Graduate Theses, Dissertations, and Problem Reports

1999

Regressive model approach to the generation of test trajectories

Brian James Taylor
West Virginia University

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Taylor, Brian James, "Regressive model approach to the generation of test trajectories" (1999). *Graduate Theses, Dissertations, and Problem Reports*. 996.
<https://researchrepository.wvu.edu/etd/996>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Regressive Model Approach to the Generation of Test Trajectories

by
Brian J. Taylor
B.S.EE., B.S.Cp.E, West Virginia University

Thesis

Submitted to
the Department of Computer Science and Electrical Engineering of
the College of Engineering and Mineral Resources
at

WEST VIRGINIA UNIVERSITY
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering

Dr. Bojan Cukic, Ph.D., Chair
Dr. Marcello Napolitano, Ph.D.
Dr. Stephanie Schuckers, Ph.D.

Morgantown, West Virginia
1999

Keywords: Test Data Generation, Software Testing, Software Reliability
Copyright © 1999 Brian J. Taylor

Abstract

Regressive Model Approach to the Generation of Test Trajectories

Brian J. Taylor

One possible method of reliability assurance is the application of large testing sets upon a system via statistical reliability assessment. Under this method, a software system's operations are tested according to their actual usage. A problem with this approach is the large number of tests required to properly assess reliability. Complicating the process are situations where an input into the system is a trajectory of data, where a trajectory is defined as a series of data points with each point relying upon the previous points. Examples of trajectories of data include an airplane's flight path or data describing a nuclear reaction. Whereas test data generation has addressed the creation of individual test points, creating test trajectories have yet to be adequately studied.

This thesis focuses upon a trajectory generation algorithm using statistical regression models. The algorithm is developed to aid reliability assessment by increasing the size of the test data. An existing small set of trajectories is utilized to build the regressive models. Then, the independent variables of the model can be perturbed, thus allowing the generation of new trajectories.

A case study is conducted for a sensor failure detection, identification, and accommodation (SFDIA) flight control scheme. Our test trajectory generation algorithm uses test data collected from a flight simulator of a Boeing 777. The regressive models speed up the trajectory generation process by more than three orders of magnitude when compared with flight data collection.

Acknowledgements

I would like to thank each of the members of my committee for their help and guidance while writing this thesis. First, I want to thank Dr. Stephanie Schuckers for her help with signal processing principles. Next is a very important thanks to Dr. Marcello Napolitano for borrowing some of his knowledge on flight control systems and for giving me my limited experience into aerospace fundamentals. And finally thanks goes to Dr. Bojan Cukic, my committee chairman and advisor throughout this research. Without his guidance none of this would be possible.

The members of the Aerospace Engineering Lab have my appreciation for their help with the SFDIA approach and in using the AVDS system including Diego Del Gobbo, Brad Seanor and Stelios Pispitsos.

I would also like to recognize my friends and family who endured with me as I undertook this work. A huge gratification goes to the entire community of the Microelectronics Research Center (MSRC) with special emphasis for Paul Sines. His chess board provided a wonderful way to escape the stress of endless matlab programming.

Another of my strong supporters is my best friend Kolin Brown. A special thanks is reserved for him for his continued support and presence. Whether he knows it or not, I will never forget that.

Finally, thank you mom and dad. Without your support all throughout my life, I would not be where I am nor would I have accomplished what I have.

Contents

Abstract	ii
Acknowledgments	iii
Table of Contents	iv
List of Tables	viii
List of Figures	x
1 Introduction	1
1.1 Testing for Software Reliability	1
1.1.1 Input Domain Testing	2
1.1.2 Massive Testing for Adequate Reliability	3
1.2 Trajectories of Data	7
1.3 A Trajectory Generation Algorithm	9
1.4 Case Study	10
1.5 Outline of Thesis	11
2 Test Data Generation	12
2.1 Testing History	12
2.2 Test Data Generators	13
2.2.1 Pathwise Test Data Generators	14
2.2.2 Data Flow Models	17
2.2.3 Data Specification Systems	19
2.2.4 Random Test Generators	20

2.2.5 Adaptive Testing	20
2.2.6 Regression Testing	22
2.2.7 Statement Testing	22
3 Regressive Models	24
3.1 Simple Linear Regression	25
3.1.1 Regression Coefficients	27
3.2 Multiple Linear Regression	28
3.2.1 Regression Coefficients	29
3.3 Parametric Modeling	32
3.4 Non-Linear Regression	37
4 Trajectory Generation	40
4.1 Algorithm Overview	42
4.2 Collect Data	44
4.3 Process Data	45
4.4 Clustering the Data	45
4.4.1 Parameter Selection	48
4.4.2 Outliers	48
4.4.3 Scaling the Data	49
4.4.4 Distance Metrics	50
4.4.5 Clustering Techniques	52
4.4.6 Interpreting the Results	53
4.4.7 Selecting a Representative Trajectory	54

4.5 Developing a Model	54
4.5.1 Discriminant Analysis	55
4.5.2 Smoothing	56
4.5.3 Cross-correlation Analysis	58
4.5.4 Choosing the Best Model	59
4.6 Perturbing the Original Data	59
4.7 Application of the Model	61
4.8 Acceptability Rules	62
5 Case Study	64
5.1 Tools	68
5.1.1 Aviator Visual Design Simulator (AVDS)	68
5.1.2 Matlab 5.0	70
5.2 Processing the Data	70
5.3 Clustering the Data	72
5.4 Building the Regressive Models	74
5.4.1 Correlation Results	77
5.5 Generating New Trajectories	86
5.6 Acceptability Tests	94
5.6.1 Pilot Input Acceptability	94
5.6.2 Angular Rate Acceptability	95
5.6.3 Acceptability Results	95
5.6.4 Trajectory Generation Time	97

6 Analysis of the Testing Approach	99
6.1 Model Development	99
6.2 Handling Zero Inputs	102
6.3 Acceptability Rates	103
7 Conclusions	105
7.1 Case Study	105
7.2 Further Work	106
Bibliography	108
Appendix A Linear Regression Model Correlations	112
Appendix B ARMA Regression Model Correlations	119

List of Tables

1.1 Required Test Cases U for Various Confidences and Failure Rates	7
2.1 d-u Occurrences in Example Program	18
5.1 Linear Model Construction	75
5.2 ARMA Model Construction	76
5.3 Average Time Required for Linear Model Development	81
5.4 Selected Linear Models for Each Test Run	82
5.5 Average Time Required for ARMA Model Development	85
5.6 Selected ARMA Models for Each Test Run	86
5.7 Percentage of Acceptable Trajectories for the Linear Models, Tests 1-9	95
5.8 Percentage of Acceptable Trajectories for the Linear Models, Tests 1-9	96
5.9 Percentage of Acceptable Trajectories for ARMA Models, Tests 1-9	96
5.10 Percentage of Acceptable Trajectories for ARMA Models, Tests 1-9	96
6.1 Individual Trajectory Correlations for Test Run 11	100
A.1 Average Roll Rate Linear Model Correlations for Test Maneuvers 1-9	113
A.2 Average Roll Rate Linear Model Correlations for Test Maneuvers 10-17	114
A.3 Average Pitch Rate Linear Model Correlations for Test Maneuvers 1-9	115
A.4 Average Pitch Rate Linear Model Correlations for Test Maneuvers 10-17	116
A.5 Average Yaw Rate Linear Model Correlations for Test Maneuvers 1-9	117
A.6 Average Yaw Rate Linear Model Correlations for Test Maneuvers 10-17	118
B.1 Average Roll Rate ARMA Model Correlations for Test Maneuvers 1-9	120

B.2 Average Roll Rate ARMA Model Correlations for Test Maneuvers 10-17	121
B.3 Average Pitch Rate ARMA Model Correlations for Test Maneuvers 1-9	122
B.4 Average Pitch Rate ARMA Model Correlations for Test Maneuvers 10-17	123
B.5 Average Yaw Rate ARMA Model Correlations for Test Maneuvers 1-9	124
B.6 Average Yaw Rate ARMA Model Correlations for Test Maneuvers 10-17	125

List of Figures

1.1 Mapping the Input Domain into the Output Domain	3
2.1 Directed Graph of a Function	15
2.2 Adaptive testing Layout	21
3.1 Linear Regression	25
3.2 ARMA Transfer Function	36
3.3 Non-linear Regression	38
4.1 Mapping of Controllable Variables to Input Domain Trajectories	41
4.2 Generation Algorithm Layout	42
4.3 Example of Clustering with Trajectories	47
4.4 Example of a Trajectory Outlier	53
4.5 Application of an Averaging Filter	56
4.6 Application of a Triangular Filter	57
4.7 Demonstration of Multiplying and Input by a Constant	60
4.8 Demonstration of Increasing the Duration of an Amplitude	61
5.1 Analytical Approach to Fault Tolerant Flight Control	64
5.2 SFDIA Layout	65
5.3 Recorded Pilot Inputs and Respective Angular Rates	69
5.4 Pilot Input-Angular Rate Observations	71
5.5 Pilot Input-Angular Rate Observations	72
5.6 Trajectories Before Clustering	73

5.7 Trajectories After Clustering	74
5.8 Roll Rate Simple Linear Models for Test Run 8	77
5.9 Roll Rate Multiple Linear Models for Test Run 8	78
5.10 Pitch Rate Simple Linear Models for Test Run 8	78
5.11 Pitch Rate Multiple Linear Models for Test Run 8	79
5.12 Yaw Rate Multiple Linear Models for Test Run 8	79
5.13 Roll Rate ARMA Model for Test Run 8	83
5.14 Pitch Rate ARMA Model for Test Run 8	83
5.15 Yaw Rate ARMA Model for Test Run 8	84
5.16 Original Roll Rate Trajectory Versus Linear Generated Trajectory	88
5.17 Original Pitch Rate Trajectory Versus Linear Generated Trajectory	89
5.18 Original Yaw Rate Trajectory Versus Linear Generated Trajectory	90
5.19 Original Roll Rate Trajectory Versus ARMA Generated Trajectory	91
5.20 Original Pitch Rate Trajectory Versus ARMA Generated Trajectory	92
5.21 Original Yaw Rate Trajectory Versus ARMA Generated Trajectory	93

Chapter 1

Introduction

1.1 Testing for Software Reliability

As software finds its way into safety-critical applications, users of these systems need to be assured of failure-free execution. Failures in non-critical systems may result in loss of work or inconvenience, but in systems where high reliability is a requirement, failures can result in loss of human life.

As [12] points out, software, in theory, unlike hardware, can be made free of defects since all defects are present at the completion of the software system. These defects need only be removed by rigorous and systematic testing. However, 100% removal of defects is a very elusive goal. Since no one has yet produced a piece of software which can claim to have zero faults, a measure of its reliability becomes one of the few indicators of how much reliance a user can place in a system to perform as intended without experiencing any failures.

Reliance in a system can increase through testing but, as [21] points out, the number of test cases required in order to engender confidence in a piece of software is usually extremely large. For example the safety requirements for civilian airliners are to have a failure rate of less than $10^{-9}/h$, and nuclear power plant monitoring systems are required to

experience less than 10^{-4} failures per hour of operation [12]. The amount of testing that a system would have to undergo to ensure those failure rates is very impractical as they represent thousands of years of usage.

Understanding that exhaustive testing can never be completed, researchers must instead use statistical methods to assess the reliability of safety-critical systems. [21] explains that reliability estimation requires statistically valid random testing with a planned distribution of the tests throughout the input domain. The distribution of tests should resemble the typical operational usage which a software system faces upon deployment, with a possible strong emphasis in regions where criticality is the most important.

Software reliability is defined as the probability of not encountering (a sequence of) inputs that lead to a failure. But, due to a lack of knowledge of what input sequences cause failures, the only way to measure the frequency of failures is to subject the system to tests.

1.1.1 Input Domain Testing

One method of determining the reliability of a system is input domain based reliability assessment where a software system's operations are tested according to their usage in the field [5]. Using this assessment technique, the *reliability* of a program is the probability of failure free operation over the specific time interval. The *input domain* I of a *program* P is represented by test cases. Program P is treated as a function which maps the elements of the multidimensional space I into an *output space* O . The inputs that activate faults in the program are mapped into failures, i.e. incorrect outputs. Figure 1.1 shows the testing paradigm for input domain reliability assessment.

Figure 1.1 Mapping the Input Domain into the Output Domain

1.1.2 Massive Testing for Adequate Reliability

If n represents the total number of test cases applied to the program P and n_f the number of detected failures, the estimated reliability R , according to the Nelson model is

$$R = \frac{n - n_f}{n} = 1 - \frac{n_f}{n}. \quad (1.1)$$

As the number of tests becomes infinite, the equation is rewritten as

$$R = 1 - \lim_{n \rightarrow \infty} \frac{n_f}{n}. \quad (1.2)$$

The fraction n_f/n in the above limit is usually denoted by θ , which represents the estimated

probability of failure in a single run of the program. This leads to the reliability prediction of the program based on the probability of correct execution in each run. The probability of correct execution over i runs is given by

$$R(i) = (R)^i. \quad (1.3)$$

In the formula above, it is assumed that the inputs are selected independently according to the same probability distribution used to choose the random tests.

A *profile* is defined as a set of disjoint alternatives called *elements* where each element has a probability that it will occur. A *run* is defined as a task which represents the smallest division of work that can be initiated by external intervention. A *run type* is then a description of runs which have identical input states. An *operation* is a set or grouping of run types for the system as built. Run types are grouped because most systems have large amounts of run types and it would be impractical to look at individual run types. The *operational profile* is then a profile of operations.

By looking at the operational profile for a system, the probability of failure in a single run can be estimated by

$$\theta = \sum_{i \in I} p(i) \alpha(i), \quad (1.4)$$

where $p(i)$ is the probability that run i is selected from I , and $\alpha(i)$ is 1 if input i leads to a failure and 0 otherwise. Exhaustive testing can provide an exact assessment of the probability of failure, since

$$\sum_{i=1}^{|I|} p(i) = 1. \quad (1.5)$$

The goal of reliability assessment is not just to estimate the failure probability of the program, θ , but to gain the statistical confidence that θ is realistic. In practice, the required failure probability θ and the *confidence level* C are usually predefined, and testing is performed to corroborate the compliance with the reliability requirements using statistical methods. The question that arises is how much testing should be conducted?

Let T be a random variable denoting the number of test cases that are executed till the first failure is detected. To achieve a required confidence, an unknown number of test cases U needs to be executed such that

$$Prob(T \leq U) = C. \quad (1.6)$$

The distribution of T is geometric, and the probability that T assumes a specific value t is

$$Prob(T = t) = \theta(1 - \theta)^{t-1}, \quad t=1,2,\dots \quad (1.7)$$

A combination of the above two formulae describe a relation between U and C ,

$$\sum_{t=1}^U \theta(1 - \theta)^{t-1} = C. \quad (1.8)$$

The left hand side of the equation above is a geometric series

$$\sum_{k=0}^{N-1} x^k \quad (1.9)$$

and can be solved as follows:

$$\theta \sum_{t=0}^{U-1} (1 - \theta)^t = \theta \frac{(1 - \theta)^U - 1}{(1 - \theta) - 1}. \quad (1.10)$$

After algebraic simplification and combination of the equations (1.8) and (1.10),

$$(1 - \theta)^U = 1 - C, \quad (1.11)$$

and the required number of test cases U can be expressed as

$$U = \frac{\ln(1 - C)}{\ln(1 - \theta)}. \quad (1.12)$$

Table 1.1: Required Test Cases U for Various Confidences and Failure Rates.

θ	C = 90%	C = 95%	C = 99%	C = 99.9%
10^{-1}	22	28	44	66
10^{-2}	230	300	460	690
10^{-3}	2300	3000	4600	6900
10^{-4}	23000	30000	46000	69000
10^{-5}	2.3×10^5	3×10^5	4.6×10^5	7×10^5
10^{-6}	2.3×10^6	3×10^6	4.6×10^6	7×10^6
10^{-7}	2.3×10^7	3×10^7	4.6×10^7	7×10^7
10^{-8}	2.3×10^8	3×10^8	4.6×10^8	7×10^8
10^{-9}	2.3×10^9	3×10^9	4.6×10^9	7×10^9

As can be seen in Table 1.1, as θ approaches 0 and as C approaches 100%, the number of test cases U required to achieve the confidence that the assessed reliability is realistic increases rapidly. Since this number of tests are difficult to collect, analysts of a software system must turn to automated test data generators to create the test data for them.

1.2 Trajectories of Data

Reliability estimation of real-time systems is even more complicated as the inputs into these systems are frequently not the single valued data from an input domain. Examples of real-time systems may include closed-loop feedback systems, where data flow can consist of multiple inputs and multiple outputs. Instead, because a real-time system can simulate or replace an analog system, input into the system is actually a trajectory of data [21].

A *trajectory* is defined as a series of data points with each point of data dependent upon the previous point. Trajectories, then, give inputs as a function of time. A test trajectory for a real-time system would involve data streams which the system would use as inputs for the length of that trajectory. Common systems which use trajectories for input include robotics control, flight control software, speech analysis and nuclear monitoring systems [5].

The length of a test trajectory depends upon the system undergoing evaluation and the characteristics of the data. Trajectories may be so long that they make testing impractical, yet some systems can enter into states which are only achieved after a very long run time. Reliability estimates using short trajectories may not be valid on those systems.

If the system can be partitioned into states, the statistical testing could be conducted by creating short trajectories which would describe possible state transitions. The size of the trajectories would then be dictated by the number of states describing the system operations. By modeling the system as a state-machine, testing can be done as if the system were a batch program where trajectories act as input sequences for each individual state.

Determining the trajectories from the input domain must be done carefully so that chosen test trajectories can provide an accurate image of the field usage of the safety-critical real time system. The population from which trajectories are drawn must include the areas where the system will perform and, likely, take action [21]. If trajectories which do not occur in the field are used to test the system, reliability assessment of the system will not be accurate, and may be misleading.

1.3 A Trajectory Generation Algorithm

Whereas automated test data generation has addressed the creation of individual test points, creating test trajectories for the states of a software system has not been adequately studied. The research presented in this thesis focuses upon a test trajectory generation algorithm. It uses statistical models to aid reliability assessment by increasing the number of available tests through the expansion of an existing smaller set of testing trajectories. The generation system is designed for automated trajectory creation to aid in reduction of total testing time and to remove any objective biases system testers may have.

The algorithm consists of six individual modules, which can be modified depending on the function of the software system. Ideally, a small set of test data is collected, processed and clustered, and then passed into a module which develops a predictive regressive model. Suggested regressive models include simple linear, multiple linear, autoregressive moving-average (ARMA), and non-linear regression. Since all of these models require a dependent-independent variable relationship, where the dependent variable is the test trajectory being generated, the collected test data must include additional variables beyond the intended test trajectories. These additional variables will act as the independent variables which will drive the regression models.

After a model is developed, the original input trajectories are perturbed to develop new dependent variables from the predictive model. These result in artificially generated test trajectories, which, then, undergo acceptability checking to determine if the new test trajectories are valid. Acceptable trajectories can then be used in reliability assessment

testing.

The algorithm is designed with an emphasis on generality. Each module may be replaced or refined as the system undergoing trajectory generation changes. The algorithm is also simple enough that testers who may not have a detailed understanding of the equations which describe a system under test may still use it. Automation is a key aspect of the algorithm as it aids in the reduction of total test time.

1.4 Case Study

A test-case study is conducted for a sensor failure detection, identification, and accommodation (SFDIA) flight control scheme to be used on board commercial jetliners. Developed by the West Virginia University Department of Mechanical and Aerospace Engineering, the SFDIA scheme uses artificial neural networks to monitor and replicate roll, pitch, and yaw rate sensors should they fail during flight. Input into the neural networks include a trajectory of changing angular rates which an aircraft experiences during flight.

As a step to assess the reliability of the SFDIA scheme, our algorithm is used upon test data collected from an advanced flight simulator. Collected test data includes all pilot inputs, aerodynamic and euler angular rates, as well as other flight conditions such as velocity. These test trajectories provide for coverage across the operational profile of the SFDIA. When applied, the algorithm is able to generate roll, pitch, and yaw rates with close to 99% statistical correlation to the simulated angular rates.

1.5 Outline of Thesis

Chapter 2 looks at existing test generation techniques and how they are applied towards software reliability assessment. A brief overview of regression analysis is presented in Chapter 3. Chapter 4 presents a detailed description of the trajectory generation algorithm by covering the individual modules which make up the generator. The SFDIA flight control scheme is presented in Chapter 5 with an explanation as to how the generator is applied to the SFDIA. Analysis and discussion of the results from the case study are presented in Chapter 6 while Chapter 7 discusses conclusions which are drawn from the algorithm and the case study.

Chapter 2

Test Data Generation

Software testing involves executing a system with a test case and then analyzing the results for correctness. Testing has never been a glorious step to a system's development and only over the past decade has the field of software testing gained increased recognition for its significance to system design. Once software testing was looked upon as a last minute effort to ensure that the software works. Today testing is considered very early on in a system's development and integral to the overall design process.

2.1 Testing History

[24] clarifies a distinction between *static* and *dynamic* testing techniques. Static techniques investigate the code of a software system without executing the system itself. Instead inspection of the code, verification of the system, and symbolic execution are used to determine correctness of operation. The opposing viewpoint to static testing is dynamic testing. Dynamic techniques generate test data to analyze a system during execution. Once the data is generated, the system undergoes testing.

The difference between black-box and white-box testing is also seen in different software testing approaches. Black-box testing is performed upon a software system where

the internal workings of the system are unknown, only the definition of the inputs and outputs to the system are available. Common black-box approaches to testing include functional and specification-based tests [24]. These tests use only high level system requirements to generate the test data.

White-box testing looks at the design of the system by creating tests to study the internals of the program itself. Structural and code-based techniques are common to white-box test practitioners. White box testing is concerned with the lower level specifications of the system, particularly any modules or single lines of code.

2.2 Test Data Generators

A test data generator is defined in [4] as a tool which assists a user in the generation of test data. The purpose of the tool is to reduce the testing time by aiding a system developer in generating a large volume of test data. Several different types of test generators exist but three basic ones are random test data generators, pathwise generators and data specification systems. The creation of test data can be divided into automatic and manual techniques [26].

Automatic test data generation is a form of generation which can help to eliminate time and any subjective biases a developer might have in creating a system test. Automatic test data generation includes data specification systems, pathwise test data generators, random test data generators, and specification based test generation. Data specification systems generate test data from a language which describes the input data [16]. Pathwise test

data generators work to generate test data which follows an execution path throughout the program. Random test data generators are growing in importance because of the cleanroom process where the randomness is being confined to the important use cases. Specification based test data generation uses grammatical rules on the system specifications to generate test cases.

Manual test data generation includes cause-effect graphing, driven by coverage methods, equivalence partition, random user inputs, and use case analysis [26]. Cause-effect graphing is a graphical technique that maps the input domain to the output domain via true or false relationships. Driven-by-coverage-results methods generate test data with a focused purpose to increase the coverage numbers. Equivalence partitioning divides up the input domain into partitions and chooses test cases for each partition. Random user inputs, sometimes referred to as random keyboard pounding, is a process where users actually use the system giving as many random inputs as they can to simply use the system and determine any problems. Use case analysis is a process where use cases are easily transformed into test cases.

2.2.1 Pathwise Test Data Generators

Pathwise test generators are one of the more common generator techniques. A pathwise generator looks at creating test data that will exercise a certain path through the software system [4]. There are four steps to a pathwise generator: constructing a digraph

Figure 2.1 Directed Graph of a Function

representation of the program, selecting a path for a test to traverse, symbolically executing that path, and generating test data which will evaluate that path. Tests are selected from the input domain to cover as many of the program paths as desired.

The first step is to interpret the program into a directed graph showing the path between the different constructs such as conditions in the system. Once a path is chosen through the digraph, this path is symbolically executed which gives a predicate for the path that defines a region of the input space. A block of code and its associated directed graph are presented in Figure 2.1.

When a program is symbolically executed, the possible executions of the program are

characterized by an execution tree [4]. All input data and program variables are represented by values which may be purely symbolic or expressions. Execution is performed by a symbolic evaluator which interprets the symbols and produces output values consisting of symbolic formula or symbolic predicates. The outputs are then checked for correctness either by the system programmer or by a specially designed testing oracle. The result of the symbolic execution will be an equation in terms of input variables, which, if satisfied, will cause the symbolically executed path to be executed.

When a path in the program is to be tested, a test data generator chooses data from within each of the input domain regions. Sometimes a value cannot be found within the input regions. This indicates that either the path is unexecutable or the region was not properly formed.

Path predicates will either be linear or non-linear. If the path is linear, linear programming can be used by the generators to choose a value which will execute the path. If the path is non-linear, non-linear techniques are needed. [24] suggests that it is the problems associated with non-linear paths and unexecutable regions that have made for slow development of automatic test generation in the pathwise techniques. Consider a program which consists of several loops. A path in such a program could be of infinite length as well as non-linear. In response to these problems, testers sometimes apply the path-oriented models with a weaker criterion that only a subset of paths should be covered.

2.2.2 Data Flow Models

The data flow techniques were suggested by Rapps and Weyuker to solve some of the problems experienced by the path-oriented test generators [8]. In a data flow technique, test data is generated that follows a pattern of data use in a system, such as a variable as it is defined and used.

[23] developed three classifications that describe a variable as it's used in a program:

- *def*: definitional
- *c-use*: computational use
- *p-use*: predicate use

A definition of a variable is either when a variable is declared or when it is assigned a value.

A computational use of a variable occurs when the variable is used in an expression or to pass back a value to some calling function. A predicate use of a variable is where the variable is used as part of a conditional statement in the program and affects the control flow.

A control flow graph can be constructed for each variable where the edges are assigned *p-uses* and the nodes are assigned *defs* and *c-uses*. Along with the classification, Rapps and Weyuker also identify different test coverages that a test case can achieve for a system:

- all-nodes (complete statement coverage)
- all-edges (complete branch coverage)
- all-defs
- all-p-uses
- all-c-uses/some-p-uses
- all-p-uses/some-c-uses

- all-uses
- all-du-paths (complete definition and usage coverage p-use/c-use)
- all-paths

An example is presented to understand how the classification and test coverages can be combined to test the data flow. Suppose the following piece of code was undergoing data-flow testing:

```

1    int compute_factorial (int number) {
2        int factorial = 1;
3        int i = 1;
4        for (i = 1; i <= number; i++)
5            factorial = factorial * i;
6        return factorial;
7    }
```

Classifying the variable *factorial* by d-u paths, it is defined on line 2, defined and used on line 5, and used on line 7. Table 2.1 shows all of the d-u pairs.

Table 2.1 d-u pair occurrences in example program

d-u pair occurrence	d	u
1	2	5
2	2	6
3	5	5
4	5	6

The first d-u pair occurs after *factorial* is defined on line 2 and then used in line 5 should the

variable *number* be greater than 1. The second pair occurs after *factorial* is defined on line 2 and is used on line 6 if *number* is not greater than 1. Occurrence 3 happens on line 5 when *factorial* is both defined and used in the same statement. Finally, the fourth occurrence is after *factorial* is defined on line 5 and then used on line 6.

A test data generator working with a data-flow technique will identify the classifications within a program and generate test data to ensure coverage fitting the selected criteria. If we wanted to test the function *compute_factorial*, two tests will be required where *number* has a value of 3 and 0. If it assumes the value of 3, d-u pairs 1, 3, and 4 will be tested. If it assumes value 0, d-u pair 2 will be tested.

2.2.3 Data Specification Systems

Data specification systems work with a language that allows a system tester to describe the data files used in testing. The specification system can then generate a test case from the description. This generation technique is sometimes known as syntax-based testing and early study was conducted over 30 years ago. The automatic data generating program (ADG) is an example of such a system. Another type of specification based system can be found in [22].

One problem with a data specification system is that there will be classes of test data that will be impossible to describe or generate. Generation of large prime numbers is an example. Another problem is the tediousness for writing complex data file descriptions.

2.2.4 Random Test Generators

Random test generators are another common technique to generate test data. Random testing chooses tests randomly from across the input domain of the system without using a priori testing knowledge. The entire input domain of the system is considered as a whole and the test generator randomly selects inputs from this domain [28]. As long as the input domain is adequately defined, tests can be selected with a uniform distribution or any other desired distribution [4].

Random testing tends to uncover the easy faults and bugs. The only requirement for random testing is the use of a random number generator and some software that automates the testing. Random testing works very well on single value data but is unusable for trajectory generation. A method for random trajectory generation has been suggested using state-based machines and decision tree methods.

2.2.5 Adaptive Testing

The technique of adaptive testing generates new test data based upon the effectiveness of the last test data used on a system. This continues until an upper limit on test effectiveness is achieved. The architecture of an adaptive testing environment is shown in Figure 2.2.

Figure 2.2 Adaptive Testing Layout

Initially, some test data is generated either manually or automatically. The test data is used by a test driver to simulate the system environment and perform the test. Some methods test the overall software system, while other variations test individual paths or branches. The results of the test pass into an evaluator that might be a testing oracle indicating the success of the test. The test evaluation along with the test data are then used by an adaptive test generator which can modify the test data and perform a new test. The evaluation serves as a guide for the mutation of the test data. If a conditional statement is tested for a true assertion the first time, the adaptive generator can create new data which will allow for a test of a false assertion next time. Adaptive testing usually continues until total coverage is attained.

2.2.6 Regression Testing

Regression testing is a generation technique similar to adaptive testing. When a system undergoes testing, a large number of test cases are developed for assessment. If maintenance is performed on the system due to discovery of faults from these tests, or perhaps due to faults discovered during usage, tests will need to be re-applied to the system once again. While some functions will likely experience change, much of the original system functionality should remain unchanged. One approach is to apply a whole new set of test cases to the system. But this is an expensive process.

Regression testing can eliminate much of this overhead by looking at the changes made to the system and determining which tests need to be executed again. Regression test data generation occurs when existing test data is modified or rearranged for use to test only the changes made to the system.

2.2.7 Statement Testing

The goal of statement testing is to generate test data to execute every source language statement in the software system at least once. This is sometimes referred to as statement coverage. A problem with statement testing is that while it attempts to execute every conditional statement, some of those statements may be too difficult to include in the coverage. If this happens, a solution is to rigorously inspect those routines to eliminate them as being problems during use.

Statement testing can uncover unreachable code, code which cannot be reached

during execution by any data path. If certain sections of the system are unreachable, then a poor design flaw is uncovered and can be removed. Generation of test data for statement testing can be done using a technique called input analysis which generates combinations of input test data [24].

Chapter 3

Regressive Models

Regression offers the ability to predict the behavior of a variable based upon the behavior of one or more variables. *Regression analysis* is the process used to find the formula for predicting this behavior [27].

There are several different types of regressive models including simple linear regression, multiple linear regression, and non-linear regression models. It is important to note that *linear regression* is a model where the parameters occur linearly whether or not the independent variables enter the model linearly. For example, a polynomial regression model, such as

$$\hat{y} = b_0 + b_1x^1 + b_2x^2 + \cdots + b_kx^k, \quad (3.1)$$

is still a linear regressive model as long as each y is computed using the next step in x .

A very important point to remember when using regression for trajectory generation is that we are assuming that the independent variable(s) are *mathematical variables*, measured quantities with no probability distribution properties while the dependent variable, the trajectory itself, is a *random variable*. A random variable is defined as a function that associates a real number with each element in the sample space [27].

3.1 Simple Linear Regression

Simple linear regression deals with a single independent regressor variable X and a single dependent random variable Y . Since we are dealing with trajectories, both variables X and Y are treated as pairs of observations $\{(x_i, y_i); i = 1, 2, \dots, n\}$. Explained simply, the goal of linear regression is to plot the observations of X and Y against each other and to determine a best straight line fit which can be used to ‘predict’ the behavior of Y given an X intercept [15]. Figure 3.1 represents this visually.

Figure 3.1 Linear Regression

Linear regression analysis is the procedure of determine how to fit this line to the set of observed data. The model can improve if a gaussian white noise signal, e , is introduced. The white noise has zero mean and random distribution. The white noise will add a random

chance that a prediction of Y will be above or below the fitted line based upon an observed X. The formal approach follows.

If the random variable Y is defined such that $Y_i = Y|x_i$ then we could say that all means $\mu_{Y|x_i}$ fall on a straight line with each Y_i described by the simple linear regression model:

$$Y_i = \mu_{Y|x_i} + E_i = \alpha + \beta \cdot x_i + E_i \quad (3.2)$$

where E_i is the random error difference between the actual value of Y_i and the corresponding value on the straight line fit. The random error, E_i , should have a mean of zero because of the use of a best fit line for the means. Each observation pair (x_i, y_i) then follows the equation:

$$y_i = \alpha + \beta x_i + \varepsilon_i \quad (3.3)$$

where ε_i is the value for each E as y_i is the value for each Y. Because of the assumption that E was zero-mean, ε can be considered a gaussian white-noise signal for all i.

Consider that Equation 3.3, a statistical model describing the random variable Y_i , an estimated line of this statistical model or fitted regression line

$$\hat{y} = a + bx, \quad (3.4)$$

can be used where the observation pairs satisfy the relation

$$y_i = a + bx_i + e_i, \quad (3.5)$$

where $e_i = y_i - \hat{y}_i$ is called the residual error and can help determine the fit of the estimated line.

3.1.1 Regression Coefficients

The choice of the regression coefficients a and b is done to minimize the residual error of the estimation by using the method of least squares. If the sum of squares of the errors is denoted by SSE, then the following relation is defined

$$SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - a - bx_i)^2 \quad (3.6)$$

If both sides are differentiated with respect to a and b ,

$$\frac{\partial(SSE)}{\partial a} = -2 \sum_{i=1}^n (y_i - a - bx_i) \quad (3.7)$$

$$\frac{\partial(SSE)}{\partial b} = -2 \sum_{i=1}^n (y_i - a - bx_i)x_i \quad (3.8)$$

If the partial derivatives are set to zero the *normal equations* can be obtained

$$na + b \sum_{i=1}^n x_i = \sum_{i=1}^n y_i \quad (3.9)$$

$$a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i \quad (3.10)$$

which will allow for the solutions to a and b:

$$b = \frac{n \sum_{i=1}^n x_i y_i - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2} \quad (3.11)$$

$$a = \frac{\sum_{i=1}^n y_i - b \sum_{i=1}^n x_i}{n} \quad (3.12)$$

3.2 Multiple Linear Regression

In situations where a trajectory is too complex to be defined by a single-independent -single-dependent variable relation, multiple linear regression may be employed. Multiple linear regression deals with a multiple independent regressor variables and one single dependent random variable relationship. For the case of k independent variables $x_1, x_2, \dots, x_k$, the mean of the random variable Y given the independent variables is

$$\mu_{Y | x_1, x_2, \dots, x_k} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k \quad (3.13)$$

and the estimate response which closely resembles the simple-linear approach is given by

$$\hat{y} = b_0 + b_1 x_1 + \dots + b_k x_k \quad (3.14)$$

where y is the output signal and x_i is the i^{th} input signal.

3.2.1 Regression Coefficients

The coefficients for the multiple linear model, $\beta_1, \beta_2, \beta_k$ are obtained by fitting the least square estimate

$$\mu_{Y | x_1, x_2, \dots, x_k} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k \quad (3.15)$$

to the data points

$$(x_{1i}, x_{2i}, \dots, x_{ki}, y_i); \quad i=1, 2, \dots, n \text{ where } n > k \quad (3.16)$$

Each observation will then satisfy the equation

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \dots + \beta_n x_{ni} + \epsilon_i \quad (3.17)$$

which can be rewritten as the estimate model

$$y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + \dots + b_n x_{ni} + e_i \quad (3.18)$$

where ϵ is the random error and e , the residual error. Minimizing the sum of squares of the errors

$$SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - b_0 - b_1 x_{1i} - b_2 x_{2i} - \cdots - b_k x_{ki})^2 \quad (3.19)$$

If SSE is differentiated with respect to each coefficient $b_0, \dots, b_k$, and setting it to zero, the set of $k+1$ normal equations is generated:

$$nb_0 + b_1 \sum_{i=1}^n x_{1i} + b_2 \sum_{i=1}^n x_{2i} + \cdots + b_k \sum_{i=1}^n x_{ki} = \sum_{i=1}^n y_i \quad (3.20)$$

$$\begin{aligned}
& b_0 \sum_{i=1}^n x_{ki} + b_1 \sum_{i=1}^n x_{ki} x_{1i} + b_2 \sum_{i=1}^n x_{ki} x_{2i} + \cdots + b_k \sum_{i=1}^n x_{ki} x_{ki} = \sum_{i=1}^n x_{ki} y_i \\
& \quad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \\
& \quad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \\
& \quad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot \qquad \qquad \cdot
\end{aligned} \tag{3.21}$$

$$b_0 \sum_{i=1}^n x_{1i} + b_1 \sum_{i=1}^n x_{1i}^2 + b_2 \sum_{i=1}^n x_{1i} x_{2i} + \cdots b_k \sum_{i=1}^n x_{1i} x_{ki} = \sum_{i=1}^n x_{1i} y_i$$

These equations are best solved with matrix mathematics. By converting the above equations into matrices, the equations can be rewritten as

$$Y = X\beta + \varepsilon \quad (3.22)$$

where

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{21} & \cdot & \cdot & \cdot & x_{k1} \\ 1 & x_{12} & x_{22} & \cdot & \cdot & \cdot & x_{k2} \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ 1 & x_{1n} & x_{2n} & \cdot & \cdot & \cdot & x_{kn} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \cdot \\ \cdot \\ \beta_k \end{bmatrix} \quad (3.23)$$

If the estimation for β is used as in equation 3.19, then the least squares solution involves finding b for which

$$SSE = (Y - Xb)'(Y - Xb) \quad (3.24)$$

is solved by minimizing

$$\frac{\partial}{\partial b}(SSE) = 0 \quad (3.25)$$

which gives

$$(X'X)b = X'Y \rightarrow AB = C \rightarrow B = A^{-1}C \quad (3.26)$$

where

$$A = \begin{bmatrix} n & \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{2i} & \cdot & \cdot & \cdot & \sum_{i=1}^n x_{ki} \\ \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{1i}^2 & \sum_{i=1}^n x_{1i}x_{2i} & \cdot & \cdot & \cdot & \sum_{i=1}^n x_{1i}x_{ki} \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ \sum_{i=1}^n x_{ki} & \sum_{i=1}^n x_{ki}x_{1i} & \sum_{i=1}^n x_{ki}x_{2i} & \cdot & \cdot & \cdot & \sum_{i=1}^n x_{ki}^2 \end{bmatrix} \quad (3.27)$$

and

$$B = \begin{bmatrix} b_0 \\ b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_k \end{bmatrix} \quad C = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_{1i} y_i \\ \cdot \\ \cdot \\ \cdot \\ \sum_{i=1}^n x_{ki} y_i \end{bmatrix} \quad (3.28)$$

3.3 Parametric Modeling

Parametric modeling techniques find the parameters for a mathematical model describing a signal, system, or process [14]. Instead of analyzing observation pairs as the linear regressive models do, parametric modeling analyzes the frequency content which makes up the signals. This is the reason the collection of parametric models are referred to

as spectral estimators.

The power spectral density (PSD), denoted by $P_{xx}(f)$, of a random process $x[n]$ is written as:

$$P_{xx}(f) = \sum_{k=-\infty}^{\infty} r_{xx}[k] e^{-j2\pi f k} \quad -\frac{1}{2} \leq f \leq \frac{1}{2} \quad (3.29)$$

where $r_{xx}[k]$ is the autocorrelation function (ACF) of $x[n]$ defined as:

$$r_{xx}[k] = \mathcal{E}(x^*[n]x[n+k]) \quad (3.30)$$

and $\mathcal{E}$ is the expectation operator[10]. The PSD describes the distribution of power with frequency of the random process. The power level associated with a frequency could be found by filtering a signal of all frequencies with a bandpass filter but the frequency, f_0 , which we are interested in measuring. Once filtered, the power of f_0 is measured. This could continue for the central frequencies of the signal within the range of $-\frac{1}{2} \leq f_0 \leq \frac{1}{2}$.

The problem with PSD prediction is the nature of the ACF values. Since $P_{xx}(f)$ depends on an infinite number of ACF values, true PSD prediction cannot be accomplished. Instead researchers work towards creating estimators of good quality by using a finite collection of data from a signal. Variations in different parametric models lies in the different techniques used to estimate the ACF values.

The autoregressive moving average model is one of a group of rational transfer function models that attempt to predict an output $y[n]$ of a system based on the previous outputs ($y[n-1]$, $y[n-2]$... $y[0]$) and inputs ($x[n-1]$, $x[n-2]$, ..., $x[0]$). Deriving the model

involves determining the coefficients in the equation:

$$y(n) = - \sum_{k=1}^p a(k)y(n-k) + \sum_{k=0}^q b(k)x(n-k) \quad (3.31)$$

The ARMA model is actually composed of two parametric models, an autoregressive (AR) model and a moving average (MA) model.

[19] *Finite autoregression* (AR) is can be written as:

$$y_t = - \sum_{k=1}^p a_k y_{t-k} + u_t \quad (3.32)$$

where u_t is an uncorrelated process with zero mean and variance σ_u^2 and a_k for $k = 0, 1, \dots, p$ is a finite sequence with $a_0 = 1$ and $a_p \neq 0$. As p defines the length of the autoregression, it is called the *order* of the process which is referred to as AR(p).

The AR model depends only on the previous outputs of the system. It is often described as being a good approximator of the ‘peaks’ of a process. The coefficients of the model, a_k , are found by solving a set of linear equations.

The finite *moving average* model (MA) is written as

$$y_t = \sum_{k=0}^q b_k w_{t-k} \quad (3.33)$$

where w_t is an uncorrelated process with zero mean and variance σ_w^2 , and b_k for $k = 0, 1, \dots, q$ is a finite impulse response with $b_0 = 1$ and $b_q \neq 0$. Since q defines the size of the moving

average it is called the order of the process which is denoted by MA(q).

The moving average is a good approximator of a series of data with ‘deep valleys’ since it can locate zeroes near the unit circle. The moving average model depends only on the inputs to the system.

Since both the AR and MA models can approximate different parts of a signal, they are often combined into the autoregressive moving average model (ARMA) which can attempt to model both peaks and valleys of a series of data.

When working in the z-domain, the PSD estimation of the ARMA process becomes:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}} = \frac{\sum_{k=0}^M b_k z^{-k}}{\sum_{k=1}^N a_k z^{-k}} \quad (3.34)$$

and the model is sometimes referred to as a pole-zero model and is denoted as ARMA(p,q) [11]. A visual representation is seen in Figure 3.2.

Figure 3.2 ARMA Transfer Function

The determination of the coefficients of the ARMA process can be done through a non-linear relationship between the parameters and the ACF. The Yule-Walker method is a common technique which reduces the non-linear relationship to a set of linear equations which can be solved to obtain the AR parameters. Durbin's method of Maximum Likelihood Estimation is often used to determine the MA parameters. Some techniques like the Steiglitz-McBride technique find both sets of coefficients of the ARMA model together [14]. Tools also exist for the parameter estimation of the ARMA models within the time-domain. These include the Batch Least Square (BLS) and Recursive Least Square (RLS) methods.

An extension of the ARMA model is the auto-regressive moving average with extra/external process often shortened to ARMAX. The ARMAX model is very similar to

the regular ARMA model, but it includes an extra input signal which could be a correlated noise signal. The ARMAX model is written as

$$y(n) = - \sum_{k=1}^p a(k)y(n-k) + \sum_{m=0}^q b(m)x(n-m) + \sum_{l=0}^s c(l)z(n-l) \quad (3.35)$$

where s denotes the order of the eXtra/eXternal process.

3.4 Non-linear Regression

Sometimes a trajectory in use by a system will be of a non-linear nature [2]. In these cases, linear regression will fail to achieve close fitting models to describe the trajectories.

However, linear regression techniques can be extended into non-linear areas.

An example of a non-linear model is given by

$$y = a_0 + a_1 x_1^2 + a_2 \log x_2 + \dots + a_n e^{x_n} \quad (3.36)$$

where the output variable y is the trajectory under regression and the independent variables are represented by x_i for $i = 1, \dots, n$. [2] describes a single-point generic non-linear regression as:

$$y_n = f(x_n, \theta) + Z_n \quad (3.37)$$

where f is the expectation function and x_n is the independent variable for the n^{th} case. The use of θ represents the nonlinear parameters and emphasizes the difference between linear

and a nonlinear models. Z_n is assumed to have a spherical normal distribution of zero mean similar to ϵ from the linear model.

An example of paired observations of an independent variable x and a dependent variable y along with a non-linear model can be seen in Figure 3.3.

Figure 3.3 Non-linear Regression

If a non-linear model represents a trajectory, the expectation function $f(x_n, \theta)$ must generally be known for the description of the system. Assuming that the inputs into the system can be multidimensional vectors of data, then for each data point in $X \in (x_1, \dots, x_N)$, the coefficients are written as:

$$\eta_n(\theta) = f(x_n, \theta) \quad n = 1, \dots, N \quad (3.38)$$

where $\eta(\theta)$ represents the set of coefficients for the model for each independent variable.

The non-linear regression model is :

$$Y = \boldsymbol{\eta}(\boldsymbol{\theta}) + Z \quad (3.39)$$

As with linear regression analysis, the coefficients of a non-linear model are determined by minimizing the least square estimates. A common minimization technique is the Gauss-Newton method explained in more detail in [2].

Chapter 4

Trajectory Generation

To aid in reliability assessment of real-time systems requiring sequences of input, we propose the trajectory generation algorithm. This algorithm works by expanding an existing set of test trajectories into a larger set. It makes use of regressive models to create new trajectories which are statistically similar to the original test trajectories but which can then be used as additional test cases.

Existing trajectories are clustered into ‘coarse grain’ regions of the operational profile. The regressive models are developed with independent variables which are not the trajectories themselves, but variables which can be collected along with the trajectories or variables which have some other correlation to the trajectories. Each model in turn acts as a predictor for its particular region of the operational profile.

Note that in our use of regression to predict new trajectories, we make an assumption the trajectories undergoing generation can be considered random variables which are describable by some function. A second assumption is that within the clustered sections of the operational profile, the mathematical independent variables can be fitted via regression to the random dependent variables which make up the corresponding test trajectories of the cluster implying a need for relatively strong correlation.

The algorithm can be thought of as a function that transforms one set of inputs which

can be controlled into a set of trajectories that define the operational profile for the system which cannot be controlled. A mapping of the independent variables into regions of the system input space will occur. Once created, the models will be able to generate several new trajectories in the input space which are then used for testing. This is represented in Figure 4.1.

Figure 4.1 Mapping of Controllable Variables to Input Domain Trajectories

An explanation about the use of the term trajectory is needed here. Since a trajectory is defined as a sequence of data, both the dependent and the independent variables qualify as trajectories. Sometimes the term is used interchangeably to describe either set, however the goal of the algorithm is to generate new dependent variable trajectories.

4.1 Algorithm Overview

The program structure of the automated trajectory generator can be seen in Figure 4.2.

Figure 4.2 Generation Algorithm Layout

This model represents the general outline of the algorithm that creates multiple test trajectories from a smaller existing set. The algorithm is separated into two sections: the model generation and the trajectory generation.

The model generation section consists of collecting a set of test trajectories, processing the data for use by later modules, clustering the test trajectories into a group and developing a regressive model which can best fit the clustered group. Several different regressive models can be used in the developing module, including simple linear, multiple

linear, autoregressive, and non-linear regressive models.

The trajectory generation section first perturbs the original trajectory test set to generate new data for the independent variables upon which the regressive models are then applied. The output from the regressive models are checked by a set of acceptability rules which decide if the newly created trajectories are valid for the input domain of the system undergoing test. Based upon the desired number of tests, the trajectory generation section repeats the process until an adequate number of trajectories is received.

Each section is composed of several individual modules. The modules are a strength of the algorithm as they have been designed to work independently of each other, allowing for a 'refine and replace' method which can change as the system undergoing trajectory generation changes. As knowledge is gained from the approach, especially within the acceptability rules and the model development, the approach can be refined to aid for better regressive model fit or for better rejection/accepting of the model output. When the system changes, the specialized modules can be replaced by other modules refined for the new system.

The algorithm was designed to allow for automation within the trajectory generation section making for fast and easy generation. However, both the perturbation of new data and the acceptability rules must be stated in a manner to facilitate automation. The automation ends after either a sufficient number of new trajectories has been developed, or if the acceptability rate of the trajectories falls below a minimum threshold.

4.2 Collect Data

Collection of the data can be done from various sources such as data collected from actual system usage or from data retrieved via a system simulator. In some situations, the data needed for the algorithm has already been collected and no further collection is required.

Because the regressive models require independent-dependent variable relationships, the collected data should consist of the intended test trajectories to be expanded, along with additional variables which either help to create the trajectories or help to define them. The sample trajectories satisfy the dependent variable need and in some cases may contain several parameters, with each parameter consisting of a series of data. The additional variables will fulfill the role of the independent variables and must contain the same size of data as the trajectories.

Two requirements for proper operation of the regressive models is that the additional variables have some correlation to the test trajectories and that they be mathematical variables. For example, if the trajectories for a system define an airplane's flight path, pilot inputs would qualify as additional variables. If the trajectories described a chemical reaction, additional variables could be the amount of reactants used and temperature variation during the process. Since these variables will later be perturbed in the algorithm, it should make sense that they can be altered in a controllable manner. If the independent variables themselves depend upon the trajectories, no prediction of new trajectories will be possible.

4.3 Processing the Data

Depending upon the data collected, some processing of the data may be required before it can be used by the model generation routines. For the clustering algorithm to work correctly, all data sets should contain the same length of data points. This can be accomplished by truncating data sets to the size of the smallest data set. If truncation would lose too much data, other possibilities include eliminating shorter data sets or interpolation of the data to increase the size of the shorter sequences.

Conversion of the data may also be required if test trajectories have been collected from sources using different units of measurement. This does not mean normalizing the different components of a trajectory, but normalizing any differences in measurement units between trajectories. This might occur if trajectories are recorded on different systems or perhaps even across different days.

If noise is part of the collected data, noise removal by filtering should be done as part of the data processing. Several well known noise filtering algorithms exist from the area of signal processing. These algorithms can consist of smoothing filters (more fully covered in section 4.5.3) that will attempt to average out spikes in the data due to noise.

4.4 Clustering the Data

If testing of a system is directed by the operational profile, failures which are likely to occur during normal operation are detected based upon how often they are used. [13]

discusses two ways that the operational profile test selection can be done: *coarse grain* and *fine grain*. Coarse grain testing is the process of selecting a region or cluster of the operational profile while fine grain testing is the process of selecting tests from within the coarse cluster. While the coarse grain guides the test selection, fine grain testing requires several different elements from within the coarse grain region.

Since the goal of the entire approach is to develop test trajectories, it is useful to separate parts of the controllable variables into clusters which will allow the generation algorithm to predict new trajectories within a coarse input domain region. By applying clustering techniques to the independent variables, division of the test trajectories into the coarse grain regions of the operational profile is done simultaneously.

Once a cluster has been determined, the centroid for each group can be found which is then used by the regressive models to obtain the best fit of the model across the entire cluster. The *centroid* is defined as the median value of all data contained within the cluster. The centroid will be a sequence of data since the independent variables are sequences themselves.

Clustering can also be used to eliminate data outliers and bring more uniformity to the overall group making the regressive models have an even closer fit. By eliminating outliers from the clusters, the centroid assumes a value closer to the majority of trajectories.

An example of clustering technique, as applied to a sequence of data, is shown in Figure 4.3.

Figure 4.3 Example of Clustering with Trajectories

Modifying the basic steps for a clustering algorithm suggested by [9], a trajectory clustering algorithm would consist of the following:

1. Acquire a set of test trajectories.
2. Select parameters which correlate the test trajectories.
3. Transform these parameters, if necessary.
4. Remove parameter outliers.
5. Scale all observations.
6. Select a distance measure.
7. Perform clustering.
8. Interpret results.
9. Change parameters or the number of clusters to separate the trajectories.

Repeat steps 3 to 7 if number of clusters is not reached.

10. Select a representative component from each cluster.

4.4.1 Parameter Selection

The independent variables which correlate to a trajectory will be the parameters that guide the clustering process. If several parameters exist, some may have a greater impact on the clustering process than others. Less important parameters should be ignored as they can increase the cost of the process. Characteristics of important parameters are their significance to the trajectory and their amount of variance. Low variance will not provide a distinguishing metric between trajectories and have little impact on clustering. No limit is imposed on the number of parameters used, but it is suggested that the clustering process use less as opposed to more parameters when possible.

4.4.2 Outliers

An *outlier* is defined as a trajectory which lies on the outer bounds of the majority of test trajectories. Removal or inclusion of outliers will give different results from the clustering operation. They may have a significant effect on intragroup variances and centroid values.

Since we deal with a series of data points, it may be more difficult to identify a trajectory outlier than outliers of data with single values. One method of removing an outlier is to look at the intragroup variance after clustering has been performed. If the intragroup variance exceeds a selected threshold, then the data which is farthest from the centroid

should be removed, repeating this process until the desired intragroup variance is achieved.

4.4.3 Scaling the Data

Data scaling may be applied if the parameters are linear in nature and have widely varying relative values or are spread across different dimensional units. Such a scenario may occur if multiple parameters are used in the clustering and one parameter uses measurements in feet while a second uses measurements in seconds. These two parameters have little relation to each other and scaling them to use non-dimensional units will help the clustering process. Scaling should not be applied to data with non-linear parameters as the scaling can remove this non-linearity. Three common techniques used to scale data are normalizing to zero mean and unit variance, range normalization, and percentile normalization.

Normalizing to zero mean and unit variance forces each value of the data sequence to fall within a uniform distribution centered around zero. Given a series of data $X \in \{x_1, x_2, \dots, x_k\}$, the i^{th} value x'_i is scaled by

$$x'_i = \frac{x_i - \bar{x}}{s} \quad (4.1)$$

where $\bar{x}$ is the measured mean and s is the standard deviation.

Range normalization transforms a series of data from $[x_{\min}, x_{\max}]$ so that each data point falls in the range $[0, 1]$. The scaling formula is

$$x'_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \quad (4.2)$$

$x_{\min}$ and $x_{\max}$ are the minimum and maximum values which make up the series of data. A problem with the range normalization approach is that data outliers of extreme ranges will have a major influence on the min and max values causing poor normalization. The next scaling technique works better under those circumstances.

Percentile normalization performs very similarly to range normalization except that data is transformed so that 95% of the data points making up X fall between 0 and 1. With this technique, data outliers which will fall above 97.5% or below 2.5% of the average value will not affect the normalization process. The data is scaled by

$$x'_i = \frac{x_i - x_{2.5}}{x_{97.5} - x_{2.5}} \quad (4.3)$$

4.4.4 Distance Metrics

A standard way of expressing relationships of trajectories in a cluster is through a distance metric between those trajectories [7]. The distance metric is calculated over an n -dimensional space where n represents the number of chosen parameters used to describe a trajectory. This metric is then used to identify which group a trajectory belongs to by determining how close it is to the group centroid. The distance metric selection is just as important as the selection of which variables are to be used to perform the clustering. One distance metric may perform well at distinguishing between trajectories in the cluster while

another distance metric may include every single trajectory.

The most commonly used distance metric is *Euclidean Distance*. Euclidean distance is simply the distance, d , between two trajectories across the entire sequence of data. Assuming that a trajectory is defined as $X \in \{x_1, x_2, \dots, x_k\}$, then the distance between two trajectories x and y is given by:

$$d = \sum_{i=1}^k (x_i - y_i)^2{}^{1/2} \quad (4.4)$$

where the distance squared between each parameter of a trajectory is calculated. If the trajectories are n dimensional, the distance metric changes to:

$$d = \sum_{i=1}^k ((x_{1i} - y_{1i})^2 + (x_{2i} - y_{2i})^2 + \dots + (x_{ni} - y_{ni})^2)^{1/2} \quad (4.5)$$

A variation on Euclidean Distance is the *Weighted Euclidean Distance*. This distance metric applies a weighted value a_i to each parameter based upon that parameters significance.

$$d = \sum_{i=1}^k a_i \cdot (x_i - y_i)^2{}^{1/2} \quad (4.6)$$

where a_i , $i = 1, 2, \dots, n$ is the chosen weight for the respective parameter. The weighted distance metric can be used if the parameters have not been scaled yet or to give more control over how the clustering algorithm will perform based on a priori knowledge [7].

Another suggested distance metric is the *Chi-Square Distance*. This metric is

primarily used in distribution fitting. Before this metric is used, each individual parameter x_i must be normalized in order to prevent parameters with lower values from having higher weights.

$$d = \sum_{i=1}^k \frac{(x_i - y_i)^2}{x_i} \quad (4.7)$$

4.4.5 Clustering Techniques

Clustering techniques fall into one of two categories: hierarchical and non-hierarchical. In non-hierarchical techniques, trajectories are assigned into k arbitrary clusters until the intragroup variances of each cluster reaches a minimum. The value of k depends upon a given threshold used to decide the minimum variance allowed within a cluster. When the addition of another trajectory to a cluster increases the group variance, a new cluster is created. The threshold is chosen based upon the desired relation of the members of the clusters. Higher thresholds will certainly allow more trajectories per cluster as lower thresholds increase the number of clusters.

In hierarchical techniques, the collection of trajectories is divided into n desired groups. Hierarchical techniques may be either agglomerative or divisive. With agglomerative techniques each trajectory is separated into its own cluster. Neighboring clusters are merged together based upon distance metrics until the desired n groups is attained. Divisive techniques start with all trajectories in one cluster. The cluster is then divided until it reaches the desired number of n clusters.

A complete discussion on the different clustering techniques is beyond the scope of

this work, but an overview of clustering techniques can be found in text like [7].

4.4.6 Interpreting the Results

The number of trajectories in each cluster will serve as a measure of the coarseness of the regions of the operational profile. A typical cluster should probably have around 4 or 5 trajectories within it. If no trajectory was omitted from the cluster, perhaps the threshold for intragroup variance should be reduced. While it can happen that each cluster contains only one trajectory, having more than one per cluster will allow for better model fitting for that particular cluster. Figure 4.4 demonstrates one possible problem.

The trajectories in each cluster should form a uniform distribution. As in Figure 4.4,

Figure 4.4 Example of a Trajectory Outlier

the intragroup variance was chosen such that several trajectories obviously fit in a cluster and another trajectory, which appears as an outlier, is also included. If there are obvious trajectories existing as outliers of the cluster, the clustering technique should probably be

refined. Any outliers will skew the model development results.

4.4.7 Selecting a Representative Trajectory

After the clustering has been performed, a set of independent variables should be selected that serves as the average of the cluster. Remembering that clustering was performed upon independent variables, the component selected should be the sequence of data which lies closest to the centroid of the cluster based upon the chosen distance measure.

After the set of independent variables have been selected, the trajectory associated with this set becomes the representative trajectory. The purpose of the representative trajectory will be to give the regressive models the best selection which they can use to develop new trajectories within the cluster. The representative trajectory serves to define the cluster by labeling the coarse region of the input domain.

4.5 Developing a Model

Ideally, whatever model is chosen should be of much less complexity than the software component being tested. Use of various regressive models is suggested, including simple linear, multiple linear, autoregressive, and non-linear models. The linear models have shown to be a simplistic approach achieving short computational time. Different regressive models are discussed in detail in Chapter 3. The important point to remember is that the success of the models will be determined by the acceptability rate determined by the

acceptance testing of the data being generated.

In order to choose the best model, the algorithm should develop several different models and try different combinations of input data. For example, for each representative trajectory, a simple linear model could be developed for one of many different independent variables. The same can be done for combinations of independent variables for the multiple-linear regressive model. By exhaustively trying combinations of independent-dependent variables, the algorithm can vote to choose which model works the best for that particular cluster of trajectories. This also prevents the algorithm from being ‘locked’ into any one type of regressive models across the entire domain. As regions of the operational profile change, the models are allowed to adapt to that part of the domain.

4.5.1 Discriminant Analysis

When attempting to determine which variables are the most significant for use in the regressive models, discriminant analysis may need to be considered. Many techniques for discriminant analysis exist such as principle component analysis and multivariate analysis.

Principle component analysis looks at the correlations between the independent and dependent data. By using a correlation matrix, PCA assigns weights to variables which have a higher correlation to the dependent data. The higher the weight, the more significant the variable will be.

The variables chosen for the models should represent the most significant parameters. If multiple linear regression is chosen, it is suggested that only models using two, three, or four variables be used. Additional variables will not increase prediction significantly while

decreasing time efficiency and adding unnecessary complexity.

4.5.2 Smoothing

Application of a smoothing function to the final results of the model may be done here to remove sharp edges in the model output. The easiest form of smoothing is *averaging* the trajectory. Under averaging, a point of data is added to neighbors on both sides of the data point and divided by n , the number of data points added together.

$$X_i = \frac{x_{i-2} + x_{i-1} + x_i + x_{i+1} + x_{i+2}}{5} \quad (4.8)$$

Figure 4.5 Application of an Averaging Filter

Figure 4.5 demonstrates an example of a predicted output, and the data after being smoothed by an averaging filter of size $n = 5$.

Another easily implemented smoothing algorithm is *triangular smoothing*. Triangular smoothing is very similar to averaging except that it uses a weighted smoothing method.

$$X_i = \frac{x_{i-2} + 2x_{i-1} + 3x_i + 2x_{i+1} + x_{i+2}}{9} \quad (4.9)$$

Figure 4.6 Application of a Triangular Filter

There are only subtle differences between the average smooth and the triangular average smoothing, but the differences may be enough to give your model generation a better fit to the trajectories under regression. Many smoothing functions exist such as further variations on the average smoothing filters, gaussian smoothing, butterworth smoothing, etc. Any smoothing filter may be chosen depending upon the desired effect.

4.5.3 Cross-Correlation Analysis

The advantage of using the trajectory closest to the centroid to represent the cluster becomes apparent when a decision has to be made about choosing the best regressive model. After a regressive model is constructed, the model can be applied to the remaining trajectories in the cluster. The regressive model predicted output to these unused trajectories can then be directly compared to the actual trajectories themselves through the use of correlation analysis to determine the accuracy of the models.

Cross-correlation analysis looks at the relationship between two sequences of data through a correlation coefficient, r . The stronger the relationship, the higher the value of the coefficient. The equation for correlation is given by

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{[\sum X^2 - \frac{(\sum X)^2}{n}][\sum Y^2 - \frac{(\sum Y)^2}{n}]}} \quad (4.10)$$

where $\sum X$ is the summation of all data points in X, $\sum XY$ is the summation of the product of all data points in X and Y, and n is the total number of data points.

The values of r can range from +1.00 to -1.00 with a perfect relationship occurring at 1.00 and a perfect inverse relationship occurring at -1.00. Values for r between 0.00 and ± 1.00 indicate that some relationship exists but that it is not perfect. As r approaches ± 1.00 the relationship between the two trajectories becomes stronger.

After a model has been developed, it is then applied to each of the remaining trajectories in the cluster. The model predicted output is then compared against the actual

recorded rates through cross-correlation. This will result in a table of correlation values for each model against each trajectory in the cluster.

4.5.4 Choosing the Best Model

The choice of the best model is dependent upon the results of the correlation analysis from each model and the total computation time per model. An equation describing the selection can be written as

$$P = \bar{T}_{model} + e^{(1.0 - \bar{cor}_{model}) \cdot 10} \quad (4.11)$$

where T_{model} represents the average time to build a particular regressive model, and cor_{model} represents the average correlation of that model. P is then a quantifiable measurement of the value of the particular regressive model approach towards the prediction within the cluster. This equation has smaller values for regressive models which can be built faster and for regressive models which have higher correlations between the predicted and recorded trajectories. Model selection would then be made by choosing the regressive model with the smallest value of P .

4.6 Perturbing the Original Data

By perturbing the original parameters used as independent variables in the regressive models, the system is able to generate predictions from the new input. The amount of newly generated trajectories then is related to the amount of new independent variables which can

be perturbed. Any trajectories within a cluster are available to undergo perturbation, even the representative component.

How the original data is perturbed is very important. It cannot be modified too much lest the data becomes irrelevant to the models just developed. Provided that the time series data resembles multiple step functions, there are various ways to perturb the data.

One method is to multiply the original data by some value. This keeps a consistent rate of change but modifies the amplitudes of the time series data.

Figure 4.7 Demonstration of Multiplying an Input by a Constant

The value by which the original data is multiplied should not be so large that the perturbed data would no longer correspond to trajectories inside the cluster. Perturbation via multiplication should probably keep the new data within $\pm 5\%$ of the original data.

A second method is to modify the duration of some value in the data sequence while keeping consistent values for the amplitudes.

Figure 4.8 Demonstration of Increasing the Duration of an Amplitude

4.7 Application of the Model

After a new set of data has been generated, this data is then given to the model for prediction of new test data. While this is the simplest module, care must be taken to ensure that all model descriptions, including coefficients have been stored somewhere after creation for later use. As was done with the model generation, a smoothing function can be applied here to the regressive model output.

4.8 Acceptability Rules

One of the most important parts to the entire algorithm is to determine if the newly created test data actually qualifies as acceptable data. A set of rules describing acceptable trajectories can be used against the predicted trajectories to determine valid tests. After every iteration of the algorithm, the rules are applied to both the output trajectories predicted by the models and the perturbed independent variables used for those models.

The acceptability rate of the generated trajectories can guide the generation process as well as give an indication to how successful the approach is. If a certain perturbation process produces higher acceptability rates, that process could be further used with minor modifications as needed during each iteration. Perturbations producing bad results would of course be avoided. This implies a relationship between the perturbation and acceptability modules.

One guide to developing acceptability rules upon the perturbed input can be the distance metric used for the clustering process. Since the clustering was performed upon the independent variables, the distance between the perturbed data and the cluster centroid can decide if the perturbed data falls within the cluster. Perturbations which produce new values falling outside the cluster are discarded.

Acceptability rules defined to analyze the regressive model outputs can be based upon the correlation of the output trajectories to the trajectory defined as the representative component for the cluster undergoing regression. Since this trajectory acts as the classification of the coarse grain input, predictive trajectories that fall outside of a 70%

correlation could be rejected as falling outside of the coarse grain region.

Another possible area of regressive output rules are those which are system specific. These rules would identify any trajectories or perturbed independent variables which violate some definition of the input domain, perhaps by exceeding minimum or maximum values. These rules can look at anything from slope analysis of the trajectories to comparisons of global maximums and minimums.

Chapter 5

Case Study

The test case chosen for this work is the Sensor Failure Detection, Identification, and Accommodation (SFDIA) flight control scheme developed by Dr. Marcello Napolitano with the Department of Mechanical and Aerospace Engineering at West Virginia University [17, 18]. The SFDIA scheme is part of an advanced form of flight control system that uses a neural network based approach instead of a physical redundancy approach to achieve fault-tolerance. Traditional flight control systems use redundant physical components by either

Figure 5.1 Analytical Approach to Fault Tolerant Flight Control

duplicating, triplicating, or in some instances, quadruplicating sensor packages. Problems with these approaches include extra hardware costs, power consumption for the redundant components, and use of costly mission space which could be used for other purposes. The

SFDIA scheme replaces the need for redundant hardware by implementing a software analytical approach as fault tolerance.

Figure 5.2 SFDIA Layout

Figure 5.2 shows a schematic overview of the SFDIA system. The SFDIA consists of one main neural network (MNN) and 3 decentralized neural networks (DNN) [17]. The role of each DNN is to replicate either a roll, pitch, or yaw rate sensor by learning how these rates change in relation to each other. The MNN compares the outputs of each of the DNNs against the actual sensor readings. A failure in a sensor is detected and identified when the output difference between a DNN and its respective sensor increases beyond a threshold. At this point accommodation is achieved when the MNN removes the DNN from learning and uses it to replace the failed sensor.

The analytical approach proves more difficult than that of the physical redundancy approach. With physical redundancy, the reliability of individual components is fairly well known and determining the overall system reliability is merely a combination of the components. But how can one determine the reliability of the NN approach? It only uses a microprocessor to operate on and a complex non-linear network which, until removed from the learning mode, can constantly change its internal behavior.

One approach towards assessment of the system's reliability is massive testing according to the operational profile using test flight trajectories. Operations which normally occur during the SFDIA's use would be heavily tested. By using statistical methods and randomly chosen trajectories across the input domain, it would be possible to assess reliability. The set of test cases consists of aircraft flight trajectories.

Generation of this large set of test trajectories can be accomplished by several means. One solution might be the use of a flight simulator with a human pilot generating random inputs thousands of times to create a thousand relatively unique flight paths. However, this

is impossible as the pilot would be faced with a task of flying many hours on the aircraft prior to its deployment. The time required for such a scenario would also make for an inefficient solution.

Another more practical solution might be to reprogram or design a flight simulator to randomly generate a set of similar pilot inputs and then record the corresponding flight paths. An advantage to this approach is that there is a direct correlation between pilot deflection input and corresponding flight path output. The simulator is operating under well understood aerodynamic equations and the output can generally be trusted. While this solution is more practical, it does involve the development of a complicated computer routine to interact with an existing flight simulator (or the rather difficult task of creating a brand new flight simulator with routines to perform such computer input). It would be very unlikely that a researcher would have the time to develop such a flight simulator and a testing package should never be on a level of complexity as the system undergoing assessment.

A third solution would be the application of our trajectory generation algorithm. This approach offers a simplistic solution to developing a sufficient number of test trajectories. An advantage is that pre-existing flight path data can come from a flight simulation program or from recorded data acquired during operation of actual aircraft or aircraft models. This subset of data can then be used in the regression modeling to create new, unique, flight paths, which are statistically similar to the original flight path data.

A disadvantage with statistical generation is that the flight path output will be indirectly related to pilot input and the set of statistical equations which will be used to generate the flight paths. In reality, the systems which describe aircraft dynamics are of a

non-linear nature. For testing purposes, only short flight path segments which last around 20 to 30 seconds will be of interest. It is hoped that during these smaller segments of the total flight path, the non-linearity will not become a problem. Aiding the algorithm will be a set of application dependent acceptability tests which will determine if the generated flight trajectories are valid.

5.1 Tools

The completion of the work was conducted across several different computer software packages. These included the Aviator Visual Design Simulator and Matlab 5.0.

5.1.1 Aviator Visual Design Simulator (AVDS)

The Aviator Visual Design Simulator (AVDS) is a simulator which was specifically designed for engineering research. Unlike commercially available flight simulators, the AVDS system allows for recording and playback of flight test maneuvers and variables such as aerodynamic and euler angles[1].

AVDS allows the user to choose which variables are to be recorded during a simulation. Default variables include dt , the length of the simulation time steps, x , y , z positions of the aircraft in feet, x , y , z rotations in degrees, the aircraft's engine power state, α and β in degrees, the acceleration of the aircraft as seen from the pilot station, v , the

total forward velocity of the aircraft and the mach number.

Whenever a flight maneuver was conducted, the aircraft was initialized to a pre-maneuver condition. This consisted of assigning the aircraft a position, rotation, altitude, directional velocities u , v , w , and initial throttle input. Pilot input was acquired through the use of a Microsoft 3D Sidewinder joystick, computer keyboard and mouse. The maneuvers began by using the mouse to click on the start button followed immediately by using the keyboard to start recording. The intended maneuver would then be flown via use of the joystick.

Figure 5.3 Recorded Pilot Inputs and Respective Angular Rates

All maneuvers were generally concluded after 25 seconds of flight time. For each series of maneuvers, around 12 to 15 similar flights would be recorded and assigned into a test data set. Some of the collected data from a flight maneuver can be seen in Figure 5.3. In this figure, the top plots show the pilot inputs of roll, pitch, and yaw while the bottom plots show the trajectory data which can be used for testing of roll rate, pitch rate, and yaw rate. Upon visual inspection there appears to be a close correlation between the set of pilot inputs and rate outputs, so pilot inputs were chosen as the independent variables to the regressive models.

5.1.2 Matlab 5.0

Matlab 5.0 was chosen as the development environment for the generation algorithm because of the built-in functionality. The Signal Processing Toolbox included functions which would compute correlations, standard deviations, etc., while basic Matlab functions include utilities to handle matrix algebra.

5.2 Processing the Data

To ease the clustering process each simulation run was manually collected into a separate data file. Aircraft data from the same flight maneuver were named with specific names to indicate which series they belonged to. This allowed for easier coarse grain selection because each flight maneuver was considered to belong to a different region of the operational profile.

Processing of the collected data was performed next by first ensuring that all flight trajectories corresponding to each run type were of the same data size. Since most trajectories were recorded for periods around 25 seconds, this usually meant one data set around this cut-off time defined the duration for all remaining sets. Data in each flight which exceeded the cut-off time was discarded.

After that adjustment was made, a set of variables describing each trajectory were read into the Matlab program. These included the roll rate, P , pitch rate, Q , yaw rate, R , all pilot inputs including aileron, elevator, and rudder deflections and thrust, and the aircraft's position in x , y , and z coordinates. As can be seen in Figures 5.4 and 5.5, maneuvers

Figure 5.4 Pilot Input-Angular Rate Observations

Figure 5.5 Pilot Input-Angular Rate Observations

produced by similar inputs produced closely related rate outputs. This gives an indication that clustering upon the pilot inputs as independent variables will give similar rate trajectories within the input domain.

5.3 Clustering the Data

Data had to be clustered into fine regions within each of the coarse grain regions of the operational profile. Since we had already defined a coarse grain segment as each particular flight maneuver that an aircraft might experience within a short period of time, the fine grain segments would correspond to variations of the flight maneuver that produce

closely corresponding angular rates for the aircraft.

In order to achieve this clustering of the rate trajectories, parameters were selected as the pilot inputs of roll (aileron deflection), pitch (elevator deflection) and yaw (rudder deflection). These deflections were measured in degrees/sec and an arbitrary intragroup variance of 12.0 dimensionless units was used. This produced results often reducing the trajectories within the cluster to 8 or fewer.

Pictured in Figures 5.6 and 5.7 are the trajectories before and after the clustering process. In order to display the data in an easy to understand manner, the trajectories are plotted as aircraft positional data rather than pilot input even though clustering was performed on the pilot inputs.

Figure 5.6 Trajectories Before Clustering

A non-hierarchical clustering technique was used which attempted to minimize the intragroup variance for each group. A simple euclidean distance was used as the distance

metric for the clustering process. The euclidean distance was computed across three dimensions, one dimension for roll input, one dimension for pitch input, and one dimension for yaw input.

Figure 5.7 Trajectories After Clustering

5.4 Building the Regressive Models

Only three regressive model types were considered thus far for the SFDIA. These include the simple linear, multiple linear, and ARMA models. For the first iteration of model construction, a total of 15 different models were developed for each rate trajectory. Table 5.1 shows how they were constructed.

Table 5.1 Linear Model Construction

	Description (independent variable(s))
Model 1	Simple Linear (roll)
Model 2	Simple Linear (pitch)
Model 3	Simple Linear (yaw)
Model 4	Simple Linear (mach)
Model 5	2-variable Multiple Linear (roll, pitch)
Model 6	2-variable Multiple Linear (roll, yaw)
Model 7	2-variable Multiple Linear (roll, mach)
Model 8	2-variable Multiple Linear (pitch, yaw)
Model 9	2-variable Multiple Linear (pitch, mach)
Model 10	2-variable Multiple Linear (yaw, mach)
Model 11	3-variable Multiple Linear (roll, pitch, yaw)
Model 12	3-variable Multiple Linear (roll, pitch, mach)
Model 13	3-variable Multiple Linear (roll, yaw, mach)
Model 14	3-variable Multiple Linear (pitch, yaw, mach)
Model 15	4-variable Multiple Linear (roll, pitch, yaw, mach)

The independent variables chosen for the models included aileron deflections (roll input), elevator deflections (pitch input), rudder deflections (yaw input), and mach (thrust input). Thrust was never changed during any of the flight maneuvers, but the aircraft's forward velocity was highly correlated to its pitching moments. Because thrust never changed, it was not used in the clustering techniques. However, perturbations can still be introduced so the changes in the trajectory generation can be studied.

Table 5.2 ARMA Model Construction

	Driving Input	Model Orders
Model 16	roll	$p = 5, q = 5$
Model 17	pitch	$p = 5, q = 5$
Model 18	yaw	$p = 5, q = 5$
Model 19	mach	$p = 5, q = 5$
Model 20	roll	$p = 10, q = 5$
Model 21	pitch	$p = 10, q = 5$
Model 22	yaw	$p = 10, q = 5$
Model 23	mach	$p = 10, q = 5$
Model 24	roll	$p = 5, q = 10$
Model 25	pitch	$p = 5, q = 10$
Model 26	yaw	$p = 5, q = 10$
Model 27	mach	$p = 5, q = 10$
Model 28	roll	$p = 10, q = 10$
Model 29	pitch	$p = 10, q = 10$
Model 30	yaw	$p = 10, q = 10$
Model 31	mach	$p = 10, q = 10$

For the ARMA models, variations in the model parameters were done through changing of the driving variable and changing of the model orders. Table 5.2 shows how the different ARMA models were constructed.

Regressive models were built during two separate executions of the generation algorithm. When linear models were being tested, only simple linear and multiple linear models would be developed. When ARMA models were being tested, only they were

developed.

5.4.1 Correlation Results

A total of 17 different flight trajectories were created for testing. 31 different regressive models were built to determine which models provided the best correlations. Correlation results for each run across all of the different linear models are presented in Appendix A. Figures 5.8 through 5.12 show examples of how the correlations for the linear regressive model changed as the different model types are used.

Figure 5.8 Roll Rate Simple Linear Models for Test Run 8

Figure 5.9 Roll Rate Multiple Linear Models for Test Run 8

Figure 5.10 Pitch Rate Simple Linear Models for Test Run 8

Figure 5.11 Pitch Rate Multiple Linear Models for Test Run 8

Figure 5.12 Yaw Rate Multiple Linear Models for Test Run 8

In Figures 5.8 and 5.9, the roll rate model development is shown for the representative component of flight test maneuver eight. The series of data marked as roll shows a correlation of 0.9104, while the data marked as yaw has a poor correlation of 0.374. The correlations show the relationship between the actual rolling moment (plotted in red) to the predicted rolling moment from models 1 (roll input) and 3 (yaw input). Figure 5.9 shows correlations from a 2-variable multiple linear regressive model.

Likewise, Figures 5.10 and 5.11 show the pitch rate model development for the representative component in maneuver eight. The simple linear model is in Figure 5.10 with only the pitch and yaw inputs plotted. Multiple linear regression can be seen in Figure 5.11. Figure 5.12 shows a 2-variable multiple linear model development of the yaw rate predictive model from the same test run.

Appendix A contains the cross-correlation results for the developed models. Since each cluster contained around five additional trajectories that were not used to develop the model, the remaining trajectories were predicted by the developed model to determine how well that model fit the cluster. The values in the tables of Appendix A show the median cross-correlation between the model's prediction and actual values of the remaining trajectories in the cluster.

The other guide to model selection is the time required to calculate each linear model. Table 5.3 shows the average time required for models 1 through 15.

Table 5.3 Average Time Required for Linear Regressive Model Development

Model	Average Time (seconds)
1	0.967
2	0.911
3	0.969
4	0.969
5	1.147
6	1.160
7	1.160
8	1.159
9	1.165
10	1.158
11	1.251
12	1.251
13	1.251
14	1.252
15	1.416

Since the linear models were developed separately from the ARMA models, Table 5.4 shows which linear models were selected as being the best fit predictor for each of the test runs. Each run has selected the best rolling moment predictor, best pitching moment predictor, and best yaw moment predictor. The linear model selected on average for each test run is shown at the end of the table.

Table 5.4 Selected Linear Models for each Test Run

Test Run	Roll Model Selected	Pitch Model Selected	Yaw Model Selected
1	12	4	1
2	1	4	10
3	1	4	4
4	1	4	1
5	1	2	13
6	1	2	13
7	10	2	14
8	1	2	3
9	1	2	4
10	1	2	5
11	1	2	10
12	1	2	4
13	1	1	3
14	1	4	3
15	1	9	11
16	1	9	14
17	1	2	3
Avg.	1	2	3

Correlations for the ARMA model results are found in Appendix B. Figure 5.13 shows the correlation of model 16 for the roll rate prediction. The correlation for model 19 with pitch rate prediction and the actual pitching moment is seen in Figure 5.14. Yaw rate prediction from model 18 is shown in Figure 5.15.

Figure 5.13 Roll Rate ARMA Model for Test Run 8

Figure 5.14 Pitch Rate ARMA Model for Test Run 8

Figure 5.15 Yaw Rate ARMA Model for Test Run 8

Roll rate prediction is plotted in Figure 5.13 against the representative rolling moment for test eight. Correlation between these two via model 16 is 0.99307. Figure 5.14 shows the correlation from model 19 between the representative pitching moment versus the predicted pitching moment is 0.9623. Yaw prediction with model 18 can produce a correlation of 0.9992 upon the representative yaw moment.

The average time required to compute each of the ARMA models is shown in Table 5.5.

Table 5.5 Average Time Required for ARMA Model Development

Model	Average Time (seconds)
16	0.4554
17	0.4565
18	0.4597
19	0.4602
20	0.5019
21	0.5066
22	0.5099
23	0.5088
24	0.4889
25	0.4916
26	0.4976
27	0.5006
28	0.5392
29	0.5401
30	0.5482
31	0.5472

A total of 16 different ARMA models were tested upon the 17 different flight maneuvers. For each flight maneuver, the best ARMA predictor for roll rate, pitch rate, and yaw rate was selected. These results can be seen in Table 5.6. Also included is the ARMA model selected for each particular angular rate on average.

Table 5.6 Selected ARMA Models for each Test Run

Test Run	ARMA Roll Model Selected	ARMA Pitch Model Selected	ARMA Yaw Model Selected
1	23	19	26
2	19	19	18
3	16	19	16
4	29	19	16
5	16	19	16
6	24	27	24
7	18	27	15
8	20	23	24
9	20	19	19
10	16	27	16
11	16	27	16
12	27	27	27
13	24	19	19
14	18	19	30
15	16	19	16
16	16	19	16
17	16	19	24
Avg.	16	19	16

5.5 Generating New Trajectories

The module designed to generate new pilot input used two different methods. The first multiplies a constant value between the ranges of 0.6 and 1.4 in increments of 0.02 to each pilot input. This generates 40 different variations per pilot input and, when combined

together, produces over 64,000 different inputs for a total of 3,000,000 variations on the collected fine grain flight maneuvers. By only modifying the amplitudes of the pilot deflections by small amounts and keeping the timing of the deflections the same, it is hoped that the corresponding generated trajectories will be statistically related to the original trajectories.

The second variation method would alter the durations of the pilot inputs by increasing or decreasing the step inputs within the range of 0 to 1 second in 0.1 second increments. This method alone could produce on the order of several million variations. Again, if the durations are small perturbations, the generated trajectories should be statistically related.

After new perturbed independent variables were obtained, the selected regressive model was applied to generate new trajectories. Figure 5.16 shows an original roll input and original rolling rate for test run eight along with a perturbed input and generated trajectory as created by the linear regressive models.

Figure 5.16 Original Roll Rate Trajectory Versus Linear Generated Trajectory

In Figure 5.16, the original input has been perturbed by multiplying the entire signal by 1.2 of its original value. Figure 5.17 shows the original pitch input and pitch rate from test maneuver eight along with the results from perturbing the input in a similar manner and using the linear regressive model selected for that test run.

Figure 5.17 Original Pitch Rate Trajectory Versus Linear Generated Trajectory

An example of the same perturbation and generation process for yaw rate prediction is shown in Figure 5.18.

Figure 5.18 Original Yaw Rate Trajectory Versus Linear Generated Trajectory

In Figures 5.19 through 5.21, the perturbation and generation process is shown when ARMA models are selected and used to generate new trajectories. Note that in Figure 5.20, mach is used as an input into the pitch rate ARMA model. As with the other independent

variables in this example, it is multiplied by a constant of 1.2 as well.

Figure 5.19 Original Roll Rate Trajectory Versus ARMA Generated Trajectory

Figure 5.20 Original Pitch Rate Trajectory Versus ARMA Generated Trajectory

Figure 5.21 Original Yaw Rate Trajectory Versus ARMA Generated Trajectory

5.6 Acceptability Tests

Two sets of acceptability rules were developed for the SFDIA trajectory generation. The first set tested the acceptability of the pilot inputs. The second set was developed to test the acceptability of the generated roll, pitch, and yaw rates.

5.6.1 Pilot Input Acceptability

One of the acceptability tests applied to the perturbed pilot inputs checked the inputs to determine if they exceed the acceptable range which an aircraft might experience during normal flight. The test was defined by an upper and lower limit bound.

$$-0.053 \text{ rad} \leq \textit{pilot deflection} \leq 0.053 \text{ rad}$$

If a perturbed input would exceed the threshold, it would be ignored as a test case. The choice to use 0.053 radians is based upon typical deflections of surface areas in commercial jetliners.

The second set of acceptability tests determined if the perturbed pilot inputs were members of the input domain cluster. By applying the same euclidean distance metric, the distance of the perturbed input from the cluster centroid was calculated. If this distance exceeded the intragroup variance threshold, then the perturbed inputs would be ignored.

5.6.2 Angular Rate Acceptability

Only one set of acceptability rules were created for application to the generated trajectories of the test case. A limit to the degree in which a generated trajectory differed from the actual angular rate from its corresponding original independent variables made up the third and most significant rule. Essentially, if a generated trajectory fell below an 85% correlation with a trajectory from the input domain coarse region, that trajectory would be rejected. The value of 85% was chosen arbitrarily.

5.6.3 Acceptability Results

For this sampling of the acceptability rates, a simple perturbation of applying a constant multiplier to the entire set of pilot inputs was used. Each test maneuver underwent 40 different perturbations with the constant multiplier ranging from 0.9 to 1.10. Tables 5.7 and 5.8 show the percentage of acceptable trajectories for roll, pitch, and yaw rate prediction via linear regression.

Table 5.7 Percentage of Acceptable Trajectories for the Linear Models, Tests 1-9

	1	2	3	4	5	6	7	8	9
Roll %	70.0	10.0	91.11	93.75	93.75	80.0	0	93.75	90.0
Pitch %	90.0	94.0	91.11	93.75	93.75	92.50	85.0	93.75	90.0
Yaw %	60.71	66.0	11.11	0	62.50	0	2.5	93.75	84.0

Table 5.8 Percentage of Acceptable Trajectories for the Linear Models, Tests 10-17

	10	11	12	13	14	15	16	17
Roll %	92.50	86.25	61.25	71.25	0	78.75	68.75	31.25
Pitch %	92.50	86.25	86.25	93.75	91.25	78.75	75.0	93.75
Yaw %	0	17.5	1.25	93.75	60.0	72.50	80.0	12.5

The same perturbation process was applied again with the ARMA models used to generate new trajectories. Their acceptability rates are presented in Tables 5.9 and 5.10.

Table 5.9 Percentage of Acceptable Trajectories for ARMA Models, Tests 1-9

	1	2	3	4	5	6	7	8	9
Roll %	39.29	53.33	91.11	81.25	93.75	12.5	12.5	93.75	90.0
Pitch %	90.0	96.67	91.11	93.75	93.75	0	85.0	93.75	90.0
Yaw %	67.86	96.67	91.11	25.00	93.75	0	23.75	93.75	90.0

Table 5.10 Percentage of Acceptable Trajectories for ARMA Models, Tests 10-17

	10	11	12	13	14	15	16	17
Roll %	92.5	86.25	0	93.75	18.75	62.50	81.25	93.75
Pitch %	80.0	25.0	12.50	93.75	91.25	78.75	75.0	75.0
Yaw %	92.50	86.25	12.50	93.75	60.0	78.75	87.50	25.0

The lowest acceptability percent indicates the maximum number of new trajectories which can be used for testing. This is due to the fact that a flight maneuver will consist of all three rates together. If yaw has the lowest acceptability of 12.5% for example, then even if the roll and pitch rate generation achieve 100% acceptability, only 12.5% of them may be used as testing trajectories. Looking at the ARMA model acceptability rates, approximately 50% of new flight trajectories consisting of roll, pitch, and yaw angular rates are acceptable across all 17 test maneuvers.

5.6.4 Trajectory Generation Time

The goal of the test data generation is to produce a large number of new trajectories for testing within a short period of time. The amount of time required to collect each trajectory for a test maneuver is the duration of that trajectory. This was usually around 25 seconds. For ARMA model generation, a new rate trajectory could be generated every 7.5×10^{-5} s.

Since the SFDIA scheme uses as inputs the three trajectories of roll, pitch, and yaw rate together as part of the definition of a flight maneuver, a single trajectory by itself is not enough for a test of the system. A test must be composed of the three rate trajectories which together can be generated every 2.25×10^{-3} s. This means approximately 10,000 new flight maneuvers can be created in 25 seconds. Assuming an average 50% acceptance rate, five thousand new maneuvers can be created in the time it takes to produce one simulated maneuver.

Linear regression is slower than ARMA modeling. It requires 0.6 s for each

generated angular rate. This implies around 2 seconds for a new generated testing trajectory.

In the time to simulate one flight maneuver, linear regression can generate around 13 different testing trajectories.

Chapter 6

Analysis of the Testing Approach

Overall, the generation algorithm was able to produce statistically related trajectories for the SFDIA scheme. ARMA models performed better than the linear regressive models. While the autoregressive models could predict all three angular rates well, the linear models could only predict roll and pitch well.

6.1 Model Development

ARMA models performed very well at predicting each representative component achieving higher than 99.9% for many of the models developed. For other trajectories which are within the same clustered region as the component, the prediction does decrease slightly.

For example, consider the eleventh flight maneuver found in Table A.1 in Appendix A. The average model correlation for roll rate prediction via model 16 was 0.9880. This value is arrived at from applying model 16 as developed upon the representative trajectory to a total of eight trajectories (including the representative component). Table 6.1 shows the correlations.

Table 6.1 Individual Trajectory Correlations for Test Run 11

	Correlation between Actual and Predicted
Trajectory 1	0.9936
Trajectory 2	0.9828
Trajectory 3	0.9914
Trajectory 4	0.9878
Trajectory 5	0.9922
Trajectory 6	0.9862
Trajectory 7	0.9902
Trajectory 8	0.9799

As can be seen in the table, correlation fluctuations for trajectories within the cluster range from as high as 99.3% to as low as 98%. Still, the lowest achieved correlation is typically higher than what the linear models can achieve.

The ARMA models proved suitable to model aircraft maneuvers which could be approximated as linear single-input-single-output (SISO) systems such as rolling maneuvers where the aircraft's performance was only affected by a single input. With a pure, high frequency rolling maneuver, the angular roll rate was only affected by the rolling input. Another scenario where this occurred were situations where there were only pure diving and climbing maneuvers which affected the pitch rate. Under these circumstances, the ARMA models performed well because the systems could be approximated as SISO systems with one degree of freedom.

Linear regression models generated predictions around 90% for representative

components. They performed better for roll rate and pitch rate prediction than they did for yaw rate prediction. Correlations from the built linear regressive models to actual values averaged around 95% for roll rate prediction and 90% for pitch rate prediction while yaw averaged below 90%. This could be due to the inherently non-linear nature of yaw rate. When using the ARMA models, past values of yaw rate helped its prediction. This was not available to the linear regressive models.

Many of the higher correlation values gave false impressions of close fitting by the linear models. For instance, during test maneuver eight, pitch rate prediction via simple linear regression using yaw as input gave an 81% correlation however upon visual inspection, the prediction looked very poor. Only correlations which achieve higher than 90% should be considered satisfactory while correlations above 95% should be considered good.

The high correlations are caused because of the use of the normalized xcor function in Matlab. This function works by calculating the correlation between two separate signals for a range of time lags, then adjusting all the values so that they fall between 1.0 and -1.0.. If the length of the signals is N , then the cross-correlation is computed for time lags of $-N$ to N . The case with all of the cross-correlations between the predicted and actual values, maximum cross-correlations occur at zero lag and progressively got worse for lags in either direction. When the cross-correlations were normalized, the smaller cross-correlations at the extreme ranges of the lag range would shift the correlations near zero to higher percentages.

An unexpected advantage to the ARMA models was the computational time required to develop the autoregressive models as compared to the linear models. On average, each

ARMA model only required around 0.5 seconds while the linear models each required over one second.

Model selection for the linear regressive models favored the simpler solution, often choosing an angular rate model its corresponding pilot input. Roll rate models selected simple linear regression with aileron deflections as input 15 out of the 17 test maneuvers.

The choice of model with autoregression varied with each test maneuver. Roll rate prediction often used aileron deflection as the driving input. Mach was used as the driving input the majority of times for pitch rate prediction. The ARMA models developed for yaw rate prediction switched between aileron deflections and rudder deflections indicating a strong relationship of yaw with roll and yaw input.

6.2 Handling Zero Inputs

A main problem with ARMA models is that they do not handle situations where a driving input may have zero values for all time. Such a case occurred if for example, the pilot gave no pitch input. Since the driving input corresponds to the ‘zeroes’ of the random transfer function, an input which is zero for all time will generate zeroes for the filter coefficients. Matlab would handle these cases by returning NaN (not a number) which is Matlab’s method for handling undefined values.

The problem for zero inputs also affected the multiple linear regressive models. When the coefficients for the regressive model were computed and solved via matrices, the zero input would result in several matrices containing rows and columns full of zeroes. This

creates a problem with inverting a matrix. The problem was resolved by eliminating those rows and columns corresponding to zero inputs and continuing the matrix mathematics. This allowed the determination of the coefficients of the inputs which were not zeros. All other coefficients not able to be solved were then assigned to zero.

However, there is a disadvantage to this solution as we shall soon see. The effect of assigning the coefficient to zero is the same as saying that the corresponding independent variable has no effect in the model, which might be like reducing the order of the model by one order. But this model is based upon the centroid for the cluster undergoing regression. When the selection process begins to choose which regressive model behaves the best, there is a possibility that the other trajectories within the same cluster set will not have the same independent variable of zero value. However, since the coefficient operating on that independent variable is zero, it can never have an effect on the model. The assumption of assigning the coefficient to zero should be a choice of preference.

6.3 Acceptability Rates

The first acceptability rule to eliminate pilot deflections which exceeded a threshold boundary did not eliminate any of the generated trajectories. The second rule did eliminate some, rejecting approximately 2% of the generated trajectories. The third acceptability rule had the highest rejection numbers, eliminating a total of 8% of roll generations, 4% of pitch generations, and 7% of yaw generations.

Overall, ARMA models outperformed the linear regression models in generation.

Areas of increased acceptability can be seen especially in the yaw rate generation. This coincides with the correlations from the model developments. ARMA was able to better predict the angular rates.

Looking at Table 5.9 on page 96, it can be seen that test maneuver six performed very poorly for the ARMA models. The answer for this poor performance can be seen from the ARMA model correlation Table B.1 in Appendix B. Maneuver 6 had no ARMA model developed which produced a correlation between predicted and actual trajectories above 62%. This probably was due to a poor fitting of the ARMA model to the pilot inputs. When the original inputs were perturbed, the models could not generate many predictions which were within the 85% correlation tolerance. In fact, only the roll model could produce any acceptable results at all. The linear models performed slightly better than the ARMA models did, but even they were unable to predict any acceptable trajectories for yaw. Maneuvers such as test six might require non linear regression in order to generate tests.

Frequently there were common acceptability rates among the roll, pitch, and yaw rate predictions. An example is the ARMA model predictions for test maneuver three where all rate predictions had acceptability rates of 91.1%. This indicates situations where the predictions all fall above the correlation threshold, but the new perturbed pilot inputs fail to stay within the clustered region. If the pilot inputs are not inside the cluster, it eliminates all trajectories created from the inputs. The acceptability numbers become similar then because all rates reject the same generated trajectories.

Chapter 7

Conclusions

As a beginning step towards trajectory generation, the algorithm demonstrated it could expand existing test trajectories into a larger set which could be used for reliability assessment. One of the strengths of the approach is its ability to allow refinement for each case it is applied to, adapting to that system as necessary.

The bulk of the generation time will be the development of the acceptability rules as they apply to the system to undergo assessment and the collection of the initial subset of test trajectories. Some time may be spent refining the algorithm, but the overall application of the algorithm is sufficiently short enough to qualify as a valid approach to developing tests for a system. Additionally, the generation is not subjective which can eliminate any testers bias towards developing new test trajectories.

7.1 Case Study

ARMA models performed exceptionally well towards generating angular rate trajectories for the SFDIA case study. Some of the models were able to achieve upwards of 99% prediction for the angular rates. When used in generation, the models performed fairly well. While not all test maneuvers experienced favorable generation, ARMA models were able to successfully generate new test trajectories 50% of the time.

Considering that it would take 25 seconds to simulate three angular rate trajectories of roll, pitch, and yaw, the ARMA models can produce three trajectories 5000 times faster. The linear models were only three times faster, but this is probably due to the way the matrix mathematics were implemented in Matlab m-files and might be improvable.

7.2 Further Work

One area which requires further study is the development of the acceptability rules as explained in section 6.3. By analyzing if the generated trajectories meet the flying requirements of civilian aircraft, a better value on the acceptability of the approach towards the SFDIA scheme can be ascertained.

A problem with applying acceptability tests towards the generated flight trajectories is that many of the criteria describing aerodynamically acceptable rates are to check the non-linearity of the signals. If a signal produces too non-linear of a response, it usually fails to meet specifications of the aircraft. None of the acceptability tests look for non-linearities in the generated trajectories nor use the flying requirements criteria for civilian aircraft.

All aircraft must conform to a set of flying quality requirements imposed in the United States by the Federal Aviation Association (FAA). These requirements can be split into longitudinal requirements and lateral-directional requirements.

Roskam [25] explains that an airplane's response to cockpit control inputs must be such that the pilot can achieve the mission objectives with reasonable physical and mental efforts [25]. As this pertains to a Boeing 777 commercial jetliner, a pilot should be able to

give inputs to the aircraft to provide a safe, stable, and comfortable flight within reasonable efforts. Having generated the angular rates for the aircraft, a set of guides to determine the acceptability of these rates is required. The rates must be such that they fit within the definition of a reasonable flight path.

Knowing what the flying requirements criteria are, better acceptability rules could be developed to ascertain the validity of the generated trajectories. Rules developed to look at the flying requirements would include longitudinal and lateral-directional requirements. For longitudinal requirements, the generated trajectories would need to pass phugoid and short-period damping criteria. Phugoid oscillations are the long-period air-speed oscillations which occur when an airplane seeks a stabilized air-speed following a disturbance. Phugoid analysis may be possible provided the generated trajectories cover a longer time period than 25 seconds. Short-period oscillations have higher frequencies than phugoid oscillations and can be investigated with the trajectories as they already are.

For lateral requirements, rules that could be developed would look at dutch roll damping and roll mode time constants. The dutch roll is primarily concerned with a motion of rolling, sideslipping, and yawing. The roll mode time constant is a criteria which measures the rapidity of a roll response.

The approach has only investigated simple linear, multiple linear, and autoregressive moving-average models. While non-linear models were suggested, no study of non-linear model development was made. It would be interesting to determine if non-linear models would aid the generation algorithm with the SFDIA case study.

Bibliography

- [1] *AVDS User Manual*. Artificial Horizons, Inc. November 1997.
- [2] Bates, D. M., Watts, D. G. *Nonlinear Regression Analysis and its Applications*. John Wiley & Sons. New York. 1988.
- [3] Cukic, B. Transformational Approach to Software Reliability Assessment. Doctoral Dissertation. Department of Computer Science, University of Houston, Houston, TX, August 1997.
- [4] Demillo, R. A. W. Michael McCracken, R. J. Martin, John F. Passafiume. *Software Testing and Evaluation*. The Benjamin/Cummings Publishing Company. Melo Park, Ca. 1987.
- [5] Dukelow, J. “Verification and Validation of Neural Networks,” *Proceedings of the Neural Network Workshop for the Hanford Community*, pp 76-80, Pacific Northwest National Laboratory, Richland, WA, USA, 1994.
- [6] Duran, J. W. and Ntafos, S. “A Report on Random Testing,” *Proceedings of the 5th International Conference on Software Engineering*, March 9-12, 1981. San Diego, CA, pp. 179-83.
- [7] Hartigan, J. A.. *Clustering Algorithms*. Yale University. Wiley & Sons. 1975.
- [8] Ince, D. C.. “The Automatic Generation of Test Data,” *Computer*, pp. 63-69, Vol. 30, No. 1, 1987.

- [9] Jain, R. *The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc. New York, 1991.
- [10] Kay, S. M. *Modern Spectral Estimation*. PTR Prentice Hall, Englewood Cliffs, NJ. 1988.
- [11] Lightspeed Semiconductor. IIR Filters. www.lightspeed.com/products/ipslides/cast/dsp/iir.htm. 1999.
- [12] Littlewood, B., Strigini, L. The Risks of Software. *Scientific American*, November 1992. pp 62-75
- [13] Lyu, M. R. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press. McGraw-Hill. New York, 1996.
- [14] *Matlab Signal Processing Toolbox User's Guide*, The MathWorks, Inc. 1996.
- [15] Mattson, D. E. *Statistics, Difficult Concepts, Understandable Explanations*. The C. V. Mosby Company. St. Louis, Missouri, 1981.
- [16] Maurer, P. M. Generating Test Data with Enhanced Context-Free Grammars. *IEEE Software*. July 1990. pp 50-55.
- [17] Napolitano, M.R., Molinaro, G., Innocenti, M., Martinelli, D. A Complete Hardware Package for a Fault Tolerant Flight Control System Using On-Line Learning Neural Networks. *IEEE Control Systems Technology*, Jan 1998.

- [18] Napolitano, M.R., Neppach, C.D., Casdorph, V., Naylor, S., Innocenti, M., and Silvestri, G. "A Neural Network-Based Scheme for Sensor Failure Detection, Identification and Accomodation", *AIAA Journal of Control and Dynamics*, vol 18, no 6, pp. 1280-1286, 1995.
- [19] Oppenheim, A., Schafer, R. *Discrete-Time Signal Processing*, Prentice Hall, New Jersey, 1989.
- [20] Ostrand, T.J., Balcer, M. J. The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of the ACM*. vol. 31, no. 6, June 1988, pp. 676-686.
- [21] Parnas, D., Schouwen, A. J., and Kwan, S. P. Evaluation of Safety-Critical Software. *Communications of the ACM*, vol. 33, no. 6, June 1990, pp.636-648.
- [22] Paradkar, A., Tai, K. C., Vouk, M. A. Automatic Test Generation for Predicates. 7th International Symposium on Software Reliability Engineering (ISSRE'96).
- [23] Rapps, S., and Weyuker, E.J. "Selecting Software Test Data Using Data Flow Information", *IEEE Transactions on Software Engineering*, vol. SE-13, no 7, July 1987, pp. 761-765.
- [24] Roper, M.. *Software Testing*. International Software Quality Assurance Series. McGraw-Hill. Berkshire, England. 1994.
- [25] Roskam, J. *Airplane Flight Dynamics and Automatic Flight Controls Part 1*. Roskam Aviation and Engineering Corporation. Ottawa, Kansas. 1994.
- [26] Stearns, D. "Test Data Creation", www.csc.calpoly.edu/~dstearns/570B/testDataCreation.html, California Polytechnic State University, Cailifornia, 1996.

- [27] Walpole, R., Myers, R.. *Probability and Statistics for Engineers and Scientists*.
Prentice Hall, New Jersey, 1993.
- [28] Yin, H., Lebne-Dengel, Z., Malaiya, Y. K. Automatic Test Generation using
Checkpoint Encoding and Antirandom Testing. Proceedings of the Eighth
International Symposium on Software Reliability Engineering (ISSRE'97), pp. 84-93.

Appendix A

Linear Regressive Correlation Results

Tables A.1 to A.6 contain the cross-correlation results between the predicted angular rate and the actual rate for each model for each maneuver. Since each maneuver can be composed of anywhere from five to twelve sets of flight data, the values in the table are the mean cross-correlations. Values range from -1.0000 to 1.0000 with a -1.0000 indicating a strong inverse correlation and 1.0000 indicating a strong correlation.

Tables A.1 and A.2 show the cross-correlations for the roll rate prediction. Tests 1 through 17 represent the seventeen different flight maneuvers while the left hand side indicates the 15 different linear models used upon the SFDIA scheme. Tables A.3 and A.4 show the results for the pitch rate model development and Tables A.5 and A.6 show the yaw rate model development.

Table A.1 Average Roll Rate Linear Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
1	0.8368	0.6205	0.8749	0.7909	0.9073	0.7867	0.1002	0.8924	0.8362
2	0.2794	0.1332	0.0035	0.0129	0.1926	0.0963	0.0761	0.2188	-0.1394
3	0.2194	0.3111	0.0035	0.0129	0.4762	0.1095	0.0460	0.1414	-0.0071
4	0.4234	0.0821	0.0759	0.0679	0.4762	0.0515	0.1764	0.2995	0.1612
5	0.8286	0.6104	0.8695	0.7862	0.8997	0.7776	0.0757	0.8779	0.8279
6	0.8257	0.4042	0.8695	0.7862	0.9055	0.7779	0.0458	0.8862	0.8178
7	0.8396	0.5081	0.8719	0.7857	0.9055	0.7792	0.1764	0.8877	0.8216
8	0.2719	0.3121	0.0035	0.0129	0.5351	0.1072	0.1019	0.1476	-0.1957
9	0.3389	0.0821	0.0759	0.0679	0.5656	0.1610	0.1775	0.3020	0.0280
10	0.4205	0.3265	0.0759	0.0679	0.5001	0.1439	0.2071	0.2395	0.0456
11	0.8340	0.4044	0.8749	0.7909	0.8402	0.7873	0.1023	0.8763	0.8233
12	0.8462	0.5174	0.8772	0.7904	0.8963	0.7887	0.1775	0.8810	0.7878
13	0.8379	0.4174	0.8772	0.7904	0.9068	0.7895	0.2070	0.8630	0.8337
14	0.3398	0.3258	0.0759	0.0679	0.6031	0.1487	0.1907	0.2398	0.0524
15	0.8382	0.4174	0.8772	0.7904	0.7690	0.7888	0.1907	0.8137	0.7964

Table A.2 Average Roll Rate Linear Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
1	0.9080	0.8177	0.7642	0.8076	0.1473	0.8875	0.8660	0.7210
2	0.1589	0.0594	0.1430	0.0875	0.1473	0.6103	0.1438	0.1061
3	0.1681	-0.0355	0.0961	0.0189	0.1119	0.3288	0.3412	0.0874
4	0.4473	0.2077	0.0241	0.1009	0.0400	0.5362	0.5054	0.0127
5	0.9029	0.8058	0.6796	0.7937	0.1473	0.8866	0.8614	0.7065
6	0.9038	0.8084	0.5824	0.7825	0.1115	0.8847	0.8621	0.6786
7	0.9027	0.8093	0.5862	0.7938	0.0399	0.8862	0.8621	0.6854
8	0.1894	0.0472	0.0674	0.0173	0.1115	0.6248	0.3403	0.0834
9	0.3858	0.2113	0.0300	0.1009	0.0399	0.6601	0.4777	0.1792
10	0.4379	0.2049	0.0321	0.0318	0.0295	0.5770	0.5024	0.0985
11	0.9078	0.8127	0.5810	0.7953	0.1119	0.8902	0.8640	0.6855
12	0.9029	0.8158	0.5472	0.8076	0.0400	0.8869	0.8642	0.6072
13	0.9063	0.8082	0.5631	0.7963	0.0288	0.8897	0.8662	0.6721
14	0.4041	0.2077	0.0328	0.0335	0.0288	0.6647	0.4980	0.1823
15	0.9029	0.8055	0.5227	0.7963	0.0288	0.8875	0.8631	0.6173

Table A.3 Average Pitch Rate Linear Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
1	0.9659	0.9718	0.9631	0.8882	0.8493	0.1783	0.2674	0.9539	0.9768
2	0.9580	0.9706	0.9729	0.8952	0.9645	0.9822	0.9835	0.9785	0.9815
3	0.9680	0.9590	0.9729	0.8952	0.8370	0.7505	0.5478	0.8141	0.9794
4	0.9805	0.9930	0.9946	0.9040	0.8516	0.4716	0.3934	0.9313	0.9718
5	0.9564	0.9717	0.9625	0.8880	0.9642	0.9788	0.9833	0.9800	0.9818
6	0.9671	0.9576	0.9625	0.8880	0.8584	0.7432	0.5446	0.8125	0.9763
7	0.9786	0.9930	0.9941	0.8965	0.8783	0.4894	0.3934	0.9272	0.9715
8	0.9686	0.9587	0.9729	0.8952	0.9655	0.9787	0.9826	0.8658	0.9443
9	0.9871	0.9930	0.9946	0.9040	0.9637	0.9821	0.9619	0.7789	0.9614
10	0.9809	0.9914	0.9946	0.9040	0.8493	0.7473	0.5422	0.7490	0.9655
11	0.9664	0.9580	0.9631	0.8882	0.9677	0.9759	0.9827	0.8734	0.9448
12	0.9805	0.9930	0.9942	0.8967	0.9619	0.9801	0.9620	0.7854	0.9618
13	0.9794	0.9911	0.9942	0.8967	0.8794	0.7459	0.5453	0.7447	0.9644
14	0.9877	0.9914	0.9946	0.9040	0.9670	0.9806	0.9603	0.9452	0.9367
15	0.9836	0.9911	0.9942	0.8967	0.9680	0.9774	0.9603	0.9399	0.9397

Table A.4 Average Pitch Rate Linear Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
1	0.7641	0.3834	0.3696	0.9824	0.9684	0.7126	0.6379	0.2095
2	0.9775	0.9553	0.9694	0.9809	0.9684	0.8483	0.8722	0.9834
3	0.7707	-0.0184	0.1780	0.9824	0.9689	0.6780	0.6985	0.5838
4	0.7924	0.6118	0.4685	0.9814	0.9853	0.7752	0.7186	0.7964
5	0.9789	0.9604	0.9692	0.9823	0.9684	0.8440	0.8735	0.9830
6	0.7693	0.0365	0.1797	0.9834	0.9688	0.7167	0.6969	0.5830
7	0.8005	0.6142	0.4716	0.9827	0.9853	0.7813	0.6938	0.7967
8	0.9773	0.9498	0.9696	0.9824	0.9688	0.8374	0.8782	0.9830
9	0.9352	0.9614	0.8903	0.9814	0.9853	0.8698	0.9045	0.9862
10	0.7875	0.5993	0.4689	0.9836	0.9840	0.7865	0.7437	0.8170
11	0.9797	0.9583	0.9707	0.9834	0.9689	0.8389	0.8815	0.9833
12	0.9546	0.9653	0.8849	0.9827	0.9853	0.8639	0.9039	0.9870
13	0.8004	0.5928	0.4722	0.9844	0.9840	0.7878	0.7466	0.8172
14	0.9357	0.9567	0.8766	0.9836	0.9840	0.8577	0.9077	0.9867
15	0.9544	0.9624	0.8722	0.9844	0.9840	0.8575	0.9062	0.9868

Table A.5 Average Yaw Rate Linear Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
1	0.7393	0.8392	0.1694	0.2432	0.4857	-0.0328	-0.1174	0.8779	0.8862
2	0.7258	0.8376	0.1719	0.2390	0.5634	0.0505	-0.2153	0.7773	0.8906
3	0.7145	0.8557	0.1719	0.2390	0.7391	0.2464	0.0122	0.8828	0.5465
4	0.7163	0.8554	0.1804	0.2342	0.6082	-0.0431	-0.1239	0.8763	0.9079
5	0.7209	0.8390	0.1693	0.2432	0.5984	0.0807	-0.2153	0.7539	0.8707
6	0.7172	0.8542	0.1693	0.2432	0.7776	0.2638	0.0119	0.8810	0.5173
7	0.7160	0.8595	0.1739	0.2377	0.6136	-0.0148	-0.1239	0.8762	0.9046
8	0.7156	0.8555	0.1719	0.2390	0.7556	0.2131	0.1916	0.5011	0.5753
9	0.7161	0.8554	0.1804	0.2342	0.6393	0.0564	0.0640	0.8626	0.9053
10	0.7106	0.8693	0.1804	0.2342	0.7399	0.3129	0.2610	0.7945	0.9020
11	0.7115	0.8545	0.1694	0.2432	0.7806	0.2246	0.1935	0.4854	0.5529
12	0.7082	0.8599	0.1744	0.2378	0.5716	0.0841	0.0641	0.8519	0.9059
13	0.7099	0.8656	0.1744	0.2378	0.7836	0.3263	0.2622	0.2957	0.8913
14	0.7111	0.8702	0.1804	0.2342	0.7544	0.2871	0.2724	0.8787	0.8994
15	0.7049	0.8656	0.1744	0.2378	0.7831	0.2929	0.2724	0.8434	0.8972

Table A.6 Average Yaw Rate Linear Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
1	0.5672	0.7000	0.1395	0.9074	0.8322	0.7958	0.8397	0.4526
2	0.5952	0.6857	0.1077	0.9071	0.8322	0.8236	0.8733	0.1536
3	0.3282	0.6991	0.1463	0.9117	0.8571	0.8277	0.8672	0.5076
4	0.5350	0.7000	0.2974	0.9078	0.8284	0.8029	0.8466	0.1968
5	0.6271	0.6847	0.1074	0.9072	0.8322	0.8233	0.8755	0.1707
6	0.4084	0.7010	0.1449	0.9128	0.8570	0.8226	0.8649	0.4592
7	0.6150	0.7009	0.2960	0.9081	0.8284	0.8036	0.8442	0.2114
8	0.4565	0.6791	0.1033	0.9114	0.8570	0.8453	0.8815	0.2119
9	0.5528	0.6644	0.2321	0.9078	0.8284	0.8025	0.8827	0.2206
10	0.4251	0.7096	0.2622	0.9163	0.8433	0.8397	0.8669	0.3038
11	0.5292	0.6787	0.1033	0.9131	0.8571	0.8494	0.8833	0.2278
12	0.5763	0.6663	0.2396	0.9083	0.8284	0.8028	0.8850	0.2354
13	0.5539	0.7091	0.2708	0.9175	0.8437	0.8319	0.8675	0.3172
14	0.4412	0.6768	0.1042	0.9166	0.8437	0.8270	0.8860	0.3005
15	0.5072	0.6768	0.1032	0.9175	0.8437	0.8295	0.8870	0.3126

Appendix B

ARMA Regressive Correlation Results

Tables B.1 to B.6 contain the cross-correlation results between the predicted angular rate and the actual rate for the autoregressive moving-average models for each maneuver. As in the Appendix A tables, values represent the median cross-correlation results as each model is applied to each flight maneuver. Values range from -1.0000 to 1.0000 with a -1.0000 indicating a strong inverse correlation and 1.0000 indicating a strong correlation.

Tables B.1 and B.2 show the cross-correlations for the roll rate prediction, Tables B.3 and B.4 show pitch rate prediction and Tables B.5 and B.6 show yaw rate model development. Tests 1 through 17 represent the seventeen different flight maneuvers. The left hand side shows models 16 through 31, the sixteen different ARMA models used to develop trajectories for the SFDIA scheme.

Table B.1 Average Roll Rate ARMA Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
16	0.9566	0.9352	0.9950	0.8863	0.9360	0.0870	0.0000	0.9789	0.8364
17	0.3268	0.0000	0.0000	0.0000	0.0172	0.0565	0.0526	0.0117	0.0046
18	0.3110	0.0065	0.0000	0.0000	0.5756	-0.0205	0.2850	0.0318	0.3664
19	0.3498	0.5521	0.1068	0.2014	0.6156	-0.0002	0.1791	0.1829	0.3423
20	0.9566	0.9349	0.9948	0.8959	0.8945	0.1168	0.0000	0.9837	0.9877
21	0.1167	0.0000	0.0000	0.0000	0.0412	-0.0005	0.1780	0.0799	0.2127
22	0.0001	0.0012	0.0000	0.0000	0.3753	0.0236	0.2832	0.0554	0.2412
23	0.3826	0.5516	0.2550	0.1848	0.5989	-0.0125	0.1327	0.2124	0.4286
24	0.9552	0.9351	0.9951	0.9113	0.8205	0.1281	0.0000	0.9818	0.9821
25	0.3116	0.0000	0.0000	0.0000	0.0176	0.0585	0.0720	-0.0641	-0.0320
26	0.2880	0.0000	0.0000	0.0000	0.5533	-0.0761	0.2834	0.0042	0.3782
27	0.3517	0.5515	0.1223	0.2034	0.6165	-0.0090	0.1639	0.2093	0.3675
28	0.9554	0.9341	0.9954	0.9178	0.8663	0.1238	0.0000	0.9838	0.9773
29	0.2823	0.0000	0.0000	0.0000	0.0863	-0.0043	0.1308	0.1254	-0.038
30	0.3145	0.0000	0.0000	0.0000	0.5570	0.0580	0.0834	0.2389	0.3893
31	0.3676	0.5516	0.2684	0.2017	0.6200	-0.0161	0.1298	0.1450	0.4188

Table B.2 Average Roll Rate ARMA Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
16	0.9940	0.9880	0.9636	0.9239	0.0000	0.9053	0.9283	0.9679
17	0.2255	0.0716	0.0698	0.0000	0.0000	0.5878	0.0343	0.2350
18	0.2616	0.2373	0.0582	0.0065	0.4238	0.0371	0.0343	0.1478
19	0.4987	0.4758	0.2373	0.4905	0.1925	0.6752	0.6475	0.3657
20	0.9951	0.9799	0.9574	0.9814	0.0000	0.9011	0.9230	0.9645
21	0.1860	0.0277	-0.0027	0.0000	0.0000	0.5198	0.0356	0.1605
22	0.3379	-0.0249	0.2575	-0.0065	0.4207	0.0384	0.0352	0.1124
23	0.5822	0.4536	0.1757	0.3888	0.1805	0.6693	0.6693	0.3821
24	0.9946	0.9848	0.9627	0.9854	0.0000	0.9021	0.9302	0.9253
25	0.2107	0.0974	0.0777	0.0000	0.0000	0.5870	0.0342	0.1810
26	0.2702	0.2326	0.0645	0.0058	0.4228	0.0374	0.0340	0.1600
27	0.4986	0.5033	0.3382	0.4944	0.1797	0.6708	0.6514	0.3535
28	0.9950	0.9885	0.9567	0.9577	0.0000	0.9079	0.9325	0.7978
29	0.2595	0.0846	0.0208	0.0000	0.0000	0.5448	-0.0425	0.1827
30	0.3118	0.2802	0.2108	-0.0492	0.4224	0.0369	0.0362	0.1387
31	0.5674	0.5266	0.4558	0.3969	0.1709	0.6692	0.6688	0.3900

Table B.3 Average Pitch Rate ARMA Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
16	-0.1024	0.9719	-0.1856	0.1176	0.7363	0.0845	0.0000	-0.1003	-0.0006
17	0.0688	0.0000	0.0000	0.0000	-0.0164	-0.0374	0.2676	-0.0597	-0.1528
18	-0.1787	0.0358	0.0000	0.0000	0.8577	-0.0259	0.2676	0.0785	-0.0215
19	0.9881	0.9954	0.9936	0.9875	0.9095	0.1545	0.9536	0.9268	0.9836
20	-0.3457	0.9719	-0.2565	-0.2219	0.7361	-0.1145	0.0000	0.2708	-0.3512
21	-0.0682	0.0000	0.0000	0.0000	-0.0922	0.1131	0.2749	-0.0490	0.1204
22	-0.2374	-0.1935	0.0000	0.0000	0.2939	-0.0442	0.2774	0.1812	-0.0238
23	0.9881	0.9954	0.9936	0.9874	0.7012	-0.0697	0.9364	0.9551	0.9818
24	-0.2377	0.9719	-0.1328	0.1165	0.7424	0.0876	0.0000	-0.1486	-0.0271
25	-0.1569	0.0000	0.0000	0.0000	-0.0888	0.0183	0.2491	-0.0421	0.035
26	-0.1246	-0.0439	0.0000	0.0000	0.8514	0.0597	0.2673	-0.0709	-0.0214
27	0.9881	0.9954	0.9936	0.9875	0.9096	0.6259	0.9565	0.9442	0.9831
28	-0.3207	0.9719	-0.2566	0.1065	0.7157	0.1313	0.0000	-0.2128	0.1206
29	-0.0418	0.0000	0.0000	0.0000	-0.2067	0.1315	0.3169	0.0682	0.1316
30	-0.2413	-0.2039	0.0000	0.0000	-0.2543	-0.0594	0.2770	0.1402	-0.0215
31	0.9881	0.9954	0.9936	-0.8950	0.4562	0.1144	0.9539	0.7399	0.9828

Table B.4 Average Pitch Rate ARMA Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
16	0.0499	0.4331	0.2997	-0.2287	0.0000	-0.3051	-0.6374	0.2302
17	0.0329	-0.0058	-0.1688	0.0000	0.0000	-0.3760	-0.5071	0.3058
18	0.6081	0.0408	0.2561	-0.0425	-0.0541	0.1019	-0.3904	-0.0748
19	0.8668	0.7255	0.8317	0.9956	0.9952	0.9338	0.9017	0.8879
20	0.0022	0.0821	0.0052	-0.3212	0.0000	0.1569	-0.0001	0.3202
21	0.0164	0.0415	0.0611	0.0000	0.0000	-0.0120	-0.0425	0.2638
22	0.0718	-0.1282	-0.2669	-0.3006	0.1757	-0.1499	-0.2644	0.3173
23	0.0926	0.3115	0.7629	0.9956	0.9952	-0.0607	0.4628	0.2473
24	0.0106	-0.0761	0.2998	-0.3139	0.0000	-0.3063	-0.6376	0.2290
25	-0.0282	-0.0085	0.1044	0.0000	0.0000	0.2685	-0.5479	-0.0225
26	0.5702	0.0410	0.2557	0.2338	-0.0672	-0.1040	-0.3873	-0.0553
27	0.8929	0.7305	0.8390	0.9956	0.9952	0.9296	0.8999	0.8716
28	-0.0635	0.0149	0.3389	-0.2138	0.0000	-0.2624	-0.0864	-0.0472
29	-0.0168	0.0863	0.1072	0.0000	0.0000	-0.2768	-0.4416	0.3030
30	0.5763	-0.1628	-0.2771	-0.2867	-0.2969	-0.0483	-0.4984	-0.1059
31	0.8957	0.0004	0.0151	0.9956	-0.0091	0.7239	0.8947	0.8749

Table B.5 Average Yaw Rate ARMA Model Correlations for Test Maneuvers 1-9

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9
16	0.7961	0.9294	0.9629	0.2583	0.9339	0.2298	0.0000	0.9118	0.6591
17	0.6644	0.0000	0.0000	0.0000	0.4285	0.1985	-0.2702	0.0321	0.1339
18	0.7871	0.9410	0.0000	0.0000	0.7994	0.0171	0.4989	0.7792	0.4355
19	0.7682	0.9100	0.1701	0.2583	0.9084	0.1009	-0.0901	0.8549	0.9480
20	0.7777	0.9308	0.9627	0.1919	0.9313	0.0536	0.0000	0.2438	0.6774
21	0.6688	0.0000	0.0000	0.0000	0.4277	0.0030	-0.2635	0.0629	0.0310
22	0.7743	-0.0233	0.0000	0.0000	0.3136	0.0338	0.4053	-0.0214	-0.0027
23	0.7688	0.9100	0.1888	0.2573	0.9085	0.1905	-0.1524	0.8788	0.9370
24	0.7960	0.9298	0.9641	0.2300	0.9341	0.2302	0.0000	0.9227	0.1977
25	0.6647	0.0000	0.0000	0.0000	0.4017	0.1310	-0.2579	0.0690	0.1360
26	0.7886	-0.0056	0.0000	0.0000	0.8047	-0.0314	0.5090	0.7799	0.4346
27	0.7683	0.9097	0.1822	0.2574	0.9084	0.1215	-0.0895	0.8547	0.9483
28	0.7967	-0.0230	0.9586	0.0908	0.9353	-0.1833	0.0000	0.8838	0.0649
29	0.6643	0.0000	0.0000	0.0000	0.0310	-0.0222	-0.2608	0.0642	0.1504
30	0.7863	-0.0042	0.0000	0.0000	0.1608	-0.0061	0.5353	-0.0213	-0.0480
31	0.7697	0.9096	0.1830	0.2532	0.9084	0.0942	-0.0995	-0.8796	0.9410

Table B.6 Average Yaw Rate ARMA Model Correlations for Test Maneuvers 10-17

	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15	Test 16	Test 17
16	0.9801	0.9382	0.5975	0.6217	0.0000	0.9639	0.9871	0.5993
17	0.3409	0.2483	0.1157	0.0000	0.0000	0.8402	-0.0808	0.2261
18	0.6920	0.0775	0.0274	0.9262	0.7921	0.0991	0.8229	0.2032
19	0.8634	0.3225	0.1474	0.9417	0.8476	0.9505	0.9581	0.2577
20	0.9790	-0.0336	0.6047	0.3944	0.0000	0.0266	0.2400	0.0843
21	0.3374	0.2291	0.1179	0.0000	0.0000	0.8392	0.0000	0.1150
22	0.2360	0.0859	-0.0456	0.9130	0.5632	0.1267	-0.2034	0.2310
23	0.8632	0.0006	0.1450	0.9424	0.8365	0.5691	0.7505	0.2262
24	0.9812	0.0800	0.6050	0.6275	0.0000	0.9642	0.9872	0.7142
25	0.3401	0.2412	0.1110	0.0000	0.0000	0.8397	-0.0892	0.2158
26	0.6920	0.0808	0.0246	0.9247	0.7923	0.1169	-0.1479	0.1715
27	0.8634	-0.0328	0.1531	0.9408	0.8477	0.9508	0.9581	0.2588
28	-0.0009	0.9336	0.6343	0.6107	0.0000	-0.4577	0.2720	0.5040
29	0.3397	0.2350	0.1108	0.0000	0.0000	0.8385	0.0000	0.2306
30	0.6661	0.0337	-0.0224	0.8937	0.8542	0.1267	-0.1525	0.2549
31	0.8631	-0.0006	0.1581	0.9416	0.5861	-0.6931	0.7509	0.2221