RIGOROUS MODELING AND SIMULATION OF MECHATRONIC SYSTEMS

James H. Taylor
Professor Emeritus, Systems and Control
Department of Electrical and Computer Engineering
University of New Brunswick
Fredericton, NB CANADA E3B 5A3
E-mail: *jtaylor@unb.ca*

ABSTRACT:

A brief overview of modeling and simulation (M & S) technology is presented, describing the development of M & S in a variety of industrial contexts, such as aerospace, manufacturing, and process industries. Typically this occurs in phases: introduction of M & S, shaping M & S to meet an industry's needs, and maturation. In industry after industry M & S has become increasingly sophisticated, increasingly utilized, and increasingly important – to the point that in industries where M & S is mature it is one of the keys to product design, development, improvement, and, in the end, increased competitive position and profit margin.

The technical side of M & S is also discussed, in more detail. Important points include the selection of modeling and simulation methods for various application areas, and the state-of-the-art in simulating dynamic systems, with a particular focus on mechatronic systems. Emphasis is placed on rigorous techniques and selecting the most appropriate method for a given problem. The final goal is to aid in making M & S an effective tool for achieving the benefits mentioned above for mechatronic systems.

Key Words: Mechatronic systems, modeling, simulation, numerical methods, discontinuity.

Main Subject Area: 19 Mechatronics & Electromechanical & Automation

1. Introduction

Computer-based modeling and simulation (M & S) has been under steady development for about 50 years now. In one form or another, this methodology has become ever more important in an ever broadening arena of applications. Here we overview a part of this wide field, the modeling and simulation of dynamical systems, specifically, systems that can be appropriately modeled by ordinary differential equations (ODEs), partial differential equations (PDEs), and differential algebraic equations (DAEs). We will not deal with continuous-time systems interfaced with discrete-time algorithms (DTAs), as DTAs can be faithfully emulated in a digital simulation (including effects due to word length, execution time etc.) in a relatively straightforward manner. Some aspects of the technical discussion of modeling are, of necessity, quite sketchy; the more comprehensive presentation of simulation methods focusses primarily on ODE systems.

Significant utilization of dynamic M & S has started at different times and progressed at different rates in various application areas. As a generalization, the early years of M & S were focussed primarily on aerospace and military applications – i.e., high-tech and expensive technological systems, where the engineering analysis and design effort was substantial and the M & S effort was essential and could be afforded. Then M & S spread to smaller high-tech and civilian arenas – robotics and transportation systems coming immediately to mind. At the present, it is quite safe to state that M & S, in one form or another, is applied to almost every area of human endeavor, from agriculture to biology to economics to manufacturing . . . to social systems to zoology – although the approaches, the degree of maturity and the pervasiveness varies greatly from area to area. Regarding affordability, in many applications today, most engineers feel that they cannot afford **not** to perform M & S.

A scan of contemporary literature in the field reveals that the motivation and goals of M & S are essentially simple and universal: M & S is utilized to *understand* the behavior and to *improve* the behavior of dynamic systems. However, the specific focus and approach differs significantly from application to application. This is due to (1) the different types of models required for studying different types of systems, and (2) the different types of investigations that are undertaken. Therefore, in virtually every area, M & S has been refined, extended and customized to meet the specific needs of that discipline. By now, these contributions make up a corpus of techniques and tools/algorithms that is truly vast.

Finally, there are numerous driving forces for the rapidly increasing necessity and popularity of M & S. Five general, dominant factors are:

- the increasing complexity of advanced technological systems;
- the continuing need to achieve better process or product performance;
- the growing need for competitive advantage, e.g., efficiency, economy;
- the phenomenal increase in available computer power and decrease in cost; and
- the coupling of M & S with other powerful computer-based methods, e.g., optimization.

These all provide strong motivation for M & S of mechatronic systems.

For a graphic and well-known illustration of these trends, we have only to look at the current standard practices in the automotive industry: The performance (not only power and handling, but emission control, safety, comfort, etc.) has increased greatly in the last 25 years, the complexity of the entire package (engine/powertrain/chassis/features) has increased at least 10-fold in that same time-frame (if you still do all your own car maintenance, please raise your hand), and the fuel economy *vs* performance ratio has (by mandate) also increased significantly. This evolution led to a hundred-fold increase in M & S, including use in conjunction with large-scale optimization.

For mechatronic systems the modern digital camera provides another well-known example. Over the last 50 years changes have been dramatic – my first camera had a fixed-focal integral lens and did not even have a built-in light meter. Now we expect (or hope for) a 10x optical wide-angle zoom lens; a "smart portrait system", which automatically detects the subjects face, takes a picture when they smile and warns you if they blinked, plus red-eye fix and face priority automatic exposure; an image stabilization system for both camera shake and moving subjects, and more . . . the huge number of electro-mechanical components in a modern digital camera (all super miniaturized) is staggering.

2. Modeling Overview

Modeling is not the primary focus of this presentation, so what follows is, of necessity, merely an informal review of basic ideas. This will provide the context for the discussion of simulation methods later. First, some terminology:

1. *"Hard" modeling* – using scientific principles (Newton's law, Kirchhov's laws, laws of thermodynamics, reaction kinetics, etc.) to derive an analytical model. The feasibility of doing so varies a great deal from one discipline to another. This procedure may be said to be "easy" for electro-mechanical systems (e.g., robotics) and "very difficult or impossible" in some biological application areas. In fact, these lines are not entirely clear, since some biological phenomena can be modeled readily from first principles, and some effects in electro-mechanical systems cannot (friction being a notorious example). Of course, someone who has spent a year or more developing and validating a realistic, detailed model of a robot might disagree with the characterization "easy", but by that is meant only that the physical principles and approach are well established.
2. *"Soft" modeling* – using formal or informal fitting techniques plus intuition to match a mathematical model's behavior to empirical data/observations. The Lotka-Volterra competition equations [1] for population dynamics are a well-known "classical" example of this process.
3. *Model identification* – A large number of methods and software packages exist for model identification, informally described as determining model structure (e.g., order) and parameters based on time-series data of an object's or process' input/output behavior. These include frequency response (nonparametric) modeling, regression, least squares techniques, maximum likelihood, instrumental variables – for a comprehensive coverage, see Ljung [2].

Developing a system model may well involve a combination of the above approaches. "Hard modeling" produces a so-called "white box" model, and a model based solely on determining the parameters of a model of assumed (nonphysical) structure is called a "black box" model. Obviously, a combination of hard and soft modeling or model identification produces a "grey box" model; this is done quite commonly.

The modeling process outlined above produces a set of ODEs, DAEs or PDEs that characterize the continuous-time part of the system (process) plus perhaps DTAs describing the behavior of digital components interacting with the process. Since DTAs can be faithfully emulated in a digital simulation (including effects due to word length, execution time etc.) in a relatively straightforward manner we only focus on handling continuous-time systems or the continuous-time part(s) of systems. After a few general comments regarding DAEs and PDEs, we further restrict our attention to ODEs.

First, we distinguish between ODEs and DAEs as follows: A generic form of a set of DAEs and output equations may be expressed as:

$$0 = F_c(x_c, \dot{x_c}, u_c, t) \tag{1}$$
$$y_c(t) = H_c(x_c, \dot{x_c}, u_c, t) \tag{2}$$

where x_c is the state vector, y_c is the output vector, u_c is the input signal (continuous-time), and t is time; in general u_c is a vector. We observe that the Jacobian $F_{\dot{x_c}} \overset{\Delta}{=} \partial F_c / \partial \dot{x_c}$ is usually identically singular; otherwise the system in Eqs. (1,2) can be treated as an ODE set [3].

The form in Eqs. (1,2) is called a *fully implicit* DAE [3]; without imposing additional conditions or constraints, it generally cannot be solved by any existing numerical code. In fact, determining if such a model is solvable [3, 4] and arriving at consistent initial conditions [3, 5] is a complicated matter. To achieve a practical definition of the class of continuous-time models to be treated, we need to specialize the form in Eqs. (1,2) in some manner:

- Most simply, we may replace the DAE form in Eqs. (1,2) with the following ODE set:

$$\dot{x_c}(t) = f_c(x_c, u_c, t) \tag{3}$$
$$y_c(t) = h_c(x_c, u_c, t) \tag{4}$$

  Such models have been the focus of most commercial modeling and simulation environments in the last five decades [6, 7, 8, 9].
- Next most simply, we may replace the form in Eqs. (1,2) with the following *constrained* ODE set:

$$\dot{x_c}(t) = f_c(x_c, z_c, u_c, t) \tag{5}$$
$$0 = g_c(x_c, z_c, u_c, t) \tag{6}$$
$$y_c(t) = h_c(x_c, z_c, u_c, t) \tag{7}$$

  where constraint variables z_c have been added along with constraint equations Eqs. (6). Models of this form are called *semi-explicit* DAE*s*. It has been shown that ODE solvers can be used for simulating this simplest class of DAEs [10, 11]

It is important to note that the choice of continuous-time model form is not hard-and-fast; rather, it is a matter of judgment about what is required to attain suitable "realism". Very basic decisions regarding realism include not only the model type, but also what order of model is needed (how many state variables) and what nonlinearities need to be included. Such decisions may also effect solvability; for example, it is usually not a good practice to develop an ODE model that includes both very fast and very slow dynamics (called a "stiff" model [12]), since they are hard to simulate, with some solvers taking a lot of computer time, other solvers failing completely.

One way to avoid a stiff model is to convert the "very fast dynamics" into algebraic constraints, thereby converting stiff equations (3,4) into DAEs Eqs. (5,6,7). In some cases this may be trivial (e.g., neglecting the fast dynamics associated with a servomotor's inductive lag is simply done by setting the inductance to 0 and eliminating the corresponding current as a state variable) or easy – for example, if the states are separable into fast and slow states, $x_{fast}\,, x_{slow}$ then:

$$\begin{aligned} x &= [\,x_{fast}\ x_{slow}\,]^T \\ \dot{x}_{fast} &= f_{fast}(x_{fast}, x_{slow}, t) \\ \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \end{aligned} \tag{8}$$

yields the semi-explicit DAE set

$$\begin{aligned} \dot{x}_{slow} &= f_{slow}(x_{fast}, x_{slow}, t) \\ 0 &= f_{fast}(x_{fast}, x_{slow}, t) \end{aligned} \tag{9}$$

Where such a direct state decomposition is not feasible, the conversion process is not as simple but still worthwhile and done routinely in some applications, such as modeling aircraft engines. Some stiff system integration algorithms perform such a decomposition automatically [12].

Another problem that gives rise to DAEs or that requires a clever work-around is the existence of "algebraic loops". A simple example of this problem – and a common source of difficulty – is the standard control system where the open loop transfer function is not strictly proper (i.e., the numerator order is the same as the denominator order). Consider the open-loop transfer function $G(s) = (s+1)/(s+10)$ with input u and output y in a unity feedback system with command input r; a model for the closed-loop system is:

$$\begin{aligned} \dot{x} &= -10x + u \\ y &= -9x + u \\ u &= r - y \end{aligned} \tag{10}$$

Evidently there is a circularity in these equations, as one must know u to evaluate y and *vice versa.* In many environments (e.g., MATLAB, Simnon) you are not allowed to program an ODE model with algebraic loops; however, Eq. 10 may be treated as a DAE, or the work-around is to include some additional strictly proper dynamics in the loop, such as $100/(s+100)$. This could represent actuator or sensor dynamics, for example.

The fundamental distinction between PDEs and ODEs is that the former represent variation over both time and space (in one, two or three dimensions) rather than time alone – hence the need for partial derivatives, $\partial/\partial t$ as well as $\partial/\partial x$ and perhaps $\partial/\partial y$ and $\partial/\partial z$. An alternative nomenclature is *distributed-parameter* model for a set of PDEs and *lumped-parameter* model for a set of ODEs – in other words, the physical phenomena take place over a spatial distribution in a PDE but are "lumped" into single points in space in an ODE.

Digital simulation of a PDE set is itself a vast discipline, far beyond the scope of this presentation. Over the past 50 years very powerful techniques and algorithms have been developed for their solution, finite element methods and software being the best known and most utilized [13]. Here we merely observe that, as in the case of contrasting ODEs and DAEs, the decision to model using PDEs is again not firm. Instead, the requirement for realism for the problem being studied is the basis for selecting the modeling approach. For example, if one is modeling a mechanical drive system containing a flexible shaft, one may assume that the shaft is so rigid that flexibility may be ignored, or one may represent the flexibility using one lumped-parameter spring, or several such springs, or one may use a PDE representation. Considering the n lumped-parameter spring model, $n = 0,\ 1,\ 2,\ \ldots,$ the $n = 0$ option corresponds to a rigid shaft, $n = 1$ accounts for the first resonant peak in a lightly damped assembly, etc. The basis for choosing n is the required bandwidth (or fast transient response) needed in the studies to be performed using the model. For ultimate fidelity one would have to choose n quite large – or use a PDE representation. This is often unnecessary in many applications, however.

3. Simulation Overview

There are in fact two main considerations regarding the appropriate approach to be taken in simulating a dynamic system on a digital computer: the first is what language to employ in representing the model, and the second is what algorithm(s) to use to solve it. By "solve", in the context of ODEs, we informally mean the following: given an initial condition, x_0, t_0 and an input signal (or vector of signals), $u(t)$ defined for $[t_0\,, t_f]$ where t_f is the desired solution final time, and generate the corresponding solution, which we denote $x(t; x_0, t_0, u(\tau \in [t_0\,, t_f]))$ for $t \in [t_0\,, t_f]$. For DAEs and PDEs the term solve has a directly analogous meaning; hereafter, we will deal only with ODEs.

3.1. Simulation Languages

Most users take advantage of an off-the-shelf modeling and simulation environment, in which case the language is given. Well-known ground-breaking languages for modeling ODEs include the SCi Continuous System Simulation Language (CSSL [14]), the Advanced Continuous Simulation Language (ACSL [6]) and Simnon [7]. In the 1980s and 1990s the number of languages and environments proliferated, including extensions along traditional lines, e.g., MATLAB-based software such as MATRIX_X[15] and more innovative approaches, including object-oriented languages such as OMOLA [16] and DYMOLA [17]. In this same period several competing graphical modeling systems were developed and marketed, most notably MathWorks' SimuLink [8] and Integrated Systems' SystemBuild [9], where one assembles the model using interconnected blocks, picked, placed and connected graphically. Still more recently, at least in many disciplines, MathWorks' products have become dominant, with MATLAB serving as a strong formulaic language (for which one writes the ODE set as a function in an `m-file` and uses one of the numerical integrators such as `ode45`), and SimuLink. The choice of a formulaic or a graphical modeling language is largely a matter of personal preference, although many feel that writing the ODEs directly permits more control and assurance of the outcome.

3.2. Simulation Algorithms

Turning to the question of algorithms, which in terms of quality simulation is **the** most important question, one must consider the type of system being studied. To make the issues clear, we continue to focus entirely on the simulation of ODEs. Several important categories should be noted:

1. *Continuous system models,* $\mathcal{M}_c$ – ideally, the nonlinearities and the input signals in f_c (Eq. 3) should be continuous and differentiable so that the solutions $x(t; x_0, u(\tau \in [t_0\,, t]))$ can be fit accurately with high-order polynomials.
2. *System models with discontinuities,* $\mathcal{M}_d$ – either the nonlinearities or the input signals (or both) may make instantaneous changes in value (examples: a signum function or "ideal relay" nonlinearity, a square-wave input).
3. *System models with multi-valued nonlinearities,* $\mathcal{M}_{mv}$ – the nonlinearities may include effects such as hysteresis, so, for example, the value of a nonlinearity may depend not only on the current value of its input(s) but "where it's been recently".

There are several major families of integration algorithms, and their strengths and limitations make them more or less suitable for the categories of system models defined above.

1. *Continuous system models* – formerly, predictor / corrector methods [18] were considered to be the best choice for systems in $\mathcal{M}_c$. They were derived to provide the best fit of an n^{th}-order polynomial to a number of past solution points, as well as to achieve desired stability properties (stability in the sense of integration errors not growing as more integration steps are taken). For an m^{th} order algorithm the general idea is: given $x_{k-m}, \ldots x_{k-1}, x_k$ and corresponding past derivatives, we first predict to obtain $x_{p,k+1}$, then calculate $\dot{x}_{p,k+1}$, and finally determine the corrected point $x_{c,k+1}$.
2. *Discontinuous system models* – for models in $\mathcal{M}_d$ the best choices are Runge-Kutta methods. These do not use past points and polynomial fitting, rather they "explore" the derivative vector field taking several fractional and whole steps from t_k to t_{k+1}, then they combine the resulting derivatives to generate the final result. For example, a standard 4^{th}-order Runge-Kutta algorithm starts with x_k and $\dot{x}_k$ and then executes three explorations (two with half steps, one with a full step) and then combines the resulting derivatives to obtain the final result [18].

   Note that error estimates and step-size adjustment is available for both modern predictor / corrector methods as well as Runge-Kutta methods.
3. *System models with multi-valued nonlinearities* – for models in $\mathcal{M}_{mv}$ effects such as hysteresis degrade the performance of classical Runge-Kutta methods. The problem is that the output of a hysteretic nonlinearity is unknown unless you know its input's past history. For example, in a relay with hysteresis, Fig. , if the input x enters the region $-\delta < x < +\delta$ from the right then the output $\phi(x) = F$, while if it enters from left then $\phi(x) = -F$. We have introduced the concepts of "state-event handling", "switching surfaces" and "modes" to address this problem.

4. State-event Handling

A state event is defined as an instantaneous change **in the model** when an event happens. We see in Fig. that the nonlinearity $\phi(x)$ switches from $+F$ to $-F$ when the input x passes from $x = -\delta + \epsilon$ to $x = -\delta - \epsilon$. As shown in Fig. , there are in essence two models, one with $\phi(x) = +F$ and the other with $\phi(x) = -F$. We switch from the model in which $\phi(x) = +F$ to the alternative model $\phi(x) = -F$ when the switching function $S = x + \delta$ goes through zero. In our approach [19, 20] our extended integrator supports this by (1) iterating to find the exact value of x for which $S = 0$, (2) changing the mode appropriately (e.g., from $+1$ to -1 for the state event under discussion).

Clearly state-event handling requires a more complex model structure – we must provide the model with the input mode, m, and define the outputs S and $x^+ = r(x, t, m)$, i.e., the state reset if it is required, as it would be if two gears switch from disengaged to engaged and total angular momentum must be preserved.

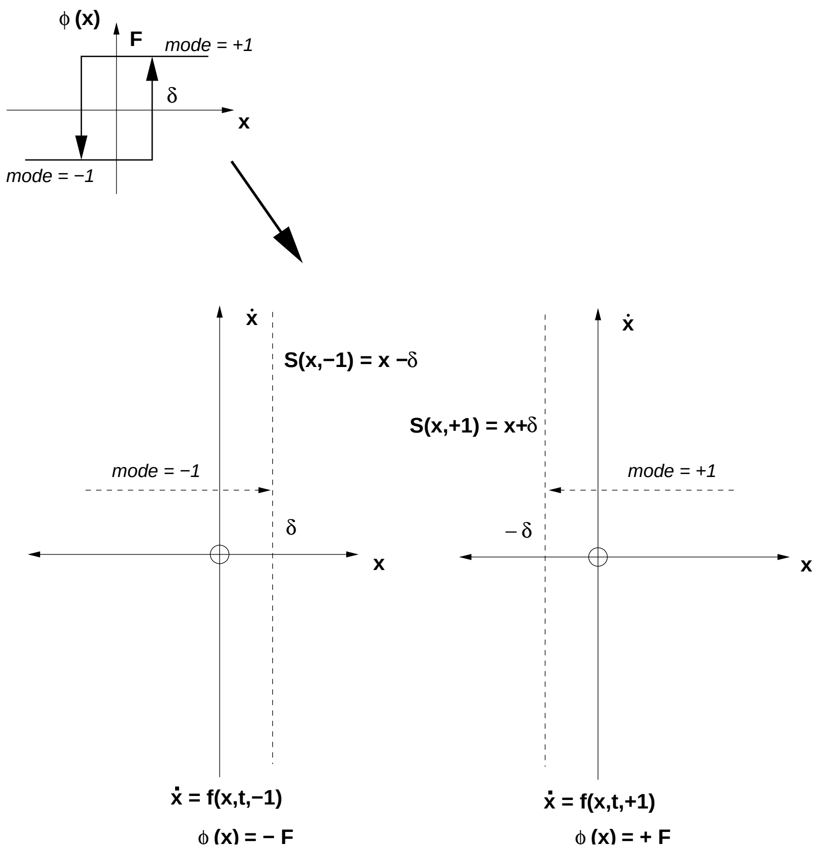


Figure 1: Illustration of switching and modes

This extended schema allows our simulation algorithms to handle state events rigorously. The corresponding software is based on combining a standard numerical integrator such as MATLABs `ode45` with the MATLAB zero-finding algorithm `fzero` (with permission). The routines, along with numerous examples, are available on my web site http://www.ece.unb.ca/jtaylor/ under Simulation Software.

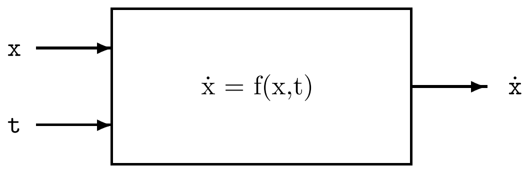
Example Application

The extensions to MATLAB outlined above were implemented and tested using a number of simple switching systems [19, 20]. In addition, we demonstrated the modeling and simulation of a much more realistic (and difficult) application, control of a nonlinear model of an electro-mechanical testbed [21, 20]; the model was composed of two subsystems: a drive subsystem (a DC motor with coulomb friction, a gear train with backlash, and an elastic shaft) and a wheel/barrel subsystem (including an inertial wheel, also with coulomb friction, and a flexible gun barrel). Here, in the interest of brevity and clarity, we will focus on simpler continuous-time dynamics, a model for a missile roll-control system due to Gibson [23].

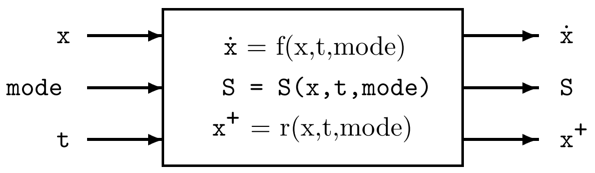
The model depicted in Fig. 3 assumes a pair of reaction jets is mounted on the missile, one to produce torque about the roll axis in the clockwise sense and one in the counterclockwise sense. The force exerted by each jet is $F = 100$ lb and the moment arms are $r = 2$ ft. The moment of inertia about the roll axis is 3.45 lb-ft/sec^2. Let the control jets and associated servo actuator have a hysteresis $h = 5$ lb and two lags corresponding to time constants of 0.01 sec and 0.05 sec. To control the roll motion, there is roll and roll-rate feedback, with gains of 420 lb/radian and 42 lb/(radian/sec) respectively. In the original formulation, the controller was analog; here however we have modeled it as a digital algorithm wherein the rate signal is generated by numerical differentiation; the sampling time is fast, to emulate the analog control with reasonable fidelity.

The hysteretic relay action was modeled rigorously using modes. The relay output corresponds to $F = 100\,m$ where m takes on values of ± 1, and the switching function is

$$S(e,m) = \begin{cases} e + h\ , & m = 1\ ; \\ e - h\ , & m = -1 \end{cases} \tag{11}$$



(a) Standard MATLAB model schema



(b) Extended MATLAB model schema

Figure 2: MATLAB model input/output structure

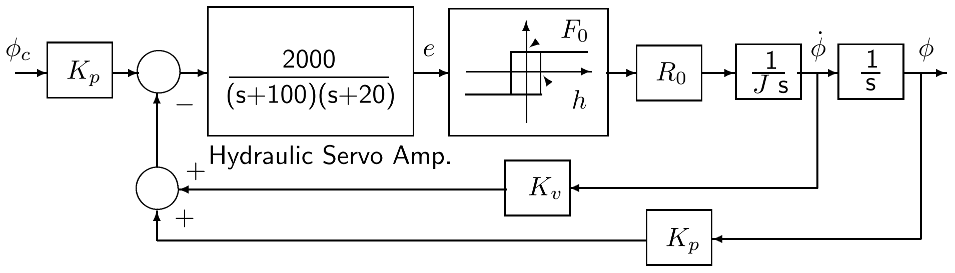


Figure 3: Gibson's roll-control model block diagram

where e is the relay input (output of the hydraulic servo amplifier, Fig. 3).

Analog simulation and describing function analysis predict that this system should exhibit limit cycles [23]. Simulation results for the digital controller are depicted in Fig. 4. According to Gibson, the amplitude and frequency of the limit cycle oscillation should be $A_{LC} = 0.135$ rad, $\omega_{LC} = 22.9$ rad/sec, which agrees quite well with these hybrid simulation results ($A_{LC} = 0.130$ rad, $\omega_{LC} = 23.1$ rad/sec).

6. Conclusion

The discussion of modeling, simulation (languages and algorithms) and state-event handling presented above provides a framework for rigorous M & S of mechatronic systems, where realistic models are inherently nonlinear, often discontinuous (due to friction effects, for example), and likely to contain multi-valued components (such as relays). The importance of careful time- and state-event handling was particularly emphasized. Introducing the concepts of **mode** and **switching function** plus the carefully prescribed "reset" protocol are signicant contributions toward making the modeling and simulation of switching in mechatronic systems more systematic and rigorous. These features permit the study of systems that are difficult for the standard MATLAB integrators such as `ode45`, which usually consume a large amount of processing when a switch occurs, because switching leads to making the step-size extremely small to provide a solution the meets the error criterion. Machinery for the execution of embedded discrete-time components further increased the level of generality available for modeling and simulation hybrid systems - but that software is not robust enough to distribute. The key factor in these developments is the ability to guarantee the correct timing of state- and time-events, so that questions such as "does the discrete-time component execute just before or after the relay switches" can be answered with high reliability.

5. References

[1] Lotka, A. J., *Elements of Physical Biology*, Williams & Wilkins Co., Baltimore, 1925; Volterra, V. Variazioni e fluttuazioni del numero dui in specie animali conviventi. Mem. R. Accad. Naz. dei Lincei. Ser. VI, Vol. 21926; presented in many books, cf. pages 409-448 in Chapman, R. N., *Animal Ecology*, McGraw-Hill, New York, 1931.

[2] Ljung, L., *System Identification - Theory for the User*, Prentice Hall, Saddle River, NJ, 1999 (second edition).

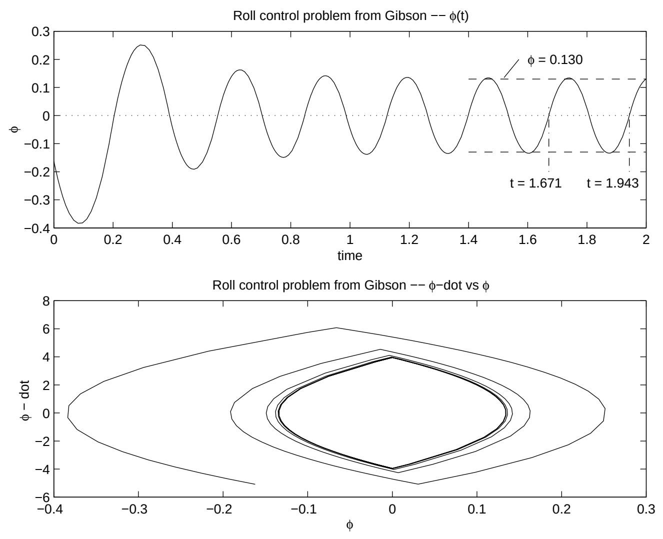


Figure 4: Illustration of State-event Handling Results

[3] Brenan, K. E., Campbell, S. L. and Petzold, L. R., *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North Holland, 1989.

[4] Rabier, P. J. and Rheinboldt, W. C., "A General Existence and Uniqueness Theorem for Implicit Differential Algebraic Equations", *Diff. Int. Eqns.*, Vol. 4, pp. 563-582, 1991.

[5] Potra, F. A. and Rheinboldt, W. C., "Differential-Geometric Techniques for Solving Differential Algebraic Equations", in *Real-Time Integration Methods for Mechanical System Simulation*, Ed. by E. J. Haug and R. C. Deyo, Springer-Verlag Computer & Systems Sciences Series, Vol. 69, pp. 155-191, 1991.

[6] *Advanced Continuous Simulation Language (ACSL), Reference Manual.* Mitchell & Gauthier Associates, Concord, MA 01742.

[7] Elmqvist, H., "SIMNON - An Interactive Simulation Program for Non-Linear Systems", in *Proc. of Simulation '77*, Montreux, France, 1977.

[8] *SimuLink User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[9] *SystemBuild User's Guide*, Integrated Systems, Inc., Santa Clara, CA 95054.

[10] Gear, C. W., "The Simultaneous Numerical Solution of Differential-Algebraic Equations", *IEEE Transactions on Circuit Theory*, Vol. TC-18, pp. 89-95, 1971.

[11] Gear, C. W. and Petzold, L. R., "ODE Methods for the Solution of Differential/Algebraic Systems", *SIAM J. of Numerical Analysis*, Vol. 21, pp. 367-384, 1984.

[12] Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[13] Reddy, J. N., *An Introduction to the Finite Element Method*, McGraw-Hill, New York, 1984.

[14] Augustin, D. C., Strauss, J. C., Fineberg, M. S., Johnson, B. B., Linebarger, R. N., and Sansom, F. J., "The SCi Continuous System Simulation Language (CSSL)", *Simulation*, Vol. 9, No. 6, December 1967.

[15] Walker, R. A., C. Z. Gregory Jr., and S. Shah. (1982). MATRIXx: A Data Analysis, System Identification, Control Design and Simulation Package. *IEEE Control Systems Magazine,* Vol. 2, pp. 30-37.

[16] Mattsson, S. E. and Andersson, M., "The Ideas Behind Omola", *Proc. CACSD'92, IEEE Computer-Aided Control Systems Design Conference,* Napa, CA, pp. 218–224, March 1992.

[17] Elmqvist, H., Cellier, F. E. and Otter, M., "Object-Oriented Modeling of Power-Electronic Circuits Using Dymola", *Proc. CISS'94* (First Joint Conference of International Simulation Societies), Zurich, Switzerland, August 1994.

[18] Pizer, S. M., *Numerical Computing and Mathematical Analysis*, Science Research Associates Inc., Chicago, 1975.

[19] Taylor, J. H. "Rigorous Handling of State Events in MATLAB", *Proc. IEEE Conference on Control Applications,* Albany, NY, pp. 156-161, September 1995.

[20] Taylor, J. H. and Kebede, D. "Rigorous Hybrid Systems Simulation of an Electro-mechanical Pointing System with Discrete-time Control", *Proc. American Control Conference,* Albuquerque, NM, pp. 2786-2789, June 1997.

[21] Taylor, J. H. and Lu ,J. "Robust Nonlinear Control System Synthesis Method for Electro-Mechanical Pointing Systems with Flexible Modes", *J. of Systems Engineering, Vol. 5* (special issue on motion control systems), pp. 192-204, January 1995.

[22] Petzold, L. R., "A Description of DASSL: A Differential/Algebraic System Solver", *Proc. 10th IMACS World Congress*, Montreal, August 1982.

[23] Gibson, J. E. *Nonlinear Automatic Control*, McGraw-Hill Book Co., New York, NY, 1963.