Rigorous Hybrid Systems Simulation with Continuous-time Discontinuities and Discrete-time Agents

James H. Taylor
Department of Electrical and Computer Engineering
University of New Brunswick
Fredericton, NB CANADA E3B 5A3
E-mail: *jtaylor@unb.ca* Web site: *www.ee.unb.ca/jtaylor*

Abstract: Earlier research in the modeling and simulation of hybrid systems led to the development of a general hybrid systems modeling language (HSML) that has been described elsewhere. In more recent work, we have implemented this concept in software. First, the standard MATLAB model framework and integration algorithms were extended to support state-event handling in continuous-time components, including approaches for dealing with discontinuities in the dynamic model, vector-field conflicts, and changing model order and structure. Then further major extensions were made to handle embedded discrete-time components, generally understood to embody any agent implemented in real-time software.

In this paper we overview the algorithmic implementation of the HSML ideas and language constructs for dealing with state events and embedded discrete-time agents in MATLAB. A simple example (a missile roll control system with a hysteretic switching function and a discrete-time controller) is presented to demonstrate the efficacy of these extensions.

Keywords: Hybrid systems; modeling; simulation; numerical integration methods; discontinuity handling; discrete-time agents.

1 Introduction

HSML, as described previously [1, 2, 16], was designed to support a broad definition of a hybrid system, which we may express informally as being an arbitrary interconnection of components that are arbitrary instances of continuous- and discrete-time subsystems. Requirements for HSML particularly focused on rigorous characterization and execution of "events", both discrete- and continuous-time, that cause discontinuous changes in system trajectories and/or the model structure itself. In this respect, there is much commonality between the HSML project and recent developments by Cellier *et al.* in the area of object-oriented modeling [3]; for a detailed view of state events see especially [4, 5].

In conceiving and developing HSML, there was no claim that one cannot rigorously model hybrid systems using certain other, extant languages. For example, ACSL [6] can be used to model and simulate state events in hybrid systems with considerable generality; however many other packages (especially commercially-supported ones) lack even the basic provisions for state-event handling. Also, the high-level features and strict semantics and syntax formulated for HSML facilitate and enforce a higher degree of rigor in hybrid systems modeling, thereby ensuring a greater probability of model correctness. For example, resetting the state after a state event can be done in ACSL, but not cleanly and reliably; here we use a strict protocol that permits state reset with safety. The ideas and algorithmic requirements underlying HSML can be translated into any modeling and simulation environments, assuming that a developer can gain access to the necessary internal "machinery", as demonstrated in this presentation.

This paper focuses on our work to implement a subset of HSML in a working modeling and simulation environment, MATLAB [7]. It also overviews an application involving significant state-event handling in the continuous-time part (to handle hysteresis) as well as dealing with "time events" associated with the execution of discrete-time elements (a conventional controller), thus providing a complete demonstration of the approach. This presentation complements the preliminary discussions of state-event handling in [8, 9], especially with the approach for coordinating the execution of embedded discrete-time algorithms [10].

The remaining parts are as follows: Section 2 outlines the HSML framework for hybrid systems modeling, Section 3 overviews state- and time-event handling, and Section 4 deals with the extensions needed in MATLAB for modeling and simulating hybrid systems. The final sections show the use of the approach on a hybrid system with discontinuities and a discrete-time component and summarize the present status and future directions of the HSML project.

2 HSML **Overview**

HSML is designed to be a rigorous and modular hierarchical scheme for modeling hybrid systems. At the lowest level HSML components are "pure" continuous-time components (CTCs) and discrete-time components (DTCs) [1]. These elements may be assembled into composite components, and then systems. Every component has an *interface* and a *body*; its interface defines the entities that are accessible from and to the outside, and the body describes the dynamic behavior of the component. As described in [1, 2], the `interface` section requires that the primary input/output variables must be typed ('signal', 'real', 'integer', 'boolean' or 'string') and may be constrained as to range, broadly interpreted to be a numerical range (`<range> = (v_min, v_max)`) or a set (e.g., `<range> = {"high", "medium", "low"}` for a string variable). In the `body`, the sections `declarations` and `assignments` exist for specifying internal variables and assigning parameter values, respectively. Beyond these general observations, a component is elaborated for the specific component type (CTC, DTC) by adding distinctive mandatory and optional sections; e.g., `initial`, `dynamics` and `output` for a CTC. (Note that we have not fully implemented all of these features in MATLAB so far; however, they represent important goals to be pursued in future work.)

Here we consider CTCs that may be represented as[1]:

$$\begin{aligned} \dot{x_c} &= f_c(x_c, u_c, u_{d,k}, m, t) \\ y_c &= h_c(x_c, u_c, u_{d,k}, m, t) \end{aligned} \qquad (1)$$

where x_c is the state vector, y_c is the output vector, u_c and $u_{d,k}$ are numeric input signals (continuous- and discrete-time, respectively), m is comprised of a finite alphabet of numeric or symbolic input variables that characterizes the "mode" of the model, and t is the time; in general $u_c, u_{d,k}$ and m are vectors. There are implicit "zero-order holds" operating on the elements of $u_{d,k}$ and m, i.e., these inputs remain constant between those times when they change instantaneously. Of particular importance to the present exposition, the **mode** input m is included to provide means of controlling the model's structure and coordinating its behavior with the numerical integration process in state-event handling, as described below.

[1]The specific class of CTC that can be modeled depends on the simulator's integration methods. So far we have only extended MATLAB's ordinary differential equation solving routine `ode45`, so we cannot handle differential algebraic equations (DAEs). This leads to a significant simplification of formulation and variable types in comparison with [1, 2].

State events are characterized by *zero crossings,*

$$S(x_c, m, t) = 0 \qquad (2)$$

where S is a general expression involving the state, the mode of the CTC, and perhaps time. An arbitrary state change in the CTC can be classified as a `negative-going` event (i.e., one in which S becomes negative), an `on-constraint` event (where S remains equal to zero until another state event occurs), or a `positive-going` event. Note that this framework provides support for models that undergo structural changes (e.g., changes in the definition or number of state variables) [16]; e.g., in the case of mechanical subsystems engaging, the number of states decreases. Finally, we include provision for instantaneous reset of the model state variables at an event:

$$x_c^+ = x_c(t_e^+) = r(x_c(t_e^-), m, t_e^-) \qquad (3)$$

where r is also an arbitrary expression and t_e is the event time. This feature is useful in resetting velocities after engagement to conserve momentum, for example. In accordance with this scheme for state-event definition, we permit elements of m to take on the values –1, 0, +1.

In the present effort, a DTC is a general algorithm which we can characterize in terms of internal variables called "discrete states" and outputs that also change discretely (instantaneously) at each execution:

$$\begin{aligned} x_{d,k}^+(t_k) &= f_d(x_{d,k}, u_{d,k}, m, t_k) \\ y_{d,k}^+(t_k) &= g_d(x_{d,k}^+, u_{d,k}, m, t_k) \end{aligned} \qquad (4)$$

where $x_{d,k}$ is the discrete state vector, k is the index corresponding to the discrete time point t_k at which the state takes on the new value $x_{d,k}^+$, $u_{d,k}$ is the input vector, and $y_{d,k}^+$ is the output. Note that there are implicit "sampling" operators on $u_{d,k}$ if continuous-time variables are involved. The times t_k are usually – but not necessarily – uniformly spaced ($t_k = t_0 + k * T_s$ where T_s is the sampling period); in any case, we assume that the update times can be anticipated. Corresponding to this, we define the vector $t_{e,k}$ which at any time is comprised of the next execution times for every DTC in the model. (In future extensions provision will be made for computational delay δ_k between the sample time and the output change; this may be modeled with varying degrees of realism, from a fixed delay time to an actual emulation of the computational burden required in handling the computations. Also, note that while the above appears in a form suggesting a discrete-time controller or Kalman filter algorithm, we emphisize that we have other more general "logic-based" component formulations in mind [1, 2, 16].)

3 Event Handling

3.1 State-Events

The HSML features for modeling state events are designed to permit the accurate and efficient integration of CTCs that may exhibit discontinuous behavior such as relays switching and mechanical components engaging/disengaging. The nature of the problem and an approach for proper handling of such events has been detailed previously[8, 9]; in this context, it suffices to observe that blindly integrating a CTC by stepping from a point t_k before switching to t_{k+1} after the discontinuity usually produces results that are both inaccurate and inefficient (in the sense of consuming an inordinate amount of computation).

The appropriate handling of state events requires coordination between the model and simulation package. This is achieved in HSML via `flag` variables in the model (S in Eqn. 2; these signal the integrator that a state event has been overshot), and the model input variable m that can be used to control model switching. State-event handling then proceeds as follows:

1. Integrate as usual as long as the `flag` variables do not change sign. Each integration point is treated as a "trial" point until the sign condition is checked; if no sign change has occurred, the point becomes "accepted".
2. When a sign change is detected, the trial point is discarded and an iterative procedure is initiated (within the simulator) to find the step h^* such that the `flag` variable is zero (within a small tolerance ϵ "on the other side"). The model does not switch during this part of the procedure.
3. The integrator produces an accepted point just past the switching curve (Eqn. 2) and then signals the model to switch (e.g., by changing m from 1 to -1 or *vice versa* if the boundary is to be crossed, or to 0 if the trajectory is to be confined to the boundary until the next state event).
4. The integrator then calls the model to determine if a state reset is desired, and if so executes it.
5. Normal integration proceeds from that point until the next state event is encountered.

This procedure is illustrated below using MATLAB extensions.

3.2 Time-Events

The approach and conventions needed to handle time events are much simpler than those required for state events. After all, we are merely emulating the execution of a computer algorithm in a digital setting (but without actual real time considerations). As we are implementing this feature, we assume that each DTC will "notify" the higher-level system integration block (SIB) about the next execution time, `t_exec` ($t_{e,k}$). This is done at the beginning of the simulation and at every subsequent DTC execution. The SIB determines the earliest of the anticipated time events (if there is more than one DTC), and signals the numerical integrator to stop at that time. At that point the SIB is invoked with $t = t_{e,k}$ and it proceeds to execute the appropriate DTC(s), handling priority issues as required. At each DTC execution this process is updated and continued until the end of the simulation run.

4 MATLAB Extensions for Event Handling

The above outline of HSML and it's approach for characterizing state and time events provides a clear set of requirements for implementation in a software package. We have focused on MATLAB for this purpose, since it is so readily extensible. Generalizations are needed in two major areas: modeling schemes and numerical integration methods.

4.1 Extended Model Schema

The model input/output framework had to be extended as follows [8, 9]: The original MATLAB scheme was to create models in the form of functions with two inputs (t, x) and one output ($\dot{x}$). To this, we added the new input variable m (mode), allowing the numerical integration routine to request that the model switch according to state events as they are detected. In addition, two new output variables are S, the `flag` variable in Eqn. 2 that signals a state event, and r, included to permit state reset (Eqn. 3). Note that S and m may be vectors, to support multiple state event mechanisms (switching conditions).

The inclusion of embedded DTCs in hybrid system models necessitates a further increase in complexity in comparison with these earlier extensions. We recently adopted a modular model-building scheme much closer to the conceptual design of HSML (and reminiscent of SIMNON [11]). This scheme is portrayed in Fig. 1 (at the end of the paper). In this diagram, observe that:

- The Numerical Integrator (NI – see [8, 9]) must now serve as the "memory" for the *aggregate discrete-component states* (x_d) and for the DTC*'s times of* *next execution* t_e, a vector having dimension equal to the number of DTCs. The NI has the new requirement of stopping exactly at t_n, the earliest of the elements of t_e, and the "System Integrator Block" (SIB) has the responsibility of executing the correct DTC(s) when $t = t_n$.

- If multiple DTCs are to be executed at t_n, then the SIB has the responsibility of calling them in the correct order.
- The continuous-time dynamics can reside in the SIB if they are simple; if it is helpful to create one or several separate CTCs, then one can do this as diagrammed; note that this necessitates CTCs having inputs and outputs that are defined at the interface, as shown.

4.2 Extended MATLAB Integrators

Significant extensions must also be made in the MATLAB numerical integration algorithms. There are three features needed to permit the MATLAB integration routines to deal with state and time events:

1. the numerical integrator must coordinate with the extended model to establish the initial values of m and t_e;
2. the routine must continuously test for the occurrence of events by:
   (a) ensuring that t stops at t_n for a time event, and/or
   (b) watching for zero crossings in S, iterating exactly to the switching point and then changing m; and
3. it must execute a state reset operation after a state event, if it is called for by the model.

To support this functionality, the following conventions are imposed: The value of m for initialization is "empty" ($m = [\ ]$). The model must return the appropriate value of S, based on the stipulated initial condition x_0. From this information, the integration routine will set $m = \text{sign}(S)$. During normal integration the value of m's elements will be $-1, 0, 1$. When a state event is detected and determined[2], the corresponding element of m is switched; then m is temporarily made complex and the model should respond by returning the reset value $\mathtt{r}$ (Eqn. 3) or $r = [\ ]$ if no reset is to be done. Finally, that element of m is returned to $-1, 0, 1$ and numerical integration is resumed with the indicated mode change.

5 Example Application

The extensions to MATLAB outlined above were implemented and tested using a number of simple switching systems [8, 9]. 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 [12, 13]; 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 [14].

The model depicted in Fig. 2 (at the end of the paper) 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

$$S(e,m) = \begin{cases} e + h\ , & m = 1\ ; \\ e - h\ , & m = -1 \end{cases} \tag{5}$$

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

Analog simulation and describing function analysis predict that the system should exhibit limit cycles [14]. Simulation results for the digital controller are depicted in Fig. 3. 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 MATLAB implementation presented above provides a demonstration of HSML in general and of the importance of careful time- and state-event handling in particular. Introducing the concept "mode" and the carefully prescribed "reset" protocol are both contributions toward making the modeling and simulation of switching

[2]We determine zero crossings by embedding a modified version of MATLAB's `fzero` algorithm within the integrator [9].

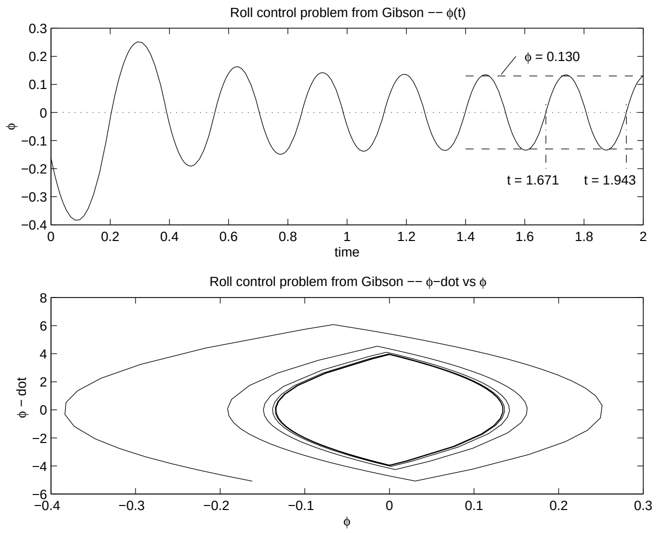

Figure 3: Illustration of Hybrid Simulation Results

in continuous-time systems more systematic and rigorous. These features permit the study of systems that are well beyond the capabilities of the standard MATLAB integrators such as `ode45`. Machinery for the execution of embedded discrete-time components further increases the level of generality available for modeling and simulation of hybrid systems. A 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 DTC execute just before or after the relay switches" can be answered with high reliability.

Extending this modeling approach and associated numerical integration routines can be pursued in several obvious ways, e.g., they can be inserted into more sophisticated modeling environments (like the SIMULINK framework [15]). A more important extension would involve the development of a "HSML compiler", that would take the more rigorous HSML formulations and autocode extended MATLAB models. (See Section 2, e.g. typing and constraining input/output variables.) In a more technical vein, handling vector-field conflicts (situations where the solution must remain on the surface $S = 0$ because the ODE vector is "into" the surface on both sides) is not completely treated in the above approach. In short, substantial work remains to be done!

References

[1] Taylor, J. H. "Toward a Modeling Language Standard for Hybrid Dynamical Systems", *Proc. 32nd* IEEE *Conference on Decision and Control,* San Antonio, TX, December 1993.

[2] Taylor, J. H. "A Modeling Language for Hybrid Systems", *Proc.* IEEE/IFAC *Symposium on Computer-Aided Control System Design*, Tucson, AZ, March 1994.

[3] 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.

[4] Cellier, F. E., Elmqvist, H., Otter, M. and Taylor, J. H., "Guidelines for Modeling and Simulation of Hybrid Systems", *Proc.* IFAC *World Congress*, Sydney Australia, 18–23 July 1993.

[5] Cellier, F. E., Otter, M. and Elmqvist, H., "Bond Graph Modeling of Variable Structure Systems", *Proc. ICBGM'95* (Second International Conference on Bond Graph Modeling and Simulation), Las Vegas, Nevada, January 1995.

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

[7] MATLAB *User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[8] Taylor, J. H., "Rigorous Handling of State Events in MATLAB", *Proc.* IEEE *Conference on Control Applications*, Albany, NY, 28–29 September 1995.

[9] Taylor, J. H. and Kebede, D., "Modeling and Simulation of Hybrid Systems", *Proc.* IEEE *Conference on Decision and Control*, New Orleans, LA, 13–15 December 1995.

[10] J. H. Taylor and D. Kebede, "Modeling and Simulation of Hybrid Systems in MATLAB", *Proc. IFAC World Congress Vol. J*, San Francisco, CA, pp. 275-280, July 1996.

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

[12] J. H. Taylor and J. Lu, "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.

[13] J. H. Taylor and D. Kebede, "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.

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

[15] SIMULINK *User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[16] Taylor, J. H. *A Rigorous Modeling and Simulation Package for Hybrid Systems*, US National Science Foundation SBIR Report, Award No. III-9361232, Odyssey Research Associates, Inc., June 1994 (available only from the author).

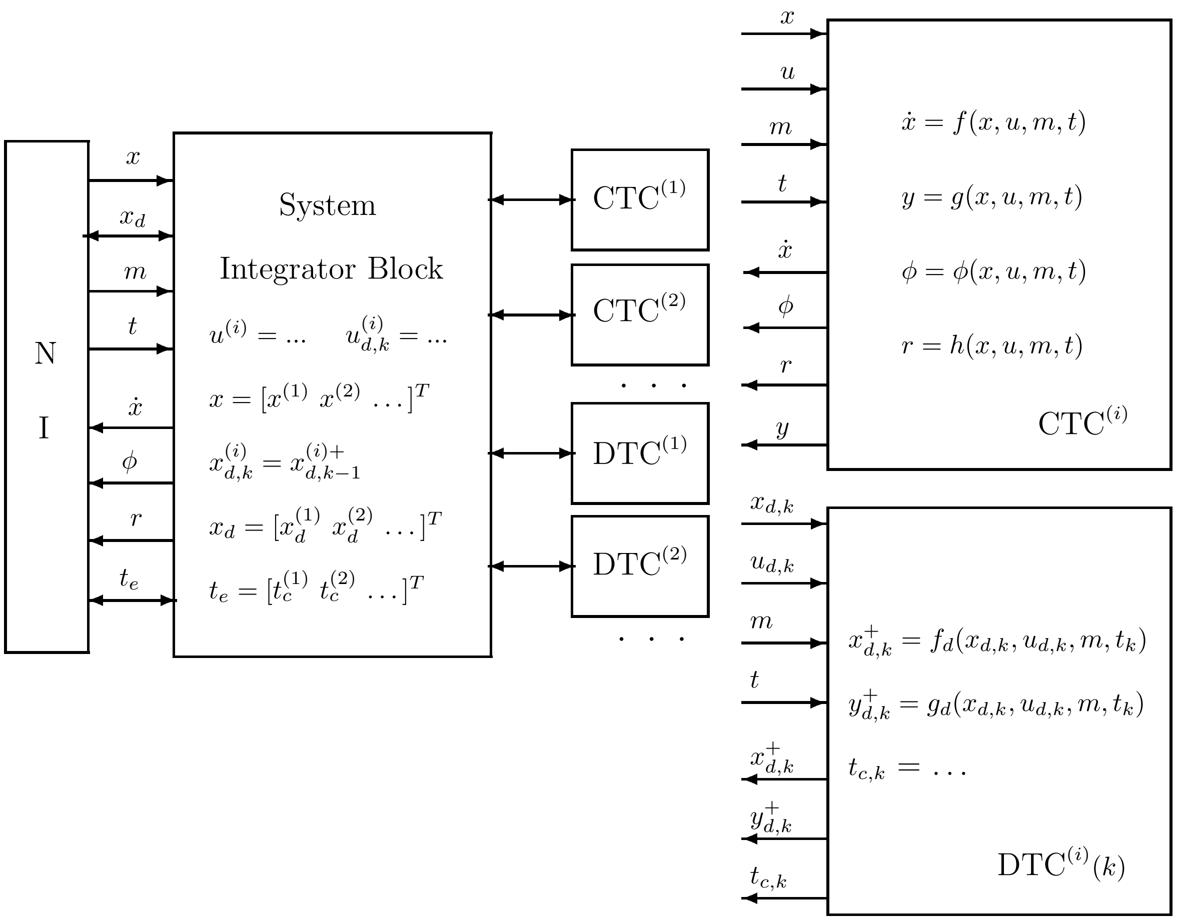

Figure 1: New MATLAB model component input/output structures

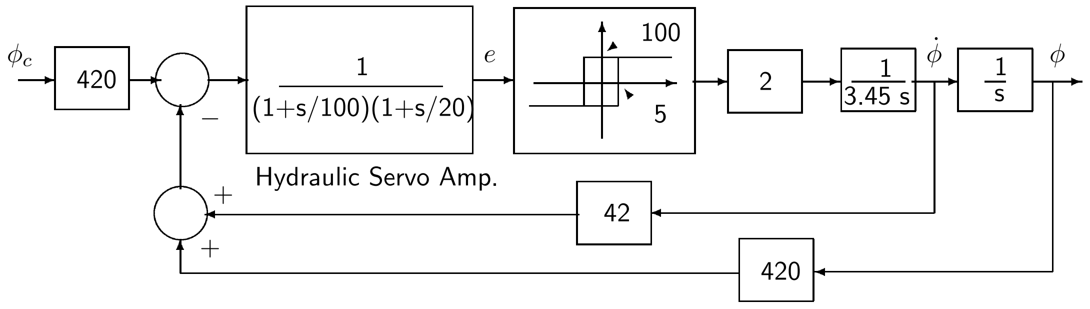

Figure 2: Missile Roll Control System Schematic