MODEL-BASED DESIGN FOR TEST VECTOR VERIFICATION

Pieter J. Mosterman
The MathWorks, Inc.
3 Apple Hill Dr.
Natick, MA 01760
(508) 647 7765

Rohit Shenoy
The MathWorks, Inc.
3 Apple Hill Dr.
Natick, MA 01760
(508) 647 7972

Jason R. Ghidella
The MathWorks, Inc.
3 Apple Hill Dr.
Natick, MA 01760
(508) 647 7573

Brett Murphy
The MathWorks, Inc.
3 Apple Hill Dr.
Natick, MA 01760
(508) 647 7150

Abstract - To meet the needs of a competitive marketplace, Model-Based Design is increasingly being adopted by companies. The use of computational models throughout the system design process accelerates development and increases quality by executable specifications, broader and automatic exploration of the design space, and high-level verification and validation. This paper shows how coverage technologies used in Model-Based Design can be exploited in the verification and selection of test vectors used to troubleshoot faulty equipment.

INTRODUCTION

The lifecycle of an engineered system typically consists of a design and a deployment stage. In the design phase, requirements are compiled and specifications are derived from them. The specifications are then increasingly refined until they can be successfully implemented. The design phase concludes upon successful implementation, after which the system is deployed [1].

Much of the total cost over the lifecycle of a system is expended in the deployment stage. To keep this cost low, efficient maintenance is required. To this end, problems in the system must be easily reproduced and analyzed so they can be traced back to the root-cause faulty components.

In order to efficiently find the faulty components in a system that operates erroneously, systematic analysis is required. This analysis often takes place based on a search tree. At each node of the tree measurements can be taken. Which path is taken through the node hierarchy depends on the results of the measurements, The leaves of the search tree are unit components. The faulty component is identified when the search arrives at a leaf of the search tree.

These search trees are typically statically determined. The process to design such search trees is often exclusively based on the final system to be deployed. Tests to go into the search tree are determined in a pre-deployment phase, that tends to be dissociated from the design process.

Traditionally, either heuristics or experience, or a combination of the two, is used to diagnose faults in the deployed system. Such approaches fail to take advantage of the analyses performed in the design phase of a system, although in the design extensive testing is performed as well.

With the increasing use of Model-Based Design [2], technologies and methods that are exploited in the design of a system can be carried over in the verification of test vectors.

Modeling has always been an important aspect in the design of systems, from early stage mock-ups to detailed scaled models of the design. The availability of abundant computing power has enabled the use of computational models instead of the use of physical models. The advantages of computational models include:

- The design can easily be changed.
- Automatic synthesis methods based on models can be designed.

- Automatic design space exploration becomes possible.
- Operating regions that are difficult if not impossible to achieve in reality can be investigated.
- Test vectors can be automatically generated.

Model-Based Design is increasingly being exploited in the design stages where the requirements are decomposed into subsystems, modules and components, for which an implementation is then derived. Once the decomposed parts are implemented, the integration tasks consisting of verifying the overall objectives of the design, as well as the calibration and tuning of the system, and systematic testing at different levels of system hierarchy, can also take advantage of Model-Based Design.

The use of models allows design knowledge about the system to be re-used in setting up the maintenance schedule and analyses. This paper presents the use of Model-Based Design technologies in the verification of test vectors. Instead of using the physical components, computational models of those components are used for the verification of the test vectors. This has the following advantages:

- Test vectors can be derived before the physical hardware is available. This allows an early establishment of a test baseline of expected results based on the test excitations of the model.
- The verification of those tests can be done faster than real-time. With a computational model, simulations are not tied to physical time, and simulated time often can move faster than real-time.
- Scenarios that are difficult to reproduce in reality can be examined, because in computational simulation, reality can be manipulated as desired.
- If computational models are used in the design stages, the earlier modeling efforts can be reused in the test verification.

A case study of verifying test vectors for a Stewart platform is presented. Using a computational model, sensor and actuator failures can be easily introduced and verified against the test vector design. Moreover, arbitrary fault models can be used such as shorted electrical wiring, non-responsive actuators, changes in actuator characteristics, and unanticipated wear and tear.

Once tests have been designed and verified using computational models, the test design verification stage using the actual hardware can be expedited. As a result, less time is spent on those more costly activities of working with actual hardware.

Section 2 introduces the example system used in this paper, a Stewart platform. Section 3 discusses a model of the Stewart platform. Section 4 introduces coverage analyses to quantify the system parts that are excited by the test vector set. Section 5 outlines how coverage can be applied in testing. Section 6 applies coverage results to test vector selection for a Stewart platform. Section 7 presents conclusions of this work.

2.0 A STEWART PLATFORM

A Stewart platform consists of six legs that are attached to a common platform, configured as shown in Fig. 1. Such Stewart platforms are often used in aircraft and automobile simulators. The platform has six degrees of freedom, three translational and three rotational, and can be moved so as to give the impression of being inside, e.g., the cockpit of an airplane.

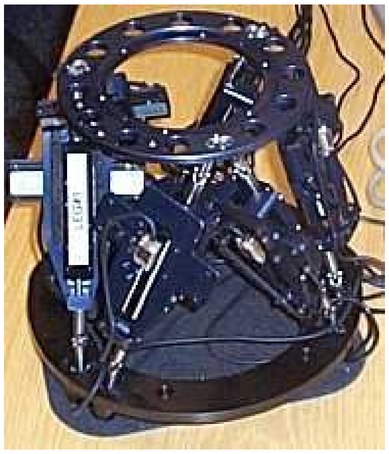
Figure 1: A Stewart Platform

Each of the six legs contains a prismatic joint and is equipped with a motor that allows controlling the extension of the leg. By changing the extension, the platform can be positioned as desired. Because of the six legs, the system is sometimes called a “hexapod.” In Fig. 2, the Stewart platform plant and the xPC TargetBox® [11] controller are shown as blocks in the center. The xPC TargetBox hardware is centered around a 400 MHz Intel Pentium III (floating-point) processor, with 128 MB RAM, and 32 MB flash RAM.

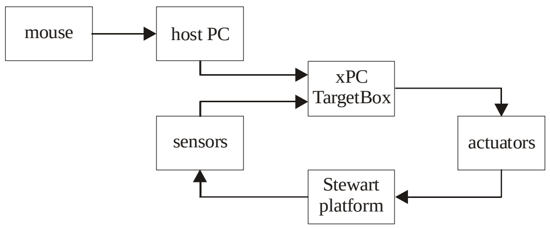

Figure 2: Stewart Platform Hardware Configuration

The control algorithm that is implemented on the xPC TargetBox computes the control signal for each of the actuators. The signal is first input to a Nanomotion amplifier that is connected to one of the six channels of the RTD DM6604 analog output of the xPC TargetBox to achieve a high-power equivalent that can drive the Nanomotion H1 piezoceramic motors on each of the legs. The motor input consists of a high-power sinusoidal voltage, the amplitude of which is used to control the leg movement.[1]

The sensor input to the xPC TargetBox comes from six MicroE Mercury 3110 incremental encoders,[2] one on each leg. Each time the leg slides a given distance, a pulse is produced by a grated slide passing by an optical beam. The generated pulses are counted by an RTD DM6814 incremental encoder board and translated into a corresponding digital value from which the actual distance is computed. The precision is determined by the total of 65,536 counts over about 4 cm of travel (about 0.61 µm of travel per count).

To calibrate the slide, a zero location is marked from which the counting is to start incrementally. In the given hardware setup, the reference marking is not detected by the encoder pickups, and, therefore, a pure incremental count is available. To still provide a reference, the actuator sliders are moved against their hard stops, and that is defined to be zero.

[1] http://www.nanomotion.com/data/docs/Tech\%20notes\%20102.pdf

[2] http://www.microesys.com/

3.0 MODELING THE STEWART PLATFORM

There are two basic approaches to plant modeling: (i) physics modeling and (ii) empirical modeling. In the first approach, a model of a physical system is designed based on the underlying physics involved. For example, laws of physics such as conservation of momentum may be applied. In the second approach, a mathematical representation is found that matches the data measured on the physical system, given a number of sets of excitations. Note that, in general, modeling proceeds by combining aspects from the two approaches.

3.1 Physics Modeling

Modeling of physical behavior from first principles is supported by many computer aided design (CAD) tools. For example, SimMechanics [4] is a dedicated product for modeling multi-body systems. As such, it can be used to model the dynamics of mechanical machines. The models that can be designed using SimMechanics inherently satisfy, e.g., conservation laws.

For modeling the Stewart platform, a SimMechanics model of the machine structure was exported from a CAD drawing in SolidWorks [8]. The resulting SimMechanics model has a base plate as *body* at the bottom that is rigidly connected (by a *weld* joint) to ground. The six legs connect the base plate to the top plate. The legs are modeled by *bodies* that connect a *prismatic* joint to the *spherical* joints at the base and top plate. The prismatic joints can be actuated and are modeled to include stiction and viscous friction effects.

The SimMechanics model of each of the legs is shown in Fig. 3. It shows two bodies, *Lower Leg* and *Upper Leg*, that are connected by means of a prismatic joint. The *Connection Port 1* port connects the leg to the base and the *Connection Port 2* port connects the leg to the top plate of the Stewart platform. The prismatic joint is modeled by the *Prismatic Joint* that connects the two parts of the leg. Friction in the joint is modeled by the *Joint Actuator* that is connected to the *Force* input port. The position and velocity of the prismatic joint are obtained by a *Joint Sensor*.

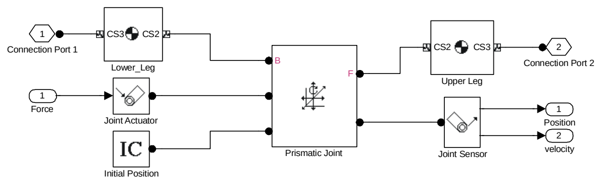

Figure 3: A SimMechanics Model of one Leg of the Stewart Platform.

Note that the connections in the SimMechanics domain as shown in Fig. 3 are not directed, unlike Simulink® [5] signal lines. SimMechanics connections are not directed because they carry two conjugate variables and it is not known a priori what the computational direction (the computational causality) of those variables is. Contrary to Simulink, it is not necessary to determine this computational causality when modeling as the SimMechanics compilation engine can automatically determine it [10]. Since the connections are not directed, a direction-neutral line ending (such as a solid circle or a square) is used.

3.2 Estimating Plant Model Parameters

Once the structure of the model is designed based on modeling from first principles, the parameter values of the model have to be determined. This is typically done by measurements. Those measurements can be static (e.g., lengths, widths, gear ratios, and diameters) or dynamic (e.g., inertias).

Parameters for some physical phenomena are difficult to obtain by static measurements. For example, kinetic friction and stiction coefficients require measurements of the plant dynamics to estimate. Obtaining such dynamic data may require a fairly elaborate experimental setup [3].

Once the data is obtained, parameter estimation tools such as Simulink Parameter Estimation [6] can be applied to systematically find the values for the free parameters and initial states that result in behavior that best matches the measured data. To ensure correct and quick convergence to the optimal values, the minimum and maximum values of the parameters and initial states can be set, as well as their expected values.

Figure 4 shows the measured extension of one of the legs of the Stewart platform for a sequence of control commands as a dotted line, while the extension as computed from the model is shown by a solid line. In these results, the model parameters were chosen based on an educated guess.

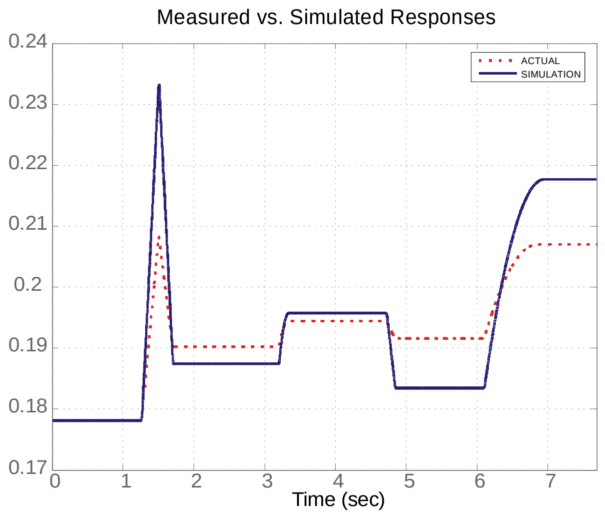

Figure 4: Model and Actual Behavior Before Estimating Parameters From Measurements.

Figure 5 shows the leg extension computed from the model in response to the same sequence of control commands after the model parameters have been estimated by Simulink Parameter Estimation to minimize the difference between the measured leg extension (shown by the dotted line) and the model leg extension (shown by the solid line).

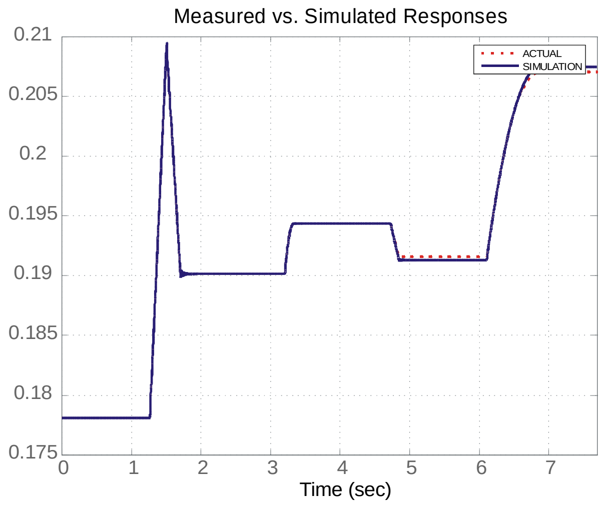

Figure 5: Model and actual behavior after estimating parameters from measurements.

4.0 TEST VECTOR GENERATION AND MODEL COVERAGE

Once a model is obtained that is an accurate representation of the system, it serves as a gold standard that can be used to design test vectors.

The input signals can now be varied across their signal ranges and the subsequent output ranges measured. These measurements provide a basis for the limits of the tests under normal operation. In addition to getting the output range for a given input, the use of a model allows access to coverage information of variables internal to a model that are not normally measurable. Simulink Verification and Validation [7] shows coverage of the model for a given input vector and provides the following coverage analysis metrics:

- For points in the model at which logical decisions are made, decision coverage is reported. This includes Simulink switch blocks and Stateflow® [9] states.
- For blocks that output a logical combination of the input, condition coverage is reported. This includes Simulink logic blocks and Stateflow transitions.
- Look-up table coverage identifies the frequency that interpolation intervals are executed and finds hot spots where the table could be made more detailed
- Modified condition/decision coverage (MC/DC) reports whether the output has been independently changed by the logical input. MC/DC is a crucial part of safety-critical software analysis as defined in RTCA DO-178B.[3]
- Signal range coverage reports on the minimum and maximum values that are achieved during simulation. These values are provided for each block output as well as Stateflow data objects.
- Cyclomatic complexity indicates the structural complexity of a model. This approximates the McCabe complexity measure of the code that is generated from the model [7].

These coverage metrics give the test designer the necessary information to prove that their test set fully exercises the device under test (DUT). They can easily see if all possible paths in the model are covered and which test vector covers which part of the model. Traditionally this information relied on the test strategy report to cover the full system but now the test programmer can see proof of the coverage for their given test vectors.

[3] http://www.rtca.org

5.0 TEST VECTOR COVERAGE OF SYSTEM FAILURES: DIAGNOSABILITY

In addition to using a model to simulate nominal behavior, the model can simulate known faults. This introduction of faults will enable test engineers to measure the output behavior for a given input signal with the introduction of a fault, thereby gaining a greater insight into the effect of the fault.

By introducing faults and looking at the outputs the test engineer can begin to understand the types of tests that need to be made to determine the fault. For example one type of fault may cause an output to be outside the nominal range whereas other types of faults may only cause the output to react slower to a given input. Each type of change to the response requires different types of measurements to determine the fault. This determination is easier to visualize using a model because a variety of faults can quickly be simulated and access to internal model variables is available.

One possible use for the introduction of the fault into a model can also be to determine the sensitivity of the measurement for a given test vector with that fault. This sensitivity can be calculated using the minimum (min) and maximum (max) values for a given output during normal operations versus the output signal measured with the fault by:

$$\frac{(Output_{fault} - Output_{\min})^2 + (Output_{\max} - Output_{fault})^2}{(Output_{\max} - Output_{\min})^2}$$

This equation will give a number below 1 for a measurement that is within the nominal range even with the fault and a number larger than one for measurements that are outside the range.

This sensitivity can then be calculated for all the output signals and all the different test vectors for all known faults easily by utilizing the model and the test vectors in a batch processing environment.

In addition to investigating faults, the model can be utilized to produce coverage information for the given test vector to help decide the appropriate test to run. As an example, if there are two test vectors with test 1 covering only 10% of a subsystem and test 2 covering 70% of the same subsystem., then, if the subsystem becomes indicted the appropriate approach may be to run test 2 first.

Lastly, the test engine can combine the coverage data for each given test and the sensitivity of the observables for a given indictment. One criterion used to help guide the test engineer in selection of the appropriate test to run for a given indictment can be to multiply the sensitivity with the coverage number.

An example of a possible test vector providing a coverage number and a deviation from the nominal is shown in Table 1. If the coverage is multiplied with the deviation from nominal, the test vector 2 should isolate the fault best, then test 3, test 4 and test 1 the least optimal.

Table 1. Test Coverage.

Test	Coverage	Sensitivity
1	10%	0.5
2	20%	11
3	70%	2
4	50%	1

In addition to this information the isolation of one type of fault versus a different type of fault needs to be considered. This information also can be obtained easily through further simulation of the model with other faults.

6.0 TESTING THE STEWART PLATFORM

Given a model of the DUT enables the test engineer a wealth of information about the DUT that traditionally only the designers of the DUT had, but without the test engineer needing to understand the full complexity of the system. They also now have the ability to see the full impact of each test vector and better understand the test strategy.

This is illustrated for the Stewart platform discussed in Section 2. The test vectors in Fig. 6 are available to troubleshoot anomalous behavior. In Fig. 6(a) a ramp signal is shown that can be used to determine when the forward stiction of a motor is broken. In Fig. 6(b) a step is shown that has a large amplitude to simply extend the corresponding leg. In Fig. 6(c) a pulse signal is shown that can be used to test positive and negative leg extensions. In Fig. 6(d) a triangular signal is shown that allows determining when the negative stiction is broken. If chosen at a sufficiently low amplitude, as determined from the model, it can also be used to determine when the forward stiction is broken. In Fig. 6(e) a frequency sweep is sweep that can be used to test overall parameter values.

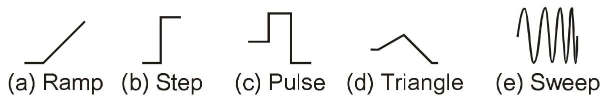

Figure 6: Stewart platform test signals.

A subset of coverage results for the tests in Fig. 6 is shown in Table 2. The coverage results can be used to guide the testing process. For the Stewart platform, five components can be identified for each of the six legs: (i) the motor, (ii) the amplifier, (iii) the bearing, (iv) the sensor, and (v) the counter. For the motor, three separate aspects are of importance: (i) a change in viscous friction, (ii) a change in forward stiction, and (iii) a change in backward stiction.

Table 2. Stewart Platform Test Coverage.

Test	Stiction Coverage	Velocity Range	Acceleration Range
Ramp	67%	[0, 0.00630892]	[0, 0.0138253]
Step	67%	[0, 0.00606338]	[-1.51543e-5, 37.6943]
Pulse	100%	[-0.0496916, 0.0432256]	[-25.5148, 13.2399]
Triangle	100%	[-0.00597691, 0.00628716]	[-0.0496711, 1.64566]
Sweep	100%	[-0.028824, 0.0247618]	[-1.11488, 1.1528]

To troubleshoot a problem down to the component level, it is first tested whether the amplifiers for each of the legs work properly. The coverage results in Table 2 indicate that the best test for this is to use the step input with a large amplitude because it has a large range of accelerations, and, therefore, large forces to overcome potential failures in the motor and legs.

In response, the position of the platform is measured and in case there is no motion, the amplifier is a likely candidate for failure. Note that acceleration ranges are difficult to measure on hardware and so this information on test effectiveness would be difficult to obtain otherwise.

If the platform does move, the amplifier is likely to be functioning correctly, and so the motor could be dysfunctional or there could be a fault in the bearing. To test whether the motor is functioning correctly, first, the stiction coefficients can be validated. The coverage results in Table 2 show that because of the low acceleration range, the ramp and triangle tests are prime candidates because they will provide more accurate information about when the stiction is broken than tests with larger acceleration ranges.

To test both the forward and backward stiction, the triangle waveform is selected, because it gives coverage of both those situations, as can be seen by the coverage column in Table 2. Instead the ramp test only tests the leg being stuck and the forward friction being broken, i.e., only 67% coverage is achieved.

If the stiction coefficients do not deviate, a next candidate for testing is the viscous friction of the motor and the bearing. These two are closely coupled, and, therefore, if the test shows anomalous behavior, both the motor and bearing are implicated and the mechanism has to be dismantled to perform further testing. This test is best performed by the frequency sweep, because of the large velocity range that it causes, as can be seen in the coverage results in Table 2.

7.0 CONCLUSIONS

The use of models in industry to improve the design process has become common place. Model-Based Design has proven to be of great benefit and has become a requirement to remain competitive.

This paper discussed how some of the Model-Based Design technology, in particular coverage analysis, can be applied in verification of the test vector set. To this end, a system model is used to generate coverage results and these are related to potential faults in the device under test. Among the benefits are having access to otherwise difficult to obtain measurements, such as accelerations, to establish the discriminatory ability of test vectors.

Further use and advantages of models in test vector design and verification has been discussed. These usages will be explored in future work.

8.0 ACKNOWLEDGMENTS

The authors wish to thank Tom Gaudette for his help in preparing this paper.

REFERENCES

[1] D.J. Hatley and I. Pirbhai. *Strategies for Real-Time Systems Specification*: Dorset House Publishing Co., New York, New York, 1988.
[2] J. Krasner. Model-based design and beyond: Solutions for todays embedded systems requirements. Technical report, Embedded Market Forecasters, American Technology International, Framingham, MA, Jan. 2004.
[3] P. J. Mosterman, S. Prabhu, A. Dowd, J. Glass, T. Erkkinen, J. Kluza, and R. Shenoy. Embedded real-time control via MATLAB, Simulink, and xPC Target. In D. Hristu-Varsakelis and W. S. Levine, editors, *Handbook on Networked and Embedded Systems*, chapter 3.4, pages 419–446. Birkhäuser, Boston, MA, 2005.
[4] *SimMechanics User's Guide*: The MathWorks, Natick, MA, 2004.
[5] *Using Simulink*: The MathWorks, Natick, MA, 2004.
[6] *Simulink Parameter Estimation User's Guide*: The MathWorks, Inc., Natick, MA, 2004.
(7) *Simulink Verification and Validation User's Guide*: The MathWorks, Inc., Natick, MA, 2004.
(8) *Introducing SolidWorks*: SolidWorks Corporation, Concord, MA, 2002.
(9) *Stateflow User's Guide*: The MathWorks, Natick, MA, 2004.
(10) G. D. Wood and D. C. Kennedy. Simulating mechanical systems in simulink with SimMechanics. Technical Report 91124v00, The MathWorks, Inc., Natick, MA, 2003.
(11) *xPC TargetBox User's Guide*: The MathWorks, Inc., Natick, MA, 2004.