International Journal of Software Engineering and Knowledge Engineering
Vol. 12, No. 2 (2002) 175–200

AUTOMATED GENERATION OF TEST TRAJECTORIES FOR EMBEDDED FLIGHT CONTROL SYSTEMS

BOJAN CUKIC

Lane Department of Computer Science and Electrical Engineering,
West Virginia University, Morgantown, WV 26506-6109, USA
cukic@csee.wvu.edu

BRIAN J. TAYLOR

Institute for Software Research,
1000 Technology Drive, Fairmont, WV 26554, USA
btaylor@softwareresearch.org

HARSHINDER SINGH

Department of Statistics, West Virginia University,
Morgantown, WV 26506, USA
hsingh@stat.wvu.edu

Automated generation of test cases is a prerequisite for fast testing. Whereas the research in automated test data generation addressed the creation of individual test points, test trajectory generation has attracted limited attention. In simple terms, a test trajectory is defined as a series of data points, with each (possibly multidimensional) point relying upon the value(s) of previous point(s). Many embedded systems use data trajectories as inputs, including closed-loop process controllers, robotic manipulators, nuclear monitoring systems, and flight control systems. For these systems, testers can either handcraft test trajectories, use input trajectories from older versions of the system or, perhaps, collect test data in a high fidelity system simulator. While these are valid approaches, they are expensive and time-consuming, especially if the assessment goals require many tests.

We developed a framework for expanding a small, conventionally developed set of test trajectories into a large set suitable, for example, for system safety assurance. Statistical regression is the core of this framework. The regression analysis builds a relationship between controllable independent variables and closely correlated dependent variables, which represent test trajectories. By perturbing the independent variables, new test trajectories are generated automatically. Our approach has been applied in the safety assessment of a fault tolerant flight control system. Linear regression, multiple linear regression, and autoregressive techniques are compared. The performance metrics include the speed of test generation and the percentage of "acceptable" trajectories, measured by the domain specific reasonableness checks.

Keywords: Test data generation; software testing; embedded systems; software safety; process control.

1. Introduction

Safety assessment requires a large number of test cases to ensure meaningful coverage of the critical sections of the system's input domain. In the aerospace engineering domain, for example, safety assessment is performed to validate system's conformance with safety related rules, called envelopes. The envelopes are usually derived from previous experience (*experience envelopes*), and known system constraints (*system envelopes*). If any of the system constraint rules is violated, the violation must be reported and investigated, potentially leading to dismissal in the flight qualification process and redesign [1, 14]. Manual development of test cases needed for this type of system assessment is tedious, time consuming and expensive. As a result, too few test cases may be available for thorough system assessment. In these situations, automated test data generators represent an attractive alternative, provided that building them is feasible.

While the test generator explained here targets embedded flight control systems, this technique is applicable to most process-control programs. The purpose of a control system is to maintain specified properties of the outputs of the process, given some reference values. The architectural solution frequently used for software implementations of these systems is the control loop paradigm. The controller executes a series of cycles or frames. At the beginning of each frame, it reads inputs from the sensors, then it performs some computations, and at the end of the frame it sends commands to the mechanisms. Computations in each frame depend on the inputs read at the beginning of the frame as well as the values of some internal variables called history or state variables.

A complication in the automated test generation stems from the fact that inputs are not independent snapshots of variable values at the beginning of a frame. Meaningful inputs consist of the *sequences* of snapshots. Values of different variables across the frame boundaries depend on the history of the computation. We call these sequences *input trajectories*. Trajectories define system inputs as a function of time and history [13]. Consequently, software faults manifest themselves as system failures as the result of the execution on a series of input readings (frame inputs). Each trajectory corresponds to the notion of a system *demand*, i.e., the demand is described by an input trajectory. Demands are selected from the input space. In the case of reliability testing, the demands are selected according to a probability distribution over the input space, called the *operational profile*. This distribution mimics the operational system usage. Reliability testing assumes that successive demands are selected independently according to the operational profile. In case of safety testing, demands are selected such that the coverage of the critical sections of the system's input domain inspires confidence that the safety constraints will not be violated. In practice, institutions such as NASA's Center for Independent Verification and Validation perform testing using test designs in which reliability and safety goals supplement each other.

Whereas research in automated test data generation has addressed the creation

of individual test points, test trajectory generation has attracted limited attention in the research community. The only exception that we are aware of is the telecommunication domain. There, short input sequences (dial-up options in a phone call) and relatively small number of system states make Markov chains a feasible methodology for describing possible trajectories [17]. Flight control systems require long input trajectories and undergo numerous state transitions during the execution. We propose a trajectory generation algorithm for systems where describing input sequences with Markov models is not feasible.

Our approach is based on the idea of expanding an existing set of test trajectories. Existing trajectories may have been handcrafted, used by an older version of the system or, perhaps, collected in a high fidelity system simulator. Without automation, generating new trajectories would be expensive and/or a time-consuming activity. Large test sets are usually needed to stress test the new system, for example, exercise borders of experience and system envelopes.

We use regressive models to create new trajectories, which are statistically similar to the trajectories in the original set, but sufficiently different to represent additional test cases. This approach violates the statistical independence requirement for the selection of demands for reliability testing [8]. However, the technique is suitable for achieving *demand coverage* within input subdomains which are interesting from the safety assessment point of view. The use of statistical regression methods does not imply that this technique is applied in regression testing (reapplying old tests to a changed system). The purpose of our framework is test data generation.

The rest of the paper is organized as follows. Section 2 provides the description of an application that motivated the development of the test trajectory generation framework. Section 3 gives a general description of our approach to trajectory generation. Section 4 briefly describes the regression models proposed for use with the test trajectory generator. Section 5 presents our results from the application of the approach in the flight control domain. Discussion of these results is presented in Sec. 6. Section 7 reviews the related work in the general field of test data generation. A conclusion and a description of future work are provided in Sec. 8.

2. Motivation: Testing of Flight Control Systems

The development of an automated technique for generation of test trajectories for flight control systems is motivated by the problems we encountered in performing independent verification and validation on the *Day of Launch I-Load Update System* (*DOLILU II*). DOLILU II has been developed for the Space Shuttle program to allow modification of the Shuttle's first stage guidance commands based on actual wind conditions measured in hours preceding the launch. This system consists of the trajectory software required to generate and verify the new I-Loads, to evaluate wind and trajectory conditions, and to recommend decisions to fly (or not to fly) with the new I-Loads.

The DOLILU II system functions as follows. When provided with the wind and atmospheric data, the executive module invokes the *Day of Launch Ascent Design System* ($DADS$). DADS module generates guidance commands for the day of launch condition. Guidance commands are passed to the *Space Vehicle Dynamic Simulation* ($SVDS$) *Processor*. It generates trajectories for reference winds and atmospheric conditions. Only successfully simulated trajectories are forwarded to the verification module.

Near real-time verification is the most critical function of the DOLILU II system. Successfully simulated trajectories and their corresponding I-Loads are verified for conformance with safety related rules, called envelopes. The envelopes have been derived from previous experience (*experience envelopes*), and known system constraints (*system envelopes*). If any of the system constraint rules is violated, the violation must be reported and the trajectory must be dismissed. *The Day of Launch I-Load Verification Table* ($DIVDT$) *Processor* performs trajectory verification, and it should detect all potentially unsafe flight conditions [1, 14].

The problem with thorough testing for safety violations is the limited number of readily available trajectories. A relatively small number of available valid atmospheric data set hampers the generation of new trajectories. The second problem is the efficiency of test data generation. It takes a long time to compute a trajectory in the described simulation environment, much longer than it takes to test whether it satisfies about forty predefined safety properties (semi-automatic test oracles are available). The software verification and validation team wanted to test DIVDT module with trajectories that challenge the limits of experience and/or safety flight envelopes. Ideally, test trajectories had to expose hazardous transient system behaviors, if any existed. The approach taken was to develop the methodology that can use existing trajectories, computed from the database of valid field conditions of an earlier version of the DOLILU system, to generate sufficiently realistic but dissimilar trajectories for safety testing.

In the next section, we present the test trajectory generation algorithm developed in response to these requirements. We are not at liberty to discuss its application to DOLILU II system. Nevertheless, the case study presented in Sec. 5 discusses the application of the trajectory generation algorithm in the testing of *Sensor Failure Detection, Identification and Accommodation* ($SFDIA$) scheme. SFDIA is a module within a piloted aircraft control system.

3. Trajectory Generation Methodology

In the approach proposed in this paper, regressive models are developed to determine relationships between independent (explanatory[a]) variables and dependent variables. The dependent variables, representing a complete description of a test

[a]In the rest of the paper, terms *independent variables* and *explanatory variables* are used interchangeably. No difference should be inferred from the use of these two terms.

trajectory, are predicted by applying regressive models upon the independent variables. The independent variables must be controllable and highly correlated to the dependent variables.

The independent, i.e., explanatory variables must be collected prior to the development of the regressive model. Independent variables must be found in the existing data set. Trajectories are clustered into regions based upon the similarities of the independent variables. Because of the correlation the independent variables have with the dependent variables, the dependent variables end up clustered into regions as well. Clusters usually represent different operational regimes. As an illustration, the case study presented later considers the flight trajectories of an aircraft. Pilot (stick) commands are used as independent variables in existing trajectories, the corresponding angular rates for the given flight are the dependent variables. Consequently, the clusters of trajectories represent different flight maneuvers. In general, system control inputs are usually good candidates for independent variables.

Regressive models are developed to describe each of these clusters. In order to generate a needed number of trajectories, the models are applied to perturbations of the independent variables, thus producing different test trajectories. A visual representation of the trajectory generation is shown in Fig. 1.

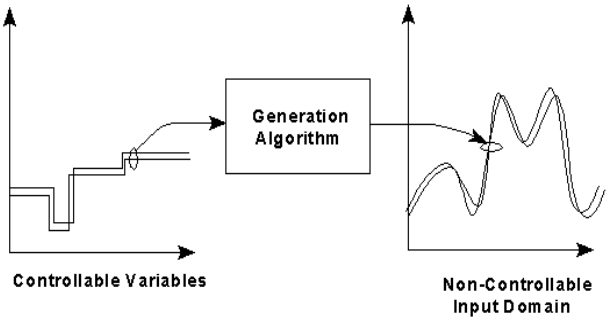

Fig. 1. Mapping of controllable variables to input domain trajectories.

The trajectory generation algorithm can be thought of as a function that transforms one set of inputs, the controlled ones, into a set of trajectories described by the corresponding dependent variables. The structure of the automated trajectory generator is shown in Fig. 2.

The algorithm consists of two modules: the model generation and the trajectory generation. The model generation consists of collecting a set of existing trajectories, preprocessing the data for use by later modules, clustering the existing trajectories and developing a regressive model which can best fit each clustered group. Different regressive models can be used in the model development component, including simple linear, multiple linear, autoregressive, and non-linear regressive models. These techniques form the core of the framework.

The trajectory generation module perturbs one or more independent variables

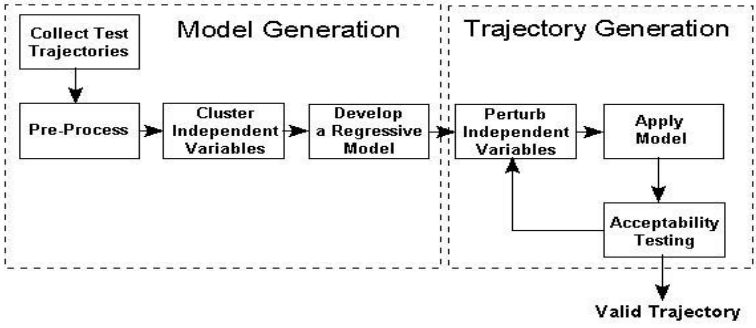

Fig. 2. Outline of the trajectory generation algorithm.

from the collected set of trajectories to generate new inputs for the regressive models. The new sets of dependent data generated by the regressive models are checked against a set of acceptability rules. These rules provide reasonableness checks, i.e., they decide if the created trajectory is a valid input for the system undergoing the test. The generation section continues to automatically generate new trajectories, as long as they are needed.

Each of the two building blocks is composed of individual modules that are designed to work independently of each other, allowing for a "refine and replace" approach. As the generation approach is improved, it will be refined to aid for better regressive model fit or for better acceptance rate of the outputs. When the application domain changes, the individual modules can be replaced by those refined to work better with the new application.

The rest of this section briefly describes individual modules.

3.1. *Collection and preprocessing of existing trajectories*

Collection of the existing trajectories can be done from various sources, such as those collected from actual system usage, retrieved via a system simulator, or from test cases applied to similar systems in the past. Because regression predicts the relationship between explanatory and dependent variables, the collected data must consist of the set of trajectories to be expanded (containing dependent variables), and some additional controllable (independent) variables that are correlated with the trajectories. The requirement that the explanatory variables be controllable allows for their perturbation in the later phases of the algorithm.

Depending upon the data collected, preprocessing of the data may be required before it can be used by the model generation routine. For the clustering algorithm to work correctly, the data sets should be of the same length in terms of the number of frames. This can be accomplished, for example, by truncating data sets to the size of the shortest trajectory. If such a truncation eliminates too much of useful information, other possibilities include the elimination of shorter data sets or interpolation of the data to increase the size of shorter sequences. Data and/or

metrics conversion may also be required if test trajectories have been collected from more than one type of source with each using different units of measurement. Noise removal may also be applied at this stage.

3.2. *Clustering*

By clustering test trajectories into regions representing different operational conditions, regressive models can be defined for each of these regions. Note that such models define relatively coarse regions of system operation. Each test trajectory generated by the model serves as a system test or, using the terminology introduced earlier, a system *demand*.

Clustering techniques fall into one of the following two categories: hierarchical and non-hierarchical [6]. In non-hierarchical techniques, trajectories are assigned into k arbitrary clusters until the intragroup variances of each cluster reach 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 set 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 are attained. Divisive techniques start with all trajectories in one cluster. The cluster is then divided until it reaches the desired number of clusters.

While it can happen that each cluster contains only one trajectory, having more than one per cluster will allow for better model fitting. Basic steps for the clustering module in Fig. 2, represented under the name *Cluster Independent Variables*, consist of the following:

1. Select independent variables to act as clustering parameters.
2. Transform these parameters, if necessary.
3. Remove parameter outliers, if necessary.
4. Select a distance measure.
5. Perform clustering.
6. Interpret results.
7. Change parameters or modify the clustering technique, if necessary.
8. Repeat steps 1 to 7 until the clustering process generates "satisfactory" classes.
9. Select a representative component for each cluster.
10. Select a representative trajectory for each cluster.

The independent variables that correlate to a trajectory will be the parameters that guide the clustering process. Clustering does not need to be performed upon

all of the independent variables, if there are several. Characteristics of important parameters are their significance to the trajectory, and their amount of variance. Low variance, for example, will not provide a distinguishing metric between trajectories and will have little impact on clustering.

Clustering allows the elimination of outliers and brings more uniformity to the trajectories within a cluster. An outlier is defined as a trajectory that lies on the outer bounds of similar trajectories. Outlier elimination improves the regressive model fit. The regressive model, then, generates dependent variables that lie closer to the centroid of the original cluster. Removal or inclusion of outliers will give different clusters and, consequently, result in different regressive models for each cluster. This choice has important consequences for the rest of the trajectory generation approach, and should be made in accordance with the assessment goals of the specific system.

A standard way of expressing the relationship between trajectories within a cluster is through a distance metric [4, 6]. The distance metric is calculated over an n-dimensional space where n represents the number of parameters used to describe a trajectory. This metric is 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 variables used to perform the clustering. One distance metric may perform well at distinguishing between trajectories in the cluster, while another distance metric may include all trajectories in a single cluster.

The most commonly used distance metric is Euclidean Distance. Assuming that a trajectory X is defined by $\{x_1, x_2, \ldots, x_i, \ldots, x_k\}$, where x_i's are the values of variable x at time i, then the distance between two trajectories, x and y, is given by:

$$d = \left\{ \sum_{i=1}^{k} (x_i - y_i)^2 \right\}^{\frac{1}{2}}.$$

If the trajectories are n-dimensional, the distance metric changes to:

$$d = \left\{ \sum_{i=1}^{k} ((x_{1i} - y_{1i})^2 + (x_{2i} - y_{2i})^2 + \cdots + (x_{ni} - y_{ni})^2) \right\}^{\frac{1}{2}}$$

Other acceptable distance metrics include the Weighted Euclidean Distance and the Chi-Square Distance [6].

Once the clusters are populated, their *centroids* must be computed. The centroid is defined as the median value of the independent variable within the cluster. If multiple parameters are used in the clustering technique, the centroid is multidimensional. Centroids are needed for selecting the *representative component* of each cluster. The representative component is an actual sequence of independent variables that lies closest to the centroid of the cluster, as determined by the distance measure. Since independent variables guide the clustering process, the representative component consists of independent variables only. The *representative*

trajectory, on the other hand, consists of the dependent variables that correspond to the representative component. The representative component and the representative trajectory become the inputs into the regressive model development. Because the regressive model will describe the cluster, any other trajectory of dependent variables chosen instead of the representative trajectory will not produce as good a fit across the entire cluster.

3.3. *Developing the regressive model*

The representative component and representative trajectory are composed of multiple variables. Based on these, different regressive models can be developed. Each model, linear or nonlinear, originates from a combination of independent-dependent variable(s) pairs. For example, let us consider a simple linear model. This model predicts a dependent variable's behavior based upon a single independent variable. For a representative trajectory, a simple linear model can be developed for *each pair* of independent-dependent variables. By exhaustively trying all the combinations of independent-dependent variables, the algorithm can select the model that works best for a given dependent variable. This prevents the algorithm from being "locked" into a specific type of regressive models across all of the dependent variables.

After a regressive model is constructed for the representative trajectory, it is applied to the remaining trajectories in the cluster and analyzed. 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 values of r can range from 1.0 down to -1.0 with a perfect match occurring at 1 and a perfect inverse relationship occurring at -1. The equation for correlation is given by

$$r = \frac{\Sigma XY - \dfrac{\Sigma X \Sigma Y}{n}}{\sqrt{\left[\Sigma X^2 - \dfrac{(\Sigma X)^2}{n}\right]\left[\Sigma Y^2 - \dfrac{(\Sigma Y)^2}{n}\right]}}$$

where ΣX is the summation of all data points in X, ΣY is the summation of all data points in Y, ΣXY is the summation of the product of all data points in X and Y, and n is the total number of data points.

Improvement of the correlation between the predicted and actual trajectories can come from applying a smoothing function to the output of the regressive models. Sometimes the output exhibits sharp peaks and transitions due to the linear process. A filter, such as local averaging, can smooth regression output and, generally, yield stronger correlation.

The selection of the best model is based on a cost function. The cost function may look, for example, at the computational time needed to generate a new trajectory using the model and the accuracy of the model. The accuracy is determined

from cross-correlations between the predicted output of the model and the dependent variables in the set of existing trajectories, as explained above. An example of a cost function is given by the following expression:

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

where T_{model} represents the average time to generate a trajectory using the particular regressive model, and cor_{model} represents the average correlation of that model. While a dimensionless quantity, P is proportional to the cost of applying the particular regressive model for prediction within the cluster. P is smaller for regressive models that can generate trajectories faster and for regressive models that have higher correlation between the predicted and recorded trajectories. The regressive model with the smallest value of P is, generally, the most suitable one. This cost function is very subjective and, for any given application, a careful calibration is needed. Calibration should provide a balance between the accuracy and efficiency, and possibly other desirable factors such as, for example, test diversity, domain/code coverage, etc.

3.4. *Generating new trajectories*

The generation of new (different) test trajectories comes from the perturbation of the independent variables within the cluster. Independent variables of any trajectory within a cluster are available to undergo perturbation. Methods of perturbation vary and should be chosen based upon the characteristics of the independent variables. Once perturbed, the new data points are used as inputs into the selected regressive model producing a new set of dependent variables. If a smoothing function was applied to the regressive model earlier during the selection module, it should be applied to the new trajectories too.

One of the most important aspects of the entire approach is to determine if the newly created trajectory actually qualifies as a valid test case. To select valid tests, all generated trajectories should be compared against the set of rules describing acceptable trajectories. These rules must check the dependent variables predicted by the model, as well as the perturbed independent variables used in the generation.

One of the guidelines in developing acceptability rules can be the distance metric used in the clustering process. Clustering is based on values of the independent variables, so the distance between the perturbed values and the centroid of the cluster can decide if the new (perturbed) independent data still resides within the cluster. Perturbations that produce independent variables falling outside the cluster may, for example, be discarded.

Acceptability rules defined to analyze the outputs of regressive model can be based on the correlation between generated trajectories and the representative trajectory for the cluster undergoing regression. Artificially generated trajectories that fall outside of an acceptable correlation threshold could be rejected.

A very important set of additional acceptability rules is system specific. These rules should identify any trajectories that violate the definition of the input domain,

perhaps by exceeding maximum stress expected to be tolerated by the system. These rules can look at the slope analysis of the trajectories, compare global maximums and minimums, etc.

4. A Brief Description of Regressive Models

Regression methods are statistical tools used to study the relationship between a dependent variable and a set of explanatory (independent) variables. The regression models are often used for prediction of the value of the dependent variables based on known values of the explanatory variables. The most commonly used regression models in the literature are linear models, generalized linear models [27] and autoregressive models. Below we describe briefly the linear regression and autoregressive models, which are used for test trajectory generation.

4.1. *Linear regression models*

The simplest and widely used regression model is the linear regression model which assumes that the mean of the dependent variable is a linear function of the explanatory variables. Suppose $(x_{1i}, x_{2i}, \ldots, x_{ki}, y_i)$, $i = 1, 2, \ldots, n$, denote the data points on the explanatory variables $x_1, x_2, \ldots, x_k$ and the dependent variable y. The statistical model for a multiple linear regression is

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_k x_{ki} + \varepsilon_i$$

$i = 1, 2, \ldots, n$. The errors ε_i are assumed to be independent Gaussian variables with mean zero and variance σ^2. The least square principle is used to find the estimates of the parameters $\beta_0, \beta_1, \ldots, \beta_k$. Let $b_0, b_1, \ldots, b_k$ denote the estimators of $\beta_0, \beta_1, \ldots, \beta_k$, respectively. These are the solutions that minimize

$$\sum_{i=1}^{n} (y_i - b_0 - b_1 x_{1i} - \ldots - b_k x_{ki})^2 .$$

The predicted response for the ith observation is given by

$$\hat{y}_i = b_0 + b_1 x_{1i} + \ldots + b_k x_{ki} .$$

The squared multiple correlation coefficient R^2 is interpreted as the proportion of the variability in the response (dependent) variable that is explained by the explanatory variables in the multiple linear regression model.

The simple linear regression model is the linear regression model when there is only one explanatory variable. In this case this is simply the best fit straight line (in the least square sense) to the bivariate data consisting of the paired values of the explanatory and the response variable. For details of the linear regression models, refer to [25].

4.2. *Autoregressive models*

These statistical models for stationary stochastic processes depend on a finite number of parameters. They include autoregressive, moving average and autoregressive moving average models.

Autoregressive models are of the type

$$y_i = a_1 y_{i-1} + a_2 y_{i-2} + \ldots + a_p y_{i-p} + u_i \,,$$

where u_i's are uncorrelated random variables with zero mean and common variance σ_u^2. p defines the length of the autoregression and it is called the order of the process which is denoted by $AR(p)$, [26]. The maximum likelihood estimates of the parameters $a_1, a_2, \ldots, a_k$ are obtained by solving the likelihood equations. This model is often described as being a good approximation of the "peaks" of a process.

The moving average (MA) model is defined by

$$y_i = b_0 w_i + b_1 w_{i-1} + \ldots + b_q w_{i-q}$$

where w_j's are uncorrelated random variables with zero means and common variance σ_w^2 and b_j's are the parameters of the model. q defines the size of the moving average and the moving average process with size q is denoted by $MA(q)$. The moving average model is considered a good approximator of a series of data with deep valleys.

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

$$y_i = a_1 y_{i-1} + a_2 y_{i-2} + \ldots + a_p y_{i-p} + b_0 w_i + b_1 w_{i-1} + \ldots + b_q w_{i-q} \,.$$

The ARMA model attempts to predict an output yi of a system based on the previous outputs $y_{i-1}, \ldots, y_{i-p}$ and the inputs $w_i, w_{i-1}, \ldots, w_{i-q}$. The fitting of the model involves finding the coefficients $a_1, a_2, \ldots, a_p, b_1, b_2, \ldots, b_q$. The determination of the coefficients of the ARMA model can be done using Yule-Walker method. It reduces the nonlinear relationships to a set of linear equations, which can be solved to obtain the coefficients. Other methods for finding the coefficients include the Batch Least Square (BLS), Recursive Least Square (RLS) and Steiglitz-McBride method [3, 12].

5. Case Study

A case study chosen for this work is the Sensor Failure Detection, Identification, and Accommodation (SFDIA) flight control scheme [10, 11]. The SFDIA scheme is part of an advanced embedded flight control system that uses analytical instead of a physical redundancy to achieve fault-tolerance. Traditional flight control systems cope with sensor failures by duplicating, triplicating or, in some instances, quadruplicating sensor packages. Problems with these approaches include extra hardware

costs, power consumption, weight and space considerations. The SFDIA scheme replaces redundant hardware by implementing a neural network based analytical scheme that learns the correlation between values provided by different sensors. In case of a sensor failure, the neural network provides the values that mimic the expected (learned) value of the failed sensor.

Partial assessment of the SFDIA scheme's safety is the goal of the case study. The assessment is conducted by applying flight trajectories as system inputs. Each of these test trajectories is composed of a sequence of aircraft's rolling, pitching, and yawing moments, where a moment is the angular rate of change that an aircraft experiences during flight. Safety assessment included fault injection tests too. Since fault injection techniques do not contribute to the topic of this paper, their description is omitted.

SFDIA test trajectories were generated by our automated system. Pre-existing flight paths were collected from an advanced flight simulation program, which allowed for recording of multiple variables during flight maneuvers. This subset of data was then used in the regression modeling. Because of the potential problem with linear regressive models applied to a typical nonlinear domain (aircraft flight trajectories), we only considered short flight path segments lasting between 20 and 30 seconds. By limiting the duration of each trajectory, we hoped to maintain as much linearity as possible. The short duration of flight segments does not seriously diminish the value of safety assessment, because many critical flight maneuvers can be represented by short trajectories. The loss of nonlinearity has been addressed by introducing autoregressive models. The performance of linear and autoregressive models has been compared and is reported below. Since these models gave satisfactory results, we did not use nonlinear regressive models in this study. However, the principles of the trajectory generation framework apply to nonlinear regressive models as well and their application is a rather straightforward extension.

5.1. *Tools, data collection, smoothing, pre-processing*

Simulated trajectories were collected using the Aviator Visual Design Simulator (AVDS), a professional level aircraft flight simulator that allows for recording of aerodynamic variables including pilot inputs, angular moments, forward velocity, and directional position.

A total of 17 different flight maneuvers were flown in the simulator for the data collection step. During the recording of a flight maneuver, the aircraft's state was initialized by assigning it a position, rotation, altitude, directional velocities, and initial throttle input. The flight maneuver would then proceed as some combination of diving, climbing, and banking. Each flight maneuver underwent 10 to 15 simulated repetitions that lasted for approximately 25 seconds. After collection of all trajectories for a particular maneuver, they underwent a preprocessing step ensuring that each trajectory had the same number of data points.

In the given problem domain, there exists a close relationship between the pilot

inputs of aileron deflection, elevator deflection, and rudder deflection to the angular rates that serve as inputs to the SFDIA. This prompted us to choose the pilot inputs as explanatory (independent) variables for the models. One other variable recorded during the simulations was the aircraft's forward velocity, measured in mach. Forward velocity was used as an independent variable because of its relationship to the pitching moment of the aircraft. The dependent variables for the regressive models of test trajectories for the SFDIA system were the rolling, pitching, and yawing moments of the aircraft. One should note that the choice of suitable independent and dependent variables is specific for the given domain.

Even though we knew that independent variables from the data sets describing each maneuver would be similar, we still applied a non-hierarchical clustering to each maneuver. This step eliminated extreme values among the independent variables and enforced a certain level of inter-cluster uniformity required by the customer. The representative component and the representative trajectory were selected from the data within the cluster and the regressive models were developed to investigate the applicability of simple linear, multiple linear, and autoregressive models for the automated trajectory generation.

5.2. *Linear model results*

The performance of the linear regressive models built in our experiments is shown in Figs. 3–7. All the figures represent the same maneuver, denoted as Maneuver #8.

Figure 3 shows a simple linear regressive model developed for roll rate prediction from flight Maneuver 8. By using aileron deflection as an input to the model, the model reached the correlation of 91%, whereas using mach as an input to the model only achieved the correlation of 37.4%.

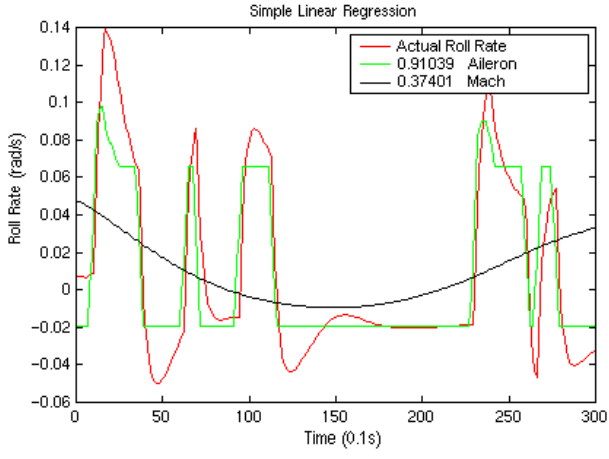

Fig. 3. Roll rate simple linear models.

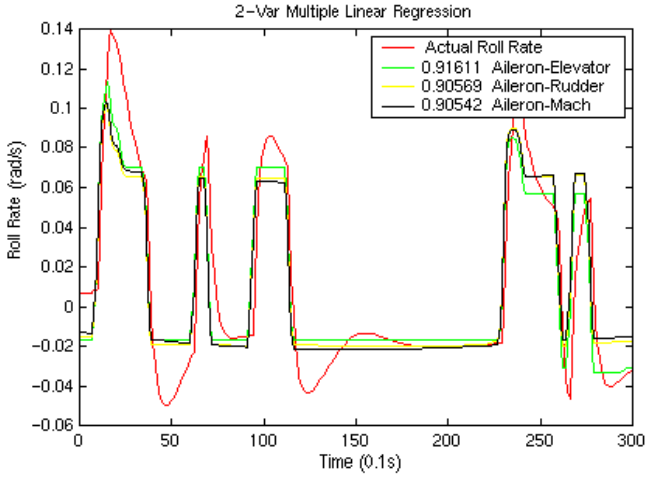

Fig. 4. Roll rate multiple linear models.

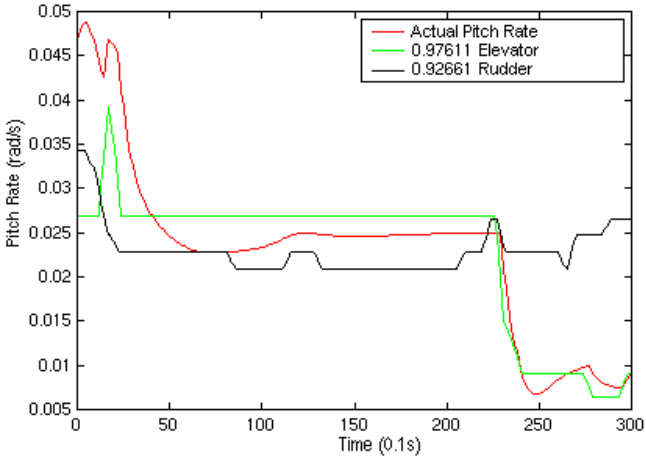

Fig. 5. Pitch rate simple linear models.

Figure 4 shows how the regressive model correlation can change by adding a level of complexity to the simple linear model and using two independent variables instead of one. The two-variable multiple linear regression using the aileron and elevator deflections as inputs has a correlation to the actual dependent variable of 91.6%. If aileron and rudder or aileron and mach deflections are used, the correlation is about 90.6%.

Sample linear regressive models from the pitch rate model development are shown in Figs. 5 and 6. Elevator deflection as an independent variable for a simple linear model resulted in a correlation of 97.6% of the actual value of pitch rate. Rudder deflection input achieved a 92.6% correlation. The addition of a second

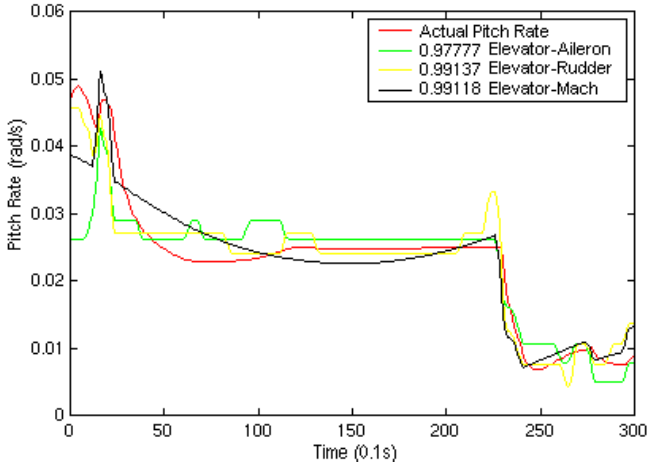

Fig. 6. Pitch rate multiple linear models.

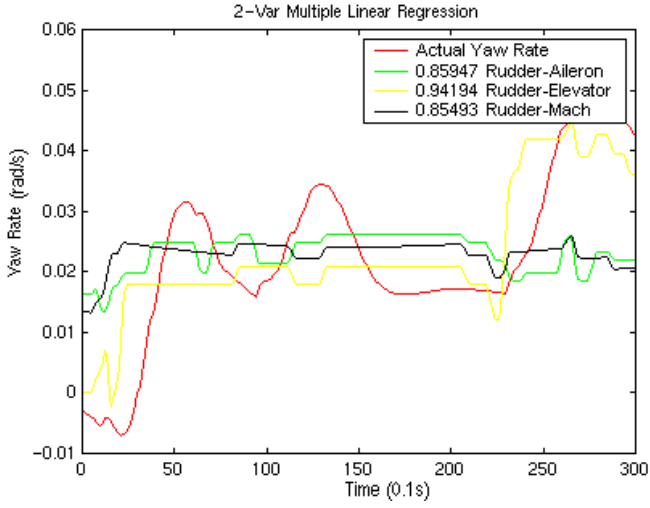

Fig. 7. Yaw rate multiple linear models.

independent variable to the model improved the prediction even further. In Fig. 6, when both elevator and rudder deflection are used, the model correlates 99.14% to the actual pitch rate. The other two-variable combinations perform just as well.

Yaw rate prediction is shown only for a two-variable model in Fig. 7. The linear models had a more difficult time fitting to the yawing moment trajectories. This is expected, due to the mathematical properties of the yawing moment. The best correlation came from the rudder and elevator deflection combination of independent variables, achieving the correlation of 94.19%.

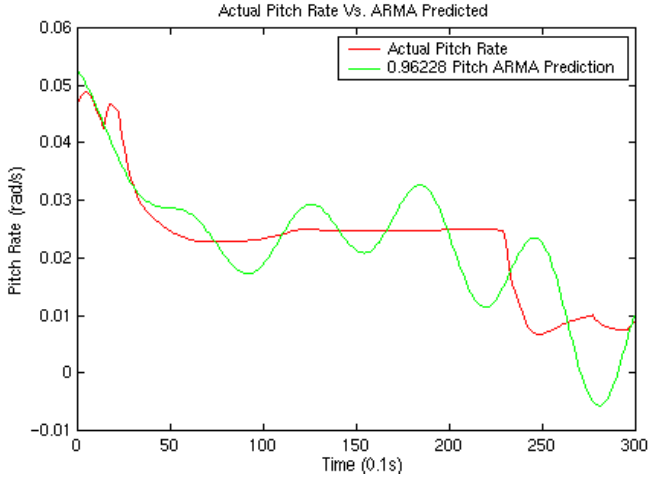

Fig. 8. Pitch rate ARMA models.

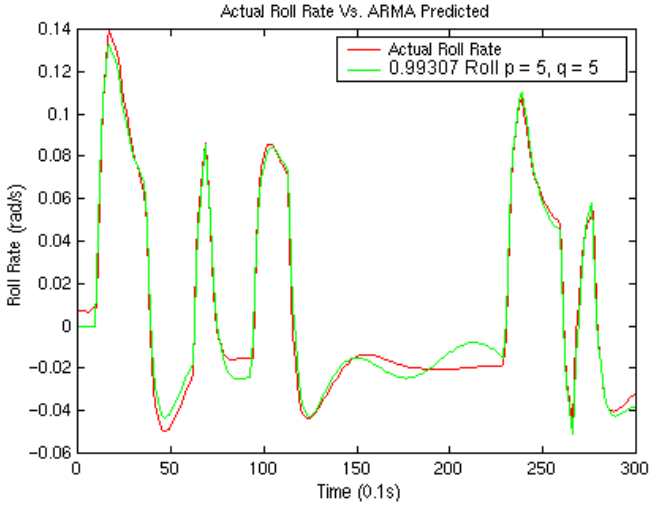

Fig. 9. Roll rate ARMA models.

5.3. *ARMA model results*

Examples of the performance of the ARMA models built in our experiments are shown in Figs. 8 and 9.

Using the same data set from flight maneuver number eight, ARMA models were developed for prediction of the same dependent variables as in the linear models. Figure 8 depicts an ARMA model result for pitching moment. In this case, the ARMA model achieves a 96.23% correlation with the actual pitching moment using the elevator deflection as an independent variable. The high correlation in this

specific case is somewhat misleading, since there is a significant difference between the actual pitch rate and the one predicted by an ARMA model. The cause for this discrepancy and its remedy are discussed in Sec. 6. Generally speaking, using an ARMA model on a highly linear moment, such as the pitch rate, is not desirable.

Figure 9 plots the actual representative trajectory rolling moment from flight maneuver eight against the predicted ARMA model. Correlation between these two is 99.31% and the plot demonstrates the close fitting achievable via the autoregressive technique. Some non-linearity is detectable between 22.5 and 27.5 seconds. Overall, the ARMA model is able to capture the rolling moment well.

Perhaps the best results come from the ARMA models when applied to prediction of the yawing moment. As seen in Fig. 10, the ARMA model is able to predict the representative yawing moment with a correlation of 99.92%. The actual data is matched very well, outperforming any of the results seen by the linear models.

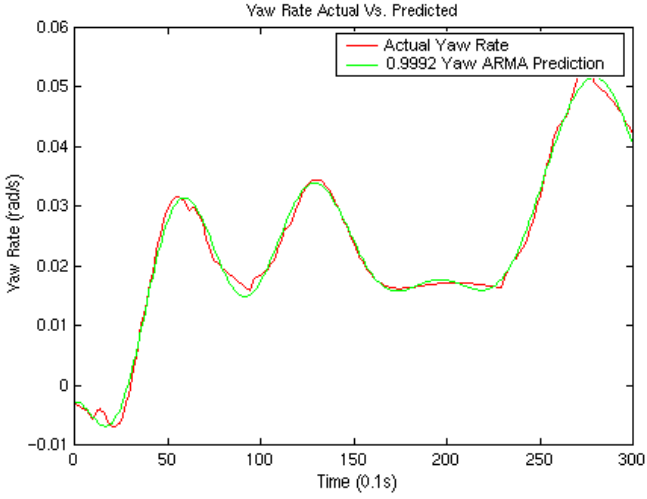

Fig. 10. Yaw rate ARMA Model.

5.4. *Trajectory acceptance rules*

Two sets of acceptability rules were developed for the SFDIA trajectory generation. The first set tests the acceptability of the pilot inputs. The second set was developed to test the reasonableness of the generated roll, pitch, and yaw rates. The acceptability tests applied to the perturbed pilot inputs check the deflections to determine if they exceed the acceptable range which an aircraft might experience during normal flight. The test is defined by an upper and lower limit bound on the pilot's deflection, as shown below:

$$-0.053 \text{ rad} \leq \text{pilot deflection} \leq 0.0053 \text{ rad}$$

If a perturbed input exceeds the threshold, it is discarded from the set of test cases. The choice to use 0.053 radians is based upon typical deflections of surface areas in commercial jetliners.

The second acceptability test checks if the perturbed deflections remain within the clusters determined earlier. By applying the Euclidean distance metric, the distance of each of the perturbed deflections from the cluster centroid is calculated. If a distance exceeded the intra-group variance threshold, then that particular perturbed deflection was eliminated.

The second set of acceptability rules applies to the generated trajectories. A limit is placed upon the degree in which a generated trajectory differs from its corresponding actual angular rate. Essentially, if a generated trajectory falls below an 85% correlation with a trajectory from the original collected dependent variables, that trajectory is rejected.

Table 1 shows the percentages of acceptable generated trajectories for roll, pitch, and yaw rate predictions with each maneuver having its own regressive model developed as either a simple linear or two-variable multiple linear regressive model.

Table 1. Percentage of acceptable trajectories for linear models.

Maneuver	Roll%	Pitch%	Yaw%
1	70.0	90.0	60.71
2	10.0	94.0	66.0
3	91.11	91.11	11.11
4	93.75	93.75	0.0
5	93.75	93.75	62.5
6	80.0	92.5	0.0
7	0.0	85.0	2.5
8	93.75	93.75	93.75
9	90.0	90.0	84.0
10	92.5	92.5	0.0
11	86.25	86.25	17.5
12	61.25	86.25	1.25
13	71.25	93.75	93.75
14	0.0	91.25	60.0
15	78.75	78.75	72.5
16	68.75	75.0	80.0
17	31.25	93.75	12.5

Because each of the generated angular rates are linked to a particular maneuver, the lowest acceptability rate was the limiting factor for some maneuvers. For example, see Maneuvers 4 and 6 in Table 1. Even though there were high acceptable percentages of roll and pitch rate generation, no acceptable yaw rate generation occurred, leaving no new tests for that maneuver. Only a few maneuvers experienced poor results, while most others had very high acceptance rates, as defined by the specific acceptability rules.

Table 2. Percentage of acceptable trajectories for ARMA models.

Maneuver	Roll%	Pitch%	Yaw%
1	39.29	90.0	67.86
2	53.33	96.67	96.67
3	91.11	91.11	91.11
4	81.25	93.75	25.0
5	93.75	93.75	93.75
6	12.5	0.0	0.0
7	12.5	85.0	23.75
8	93.75	93.75	93.75
9	90.0	90.0	90.0
10	92.5	80.0	92.5
11	86.25	25.0	86.25
12	0.0	12.50	12.50
13	93.75	93.75	93.75
14	18.75	91.25	60.0
15	62.5	78.75	78.75
16	81.25	75.0	87.5
17	93.75	75.0	25.0

The percentage of acceptable trajectories generated by ARMA models is shown in Table 2.

6. Discussion

In general the ARMA models performed well in predicting the behavior of the rolling and yawing moments while the linear models performed better at predicting the pitching moments. Much of this has to do with the non-linear relationships among the aerodynamic properties of lateral-directional flying.

6.1. *Evaluation of the regressive models*

Use of the regressive models newly generated trajectories achieved acceptance rates of as high as 90%. It is important to point out that the type of flight maneuvers considered also influenced the individual successes of the different regressive models. Many of the maneuvers selected were chosen because they represented common maneuvers that a commercial jetliner may experience. These included simple banking maneuvers, rolling maneuvers, gentle climbing and diving. It is well established in aerospace literature that many of these maneuvers can be simplified into single-input-single-output (SISO) systems where the aircraft's angular momentum is only affected by a single deflection or input. For these maneuvers, ARMA models had high success rates because they approximate SISO systems well.

Linear models performed generally well for both pitching moment and rolling moment prediction. Correlations from these models averaged around 95% for roll rate and 90% for pitch rate. However, the linear models, both simple and multiple, failed at predicting yawing moments. This is probably due to the highly non-linear

nature of the yawing moments collected from the simulator.

Some of the prediction correlation results for the linear models were very high, above 90% correlation, when visual inspection showed that the model did not closely fit the actual data. This problem is visible in Fig. 8. A cause for this was the use of the Matlab function xcor and the manner in which it normalizes the correlation computation. This function works by calculating the correlation between two separate data sequences for a range of time lags from $-N$ to N where N is the size of the data set. As was the case with all of our cross-correlations between the actual trajectory and the predicted trajectory, our maximum cross-correlation occurred at a lag of zero and progressively got worse for larger lags in either direction. We then used the normalization feature for `xcor`, which ensures that the cross-correlations returned fall between the values of -1.0 and $+1.0$. The small cross-correlation, which occurred at larger lags, caused the results around zero lag to reach higher percentages.

In addition to accuracy, another important factor in evaluating a trajectory generation algorithm is the speed of trajectory generation. The use of ARMA models, for example, showed that they could generate five thousand new test trajectories in the time required to collect one simulated test trajectory. ARMA model computation time was roughly half the time required by the linear model computation. A factor in this could be the usage of provided Matlab functions to compute autoregression, while we wrote our own functions to compute and handle linear model generation. Nevertheless, the generation efficiency of the automated framework surpassed the traditional approach by more then the three orders of magnitude.

6.2. *Evaluation of the acceptability rules*

Prior to the use of an artificially generated trajectory for the system test, the trajectory needs to pass a "sanity check", i.e., it must represent a realistic flight condition. This realism is checked by acceptability rules. The development of the domain specific acceptability rules is crucial as these indicate the true success or failure of the algorithm in generating new trajectories.

The first acceptability rule analyzed the perturbed independent variables to ensure that values for these variables did not violate acceptable ranges that can actually occur for the variables. For commercial jetliners there are upper and lower limits placed upon the deflections that a pilot can give for a surface area, which ensure safe and comfortable travel for passengers. In the case study, this rule eliminated very few of the perturbed independent variables.

The second acceptability rule eliminated perturbed independent variables that would violate the inter-group variance thresholds. That is, if the perturbed independent variables were not within the clustering threshold, they were deemed outside of the cluster model and unacceptable. This rule eliminated around 2% of the perturbed independent variables.

The third acceptability rule applied a test against new dependent variables that

the regressive models generated. One of our requirements for acceptable trajectories was that newly generated trajectories needed to be different, but within a close correlation to the original representative trajectory that was used to build the model. This would limit the input domain coverage for generated trajectories from the developed regressive model, but it increased the "trustworthiness" that those trajectories were valid data sequences that could be used as a test input into the system. This rule had the highest rejection rates, eliminating 8% of rolling moment test trajectories, 4% of pitching moment trajectories, and 7% of yawing moment trajectories.

7. Related Work

A test data generator is a tool that assists a user in the generation of test data [2]. The purpose of the tool is to reduce the testing time by allowing a system developer to generate large volumes of test data [15].

Manual test data generation includes cause-effect graphing, driven by coverage methods, equivalence partition, random user inputs, and use case analysis [15]. Cause-effect graphing is a graphical technique that maps the input domain to the output domain via true or false relationships. Driven-by-coverage methods generate test data with a purpose of increasing one of the coverage measures. Equivalence partitioning divides up the input domain into partitions and chooses test cases for each partition. Random keyboard pounding is a process where testers use the system by giving it as many random inputs as they can to determine problems. Use case analysis is a process where use cases are easily transformed into test cases.

Automatic test data generation can help reduce 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 [2, 5]. Data specification systems generate test data from a language that describes the input domain. The apparent weakness of this approach is the ability of a system designer to adequately describe the input domain, because this may be as difficult as the system requirements definition. Pathwise test data generators generate test data that follow execution paths of the program. While this type of testing is suitable for achieving high path coverage, it is not useful for safety assessment based on the extensive coverage of the critical sections of the input domain. Random test data generators are growing in importance, but their randomness is usually confined to the variations of "important" use cases or scenarios [16]. Specification based test data generation uses grammatical rules on the system specifications to generate test cases.

7.1. *Random test generators*

Random test generators are a common technique to generate large amounts of test data. Random testing uses tests without requiring a priori knowledge of the structure of the program being tested. The entire input domain of the system is

considered and the test generator randomly selects inputs from this domain [3]. The need for such systems has been recognized in the literature and various strategies to enhance the effectiveness of test data generation have been suggested [3, 18–22]. Uniform random test data generation is simple but cannot handle non-uniform operational profiles, which are more frequently encountered in real applications [18, 3]. Some recent research efforts that have considered constrained test data generation have focused on linear constraints expressed in linear algebra [19], in relational algebra [23], or in strictly boolean expressions [21].

For generation of single value data, random test generators provide an adequate solution. However, data sequences cannot be generated. A randomly generated value is unrelated with the next randomly generated value, except for a possible relationship due to the imperfection of the random number generator.

7.2. *Pathwise test data generators*

Pathwise test generators are a common generation technique. A pathwise generator looks at creating test data that will exercise a certain path through the software system [2]. The path reflects the data values throughout the software system as it passes from input to an output.

There are four steps to a pathwise generator: constructing a graph 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 newest technique for test data generation from a program model is based on model checking [24].

Once a path is chosen in the graph, this path is symbolically executed. Symbolic execution identifies path predicates that define regions of the input space. Path predicates are represented by symbolic expressions. The result of the symbolic execution will be an equation in terms of input variables, which, if satisfied, will cause the (symbolic) path to be executed. When the program is to be tested, a test data generator chooses data from within each of the input domain regions. Tests are then checked for correctness either by the test analyst or by a specially designed testing oracle. Sometimes a value cannot be found within the input regions. This indicates that either the path is unreachable or the region was not properly formed. For closed-loop control systems, pathwise testing may not be meaningful. Data might be generated to exercise a path in the system, but a snapshot of data points may not provide sufficient excitation to reveal a safety related fault in a control loop system.

In recent years, several approaches based on the combinatorial automatic test data generation have been unveiled [7, 28, 29]. These tools generate test sequences from system scenario descriptions or some other means of system specification. For example, in [29] a constrained random generator interacts with software at every execution cycle and computes the set of all input values specifying current envi-

ronment specification. This approach is, in principle, applicable to flight control systems. Nevertheless, generated sequences are not trajectories, since at the beginning of every computational cycle inputs are selected such that they are feasible for the given state of the system, regardless of the input history or state variables. We also doubt that these systems are robust enough to generate tests efficiently and appropriately for the goals of safety assurance.

8. Summary

This paper demonstrates one of the first attempts to automatically generate test trajectories, needed for the assessment of process control systems. There is ample evidence that such a tool is needed in practice. Prior to fielding a system (or at any point early in the development life cycle), only a few trajectories may be available for assessment. Enlarging the set of test trajectories is expensive and tedious if a system tester has no automated support.

We described the complete framework for trajectory generation, starting with the collection of preliminary test set through conventional means (regressive tests, simulation), and ending with an acceptance test of the artificially generated trajectory. We evaluated our system in a realistic case study, in which a prototype of a flight control system was undergoing safety assessment. The results show that the trajectory generation algorithm is able to produce new test trajectories faster and cheaper than they could be simulated or collected from actual usage. Currently, our flight trajectory generation algorithm is being considered for inclusion into the automated testing methodology for NASA's Intelligent Flight Control program.

Our study was inspired and guided by the current state of practice of independent verification and validation of software systems as practised by NASA. But it is important to note that while we applied and evaluated test trajectory generation methodology on an embedded flight control system, its intended usage is not limited to aerospace engineering domain. Most software applications in control systems need to maintain the values of state variables across the boundaries of a single run of the control loop. What differs across different domains is the methodology that evaluates the success criteria for testing in general and test trajectory generation in particular. Therefore, (sub)domain, feature and/or test coverage analysis will certainly need to be added to accuracy and efficiency considerations in some of the application domains. The described test trajectory generation framework is flexible enough to address the specifics of various application environments.

Areas of further study include looking at the cost function used to select the best regressive model and modifying it to include additional calculations such as least square errors between the actual and predicted trajectories. Another alternative is replacing the correlation computation with a simple Euclidean distance metric. So far we did not attempt to measure the "goodness" of the generated test trajectories with respect to their ability to exercise specific flight safety features. This will lead to the further refinement and addition to the acceptability rules as applied to the

SFDIA scheme. Last but not least, sophisticated nonlinear regressive models will be constructed to attempt better yaw rate prediction.

Acknowledgements

This work was supported in part by the NSF award CCR-0093315, NASA cooperative agreement NCC 2-979 and by the Institute for Software Research. The views expressed herein are those of the authors and should not be construed to reflect the position of the sponsors.

References

1. B. Cukic and D. Chakravarthy, "Bayesian framework for reliability assurance of a deployed safety critical system", *Proc. 5th IEEE Symp. on High Assurance Systems* (*HASE*2000), Albuquerque, NM, November 2000.
2. R. A. DeMillo, W. M. McCracken, R. J. Martin and J. F. Passafiume, *Software Testing and Evaluation*, Benjamin/Cummings Publishing Co., Menlo Park, CA, 1987.
3. J. W. Duran and S. Ntafos, "An evaluation of random testing", *IEEE Trans. on Soft. Eng.* **SE-10**, 4 (1984) 438–444.
4. J. A. Hartigan, *Clustering Algorithms*, John Wiley, 1975.
5. D. C. Ince, "The automatic generation of test data", *Computer* **30**, 1 (1987) 63–69.
6. R. Jain, *The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley, New York, 1991.
7. N. S. Eickelmann and D. J. Richardson, "What makes one software architecture more testable than another", *Proc. SIGSOFT '96 Workshop*, San Francisco, 1996.
8. M. R. Lyu, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, McGraw-Hill, New York, 1996.
9. Matlab, *Signal Processing Toolbox User's Guide*, MathWorks, Inc. 1996.
10. M. R. Napolitano, G. Molinaro, M. Innocenti, and D. Martinelli, "A complete hardware package for a fault tolerant flight control system using on-line learning neural networks", *IEEE Control Systems Technology*, January, 1998.
11. M. R. Napolitano, C. D. Neppach, V. Casdorph, S. Naylor, M. Innocenti and G. Silvestri, "A neural network-based scheme for sensor failure detection, identification and accomodation", *AIAA Journal of Control and Dynamics* **18**, 6 (1995) 1280–1286.
12. A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, New Jersey, 1989.
13. D. Parnas, A. J. Schouwen and S. P. Kwan, "Evaluation of safety-critical software", *Comm. ACM* **33**, 6 (1990) 636–648.
14. Rockwell International, *Software Requirements Specifications, Flight Design and Dynamics Ascent, Discipline Ascent Subsystem Day of Launch Function* (*DOLILU-II*) *DIVDT Program Function: Ver: 4.2 FD*, March 1993.
15. D. Stearns, "Test data creation", http://www.csc.calpoly.edu/~dstearn/570B/testDataCreation.html, California Polytechnic State University, California, 1996.
16. J. A. Whittaker and J. H. Poore, "Statistical testing for cleanroom software engineering", *Proc. 25th Hawaii Int. Conf. on System Sciences*, Kanai, HI, January 1992, pp. 428–436.

17. A. Avritzer and E. J. Weyuker, "The automatic generation of load test suites and the assessment of the resulting software", *IEEE Trans. on Soft. Eng.* **21**, 9 (1995) 705–716.
18. P. Thevenod-Fosse, H. Waeselynck and Y. Crouzet, "An experimental study on software structural testing: Deterministic versus random input generation", *FTCS 21 Digest of Papers*, Montreal, Canada, June 25–27, 1991, pp. 410–417.
19. R. A. DeMillo and A. J. Offutt, "Constrained-based automatic test data generation", *IEEE Trans. on Softw. Eng.* **17**, 9 (1991) 900–910.
20. D. M. Cohen, S. R. Dalal, A. Kajla and G. C. Patton, "The Automatic Efficient Test Generator (AETG) system", *Proc. Int. Symp. on Software Reliability Engineering, ISSRE '94*, 1994, pp. 303–309.
21. E. Weyuker, T. Goradia and A. Singh, "Automatically generating test data from a Boolean specification", *IEEE Trans. on Softw. Eng.* **20**, 5 (1994) 353–363.
22. B. Korel, "Automated software test data generation", *IEEE Trans. on Software Engineering* **16**, 8 (1990) 870–881.
23. W. T. Tsai, D. Volovik and T. F. Keefe, "Automated test case generation for programs specified by relational algebra queries", *IEEE Trans. on Softw. Eng.* **16**, 3 (1990) 316–324.
24. J. Callahan, F. Schneider and S. Easterbrook, "Automated software testing using model checking", *Proc. 1996 SPIN Workshop*, Rutgers, NJ, August 1996, also available as Technical Report NASA-IVV-96-022.
25. N. R. Draper and H. Smith, *Applied Regression Analysis*, John Wiley, New York, 1981.
26. T. W. Anderson, *The Statistical Analysis of Time Series*, John Wiley, New York, 1971.
27. P. McCullagh and J. A. Nelder, *Generalized Linear Models*, Chapman and Hall, London, 1983.
28. D. Cohen, S. Dalal, J. Parelius and G. Patton, "The combinatorial approach to automatic test generation", *IEEE Software* **13**, 5 (1996).
29. I. Parissis and F. Ouabdesselam, "Specification-based testing of synchronous software", *Proc. SIGSOFT '96*, San Francisco, 1996.