST files.
2. Start Prolog by typing **Prolog** (If a log of the run is desired pipe the output to a file by typing Prolog |tee <filename>
3. Load the PROTEST code by typing: **consult('protest').**
   followed by: **load_protest.**
4. A message will appear stating that PROTEST files are being loaded. PROTEST will then prompt whether or not debugging is required. Choose yes if the output is wanted.
5. At the prompt now type: **tune.**

6. Choose the manoeuvre that is being tuned, or type all for a complete tuning.
7. PROTEST now prompts for an interactive or recorded run. Type the desire option.
8. PROTEST prompts for a new or restarted run. Type: **new.** if it is the beginning of a run or type the file name in quotations to continue from a previous run.
9. PROTEST will now establish communication with the host computer. The sending socket is created first. You must press enter when prompted by the host to then establish the receiving socket.
10. The initial set of coefficients is then sent and the pilot is prompted to fly a manoeuvre.

At this point the run proceeds normally. The pilot stabilises the aircraft and informs the controller when the manoeuvre begins. The control must then start the data analysis procedure by choosing run control option 14. When the manoeuvre is complete the data collection is stopped by choosing run control option 15. When the run is complete run control option 0 is chosen. The controller is asked if the run is valid. A yes answer will cause data to be sent to PROTEST. At this point the controller will return to IRIS5 and proceed will proceed in answering the questions posed by PROTEST.

1. Choose a PROBLEM from the given list.
2. Choose the appropriate magnitude of the PROBLEM.(Note: the magnitude option that appear as an adjective and sign such as 'large+ve' must be entered in quotes)
3. Choose the degree of freedom for the PROBLEM.
4. Wait for PROTEST to produce the adjusted variable

Once this complete the adjusted coefficient is sent back to the host who will ask if tuning is to continue and if another run is desired. At this point another run can be completed and the cycle begins again. Detailed information on how to use PROTEST is given by Grant [7].

REFERENCES

1. L. D. Reid and M. A. Nahon. Simulator Motion-Drive Algorithms: A Designer's Perspective. AIAA Journal of Guidance, Control, and Dynamics, 13(2): 356-362, March-April 1990.

2 E. Tai. A Preliminary Evaluation of a Synthetic Vision System for Helicopter Search and Rescue Operations. M.A.Sc. Thesis, University of Toronto, 1998.

3. P. R. Grant. Motion Characteristics of the UTIAS Flight Research Simulator Motion-Base. Technical Note 261, UTIAS, 1986.

4. L. D. Reid and M. A. Nahon. Flight Simulation Motion-Base Drive Algorithms: Part 1 - Developing and Testing the Equations. Report 296, UTIAS, 1985.

5. L. D. Reid and M. A. Nahon. Flight Simulation Motion-Base Drive Algorithms: Part 2 - Selecting the System Parameters. Report 307, UTIAS, 1986.

6. Bernard Etkin. Dynamics of Atmospheric Flight. John Wiley and Sons, Inc., 1972

7. P. R. Grant. The Development of A Tuning Paradigm for Flight Simulator Motion Drive Algorithms. PhD. Thesis, University of Toronto, 1996.

8. A. Walker, M. McCord, J. F. Sowa,and Wilson. Knowledge Systems and Prolog. Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

9 G. L. Greig. Masking of Motion Cues by Random Motion: Comparison of Human Performance with Signal Detection Model. Report 313, UTIAS, January 1988.