Implementation of Quantized State Systems in MATLAB/Simulink

Patrick Grabher[1], Matthias Rößler[2*], Bernhard Heinzl[3]

[1]Inst. of Analysis and Scientific Computing, Vienna University of Technology, Wiedner Haupstraße 8-10, 1040 Vienna, Austria; * *matthias.roessler@tuwien.ac.at*

[2]dwh Simulation Services, Neustiftg. 57-59, A-1070 Wien, Austria

[3]Inst. of Computer Aided Automation, Vienna University of Technology, Treitlstraße 3, 1040 Vienna, Austria.

Simulation Notes Europe SNE 24(3-4), 2014, 185 - 190
DOI: 10.11128/sne.24.tn.10269
Received: June 12, 2014; Revised September 25, 2014;
Accepted: October 25, 2014;

Abstract. Common practice in the simulation of continuous systems is to discretize the time in order to obtain a numerical solution. The Quantized State System (QSS) approach makes it possible that the discretisation is applied to the state variables, instead of the time range. In other words continuous systems can be simulated event-based with the QSS method. It also effects a new orientation and leads among other things to very efficient state event detection. Ernesto Kofman and Sergio Junco presented the QSS method in the DEVS formalism [1]. This paper describes how the implementation of the QSS method in Simulink/SimEvents works and which restrictions still exist.

1 Introduction

The content of this chapter is taken from [1]. Considering a model which is described by the equation

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), \tag{1}$$

the associated QSS system is

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{q}(t), \boldsymbol{u}(t)\big). \tag{2}$$

The variable $\boldsymbol{q}(t)$ is the quantized state of $\boldsymbol{x}(t)$ and will be obtained by a quantized hysteretic function and is defined by:

Let $Q = \{Q_0, Q_1, Q_2, \dots, Q_r\}$ be a set of real numbers and $Q_{i-1} \le Q_i$ for $1 \le i \le r$. Let $x \in \Omega$ be a continues trajectory, where $x: \mathbb{R} \to \mathbb{R}$. Let $b: \Omega \to \Omega$ be a mapping and $q = b(x)$ the trajectory which satisfies

$$q(t) = \begin{cases} Q_m, & if\ t = t_0 \\ Q_{i+1}, if\ x(t) = Q_{i+1} \wedge \mathrm{q}(\mathrm{t}^-) = \mathrm{Q_i} \wedge \mathrm{i} < r \\ Q_{i-1}, if\ x(t) = Q_i - \varepsilon \wedge \mathrm{q}(\mathrm{t}^-) = \mathrm{Q_i} \wedge \mathrm{i} > 0 \\ q(t^-), & else \end{cases} \tag{3}$$

and

$$m = \begin{cases} 0, & if\ x(t_0) < Q_0 \\ r, & if\ x(t_0) \ge Q_r \\ j, & if\ Q_j \le x(t_0) < Q_{j+1} \end{cases}, \tag{4}$$

then b is the *Quantized Function with Hysteresis.* ε is called hysteresis width und describes a delay for some events. Generally you use a uniform quantum ΔQ like in Figure 1.

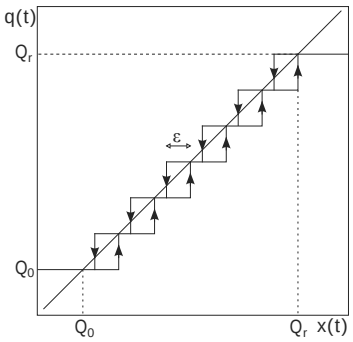

Figure 1: Quantized Function with Hysteresis.

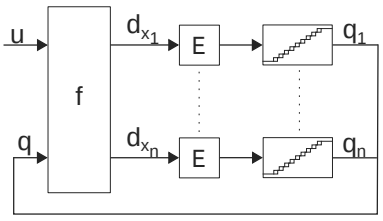

Figure 2: Block diagram of a Quantized State System, which consists of the static function f and the quantized intergrator with hysteresis.

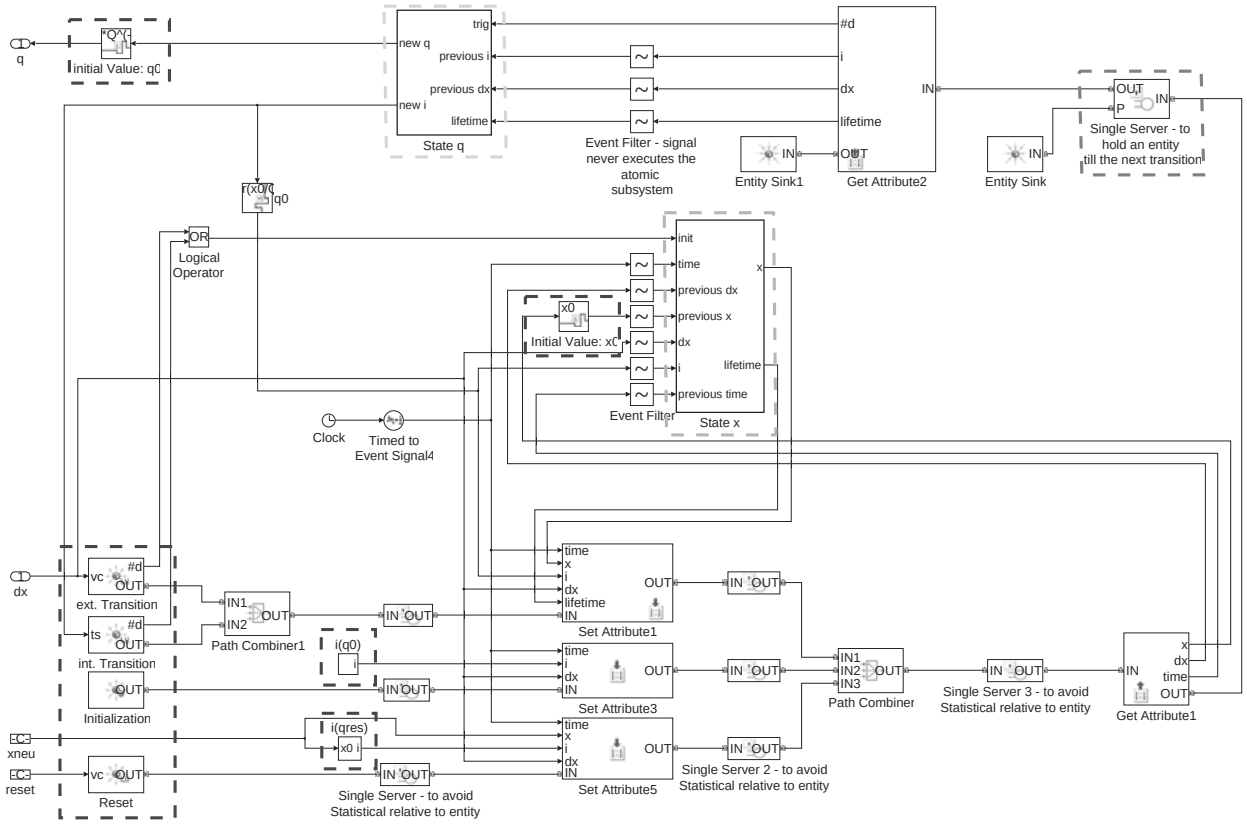

Figure 3: Implementation of the Quantized Integrator of QSS method in Simulink.

Figure 2 shows the block diagram of a QSS. Since it is a first order method, the signals $\boldsymbol{q}$ and $\boldsymbol{d_x}$ are always piecewise constant for every static function f (note that f is time-invariant). So it is sufficient to implement only the quantized integrator to obtain a legitimate QSS system.

1 Implementation of the QSS Method in Simulink

A large part of the implementation was built with elements from the SimEvents library. The general functionality of this integrator is: An event, like a change in the state derivative or in the quantized state, generates an entity. This new entity replaces the last entity. In this moment it describes the new state and specifies when the quantized state has to change.

1.1 Events and Entity Generators

Considering a quantized integrator, computations have to be performed only in moments of events. In the QSS method there are two types of events. The first type is the input event. It describes any kind of change in the state derivative. The other type is the output event, it is a change of the quantized state. In figure 3 one can see on the left side the quantized state q and the state derivative d_x as inputs for the QSS integrator.

Generally each event in the QSS system generates an entity in at least one integrator. Additional entities are generated in the beginning of the simulation or if the integrator is reset. The single purpose of these kinds of entities is to initialize the system with the according initial values, so the lifetime of these entities are zero seconds. For each case there exists an entity generator (see the bottom left corner in figure 3). In some models an input event occurs simultaneously with an output event. For this case the event priority in the entity generator for the input events is set to 1000 in contrast to 5, so that input events are replaced. It should be mentioned, that it does not matter which event is preferred because both entities get the same attributes.

1.2 Attributes and computations

The generated entities also get some attributes to describe the exact state of the integrator. Let S_n be an entity (with sequencing entity S_{n+1}). Its related attributes *time of arising* t_n, *exact state* x_n, *index* $i_n \in \mathbb{Z}$ of the quantized state $q_n = i_n \cdot \Delta Q$, *state derivative* $d_{x,n}$ and the *lifetime* σ_n are computed and stored in the moment of the creation of S_n. The initial values x_0, i_0 and q_0 (see the blue outlined blocks in Figure 3) are given by

$$\begin{aligned} x_0 &= x(0), \\ i_0 &= \left\lfloor \frac{x(0)}{\Delta Q} \right\rfloor, \\ q_0 &= i_0 \cdot \Delta Q. \end{aligned} \tag{5}$$

The single purpose of the usage i_n, which is an Int32 data type, is that simple rounding errors can be avoided in q_n. The state x_n (see the green outlined block in Figure 3) is obtained by

$$x_n = \begin{cases} x_0, \ if\ n = 0 \\ x_{n-1} + d_{x,n-1} \cdot (t_n - t_{n-1}), \ else \end{cases}. \tag{6}$$

How long this integrator holds his state q_n depends on σ_n, which is given by

$$\sigma_n = \begin{cases} 0, \ if\ n = 0 \\ \frac{(q_{n-1}+\Delta Q)-x_n}{d_{x,n}}, \ if\ n > 0 \ \wedge d_{x,n} > 0 \\ \frac{x_n-(q_{n-1}-\varepsilon)}{|d_{x,n}|}, \ if\ n > 0 \ \wedge d_{x,n} < 0 \\ \infty, \ if\ n > 0 \ \wedge d_{x,n} = 0 \end{cases}. \tag{7}$$

After this attributes were assigned to S_n, the entity reaches the entity (single) server (see the red outlined block in Figure 3). It stays for σ_n, if meanwhile no other entity replaces it.

If S_n reaches σ_n in this entity server and if $\sigma_n \neq 0$ (i.e no initialization) it generates an output event and the quantized state changes its values by

$$q_{n+1} = \begin{cases} q_n + \Delta Q, \ if\ d_{x,n} > 0 \\ q_n - \Delta Q, \ if\ d_{x,n} < 0 \end{cases}. \tag{8}$$

In the other cases q_{n+1} stays unchanged with $q_{n+1} = q_n$ (see the yellow outlined block in Figure 3).

1.3 Interaction between event and time-based blocks

The compatibility between different blocks in Simulink is an important issue. In principle, the integrator is built with event-based blocks, but for the attributes q_n and x_n computations are necessary. Therefore in Simulink exist the possibilities to use the *Attribute Function* (it is an embedded MATLAB-function), the *Gateway-blocks* (does not work for every signal and produces sometimes algebraic loops) or time-based math operations, which need to be in an *Atomic Subsystem.*

In the end the third possibility is the best in view of reliability and efficiency at the moment. Figure 3 also shows, that all atomic subsystems are executed by changes of only one signal, on that account all other input signals use an event filter.

1.4 Comments

In Figure 3 you can see some additional entity servers, that haven't been described in the previous chapter. They do not cause any delay for the entities. These blocks ensure that each entity receives its corresponding attributes. If they would not exist, the entities could receive attributes, which belong to the last entity.

The integrator allows some types of signals and changes to be reset. You can find these properties in the entity generator which is responsible for the reset (see Figure 3 in the bottom left corner).

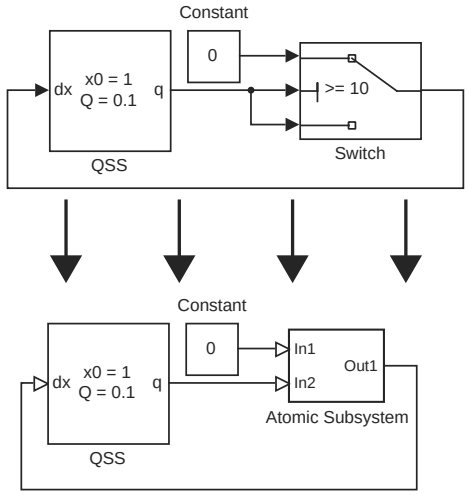

Figure 4: Incompatible time-based blocks have to be placed inside an Atomic Subsystem.

If you create a model sometimes time-based blocks are not compatible with the integrator, because they don't work with the event-based signal type which is coming from the integrator. In this case you have to put the blocks which provoke an error into an atomic subsystem (like Figure 4).

2 Case Studies and Results

In this chapter the properties of the QSS method will be presented. First it should be mentioned, that the solution of QSS method usually doesn't converge in the equilibrium point and so the state is finally oscillating. Also the behaviour with stiff equations is different. We know from time discrete systems that their solution diverges from the analytical solution if they are not stable.

This could not happen with the QSS method, because the error is always bounded with the quantum, although it is fully explicit. But how does the rate of stiff systems affect the solution of the QSS method? The answer is, the more stiff the equation is, the faster is the oscillation of the solution.

2.1 Simple equation

Let following initial value problem be given:

$$\begin{aligned} \dot{x}(t) &= -100 \cdot (x(t) - 0.1), \\ x(0) &= 1\,. \end{aligned} \tag{9}$$

Already this example shows that it is sufficient to take a small step size h for an explicit solver like the Forward Euler (FE) method. Figure 5 shows what happens, if h is chosen too large with $h = 0.025$, namely the solution of the FE diverges from the analytic solution.

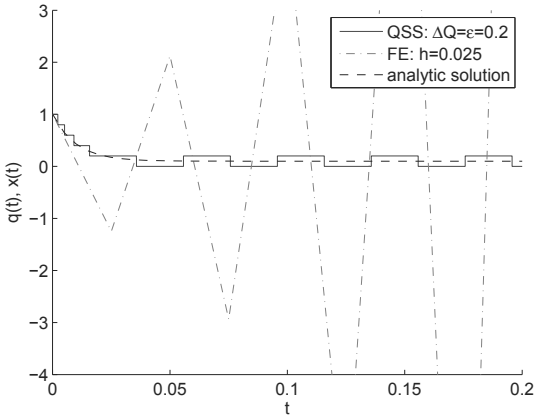

Figure 5: Numerical solutions from the different types of explicit solvers.

This kind of problem does not appear with the QSS method. Usually the quantized state is not able to attain the equilibrium point, so the solution oscillates around it, but it is always bounded with the quantum ΔQ. Figure 5 shows this behavior of the QSS method with $\Delta Q = 0.2$.

2.2 Stiff equation

Following equation represents a stiff problem:

$$\begin{aligned} \dot{x}_1(t) &= x_2(t), \\ \dot{x}_2(t) &= -1000 \cdot (x_1(t) + x_2(t)) + 9500, \\ x_1(0) &= 0, \qquad x_2(0) = 10. \end{aligned} \tag{10}$$

To solve this system a quantum $\Delta Q = 1$ is chosen. Figure 6 shows the behaviour of the QSS method, which is overwhelmed with this situation. The second state variable has 2500 changes in this simulation.

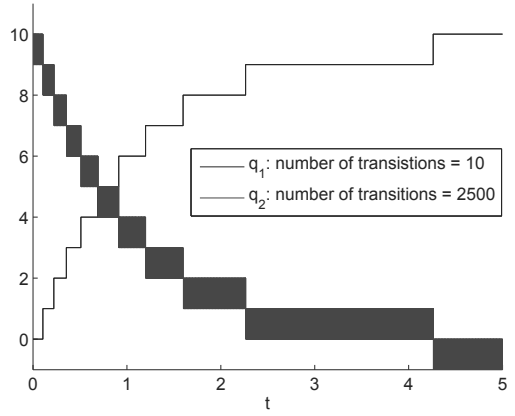

Figure 6: Stiff equations effect a fast oscillation in the state.

The conclusion is that the QSS method is always stable, but requires potentially many computations (see IQSS [2], if you are interested to avoid this problem).

Let us now consider other properties of the QSS method. The event-based system has another two advantages over common time discrete solvers. Firstly, events can be directly detected by the QSS system, so especially for models with a large number of events this method is more efficient than time discrete solvers. The second point is the asynchronous procedure, which is not possible for time discrete solvers. This could be very interesting in large models. The following examples show these capabilities.

2.3 H-Bridge

This model comes from drive engineering and is used to control a DC motor with frequently toggled (with a PWM signal) voltage supply and shows the advantages caused by the asynchronous procedure of the QSS method. Basically this model [3] is described by an electrical part

$$u_S = u_{EMF} + u_R + u_L, \tag{11}$$

and a mechanical part

$$I \cdot \ddot{\omega} = M - k \cdot \dot{\omega}. \tag{12}$$

In the first equation u_S describes the DC *voltage supply*, which is controlled by a PWMsignal, u_{EMF} the *voltage of the electromotive force*, u_R the *voltage at the resistance* and u_L the *voltage of the inductance*. In the mechanical part (12) I is the *moment of inertia* of the rotor, M the *motor torque*, k the *motor viscous friction constant* and ω stands for the *angle* ($\dot{\omega}$ *angular velocity*, $\ddot{\omega}$ *angular acceleration*).

A PWM signal with a frequency of 1kHz controls the H-Bridge for the acceleration and braking procedure. The great advantage of the QSS is that the most computations, which depends on the PWMsignal, must be made in the electronic equation (11) only, whereas the mechanical part (12) depends on a change in the electric current i (input event) or $\dot{\omega}$ (output event). In contrast a time discrete solver, which works synchronously, has to update the whole system if the PWMsignal produces an event.

Figure 7 shows a simulation, where the motor is accelerated the first 8 seconds, then it runs free for 4 seconds (no PWM signal) and at last it brakes.

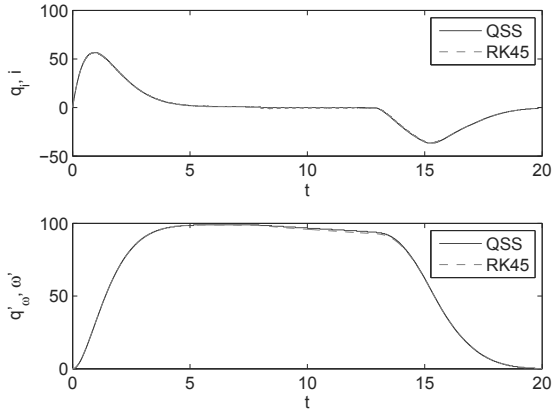

Figure 7: Simulation results of the DC motor.

For the quantum ΔQ the QSS method has 394 output events at the first integrator ($\frac{d}{dx} i \to i$) and 397 at the second ($\ddot{\omega} \to \dot{\omega}$), i.e. that the second one has overall only 791 events (output and input events). In contrast the RK45 uses 29082 grid points for this simulation. Although the motor was running free in the interval (8,12) (i.e. no PWM and $i = 0$ A) the RK45 solver has unnecessary many computations, caused by missing the event $i = 0$ A, see Figure 8. The QSS method does not have this problem and manages the running free operation perfectly with only 7 events in the interval (8,12).

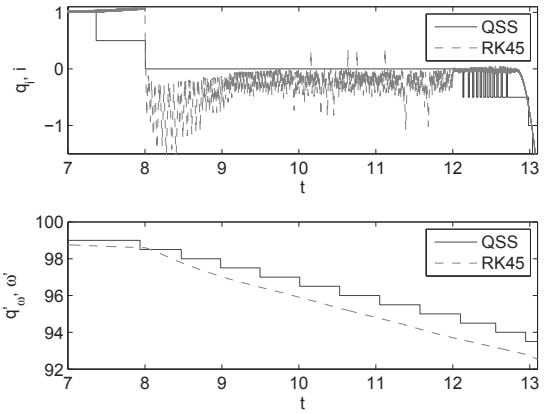

Figure 8: Simulation results of the DC motor in the interval (7,13.1).

2.4 Bouncing ball

Let us now consider a model of a bouncing ball on the floor [4]. The attention here should be pointed at the events when the ball touches the ground and especially what happens if the ball's *vertical speed* $v(t)$ goes to 0 m/s. The fall condition is described by

$$\begin{aligned} \dot{v}(t) &= -g, \\ \dot{x}(t) &= v(t). \end{aligned} \tag{13}$$

In (13) g describes the *earth gravity constant*. If the height $x(t) = 0$ the ball touches the ground and v has to be reset with $v(t) = k \cdot v_{old}$, with the *coefficient of restitution* $k \epsilon (-1,0)$. Normally (in time discrete systems) also the height $x(t)$ will be reset in the moment of an event, which won't be necessary here.

Figure 9 shows three numerical solutions. One can see that is important which quantum is chosen.

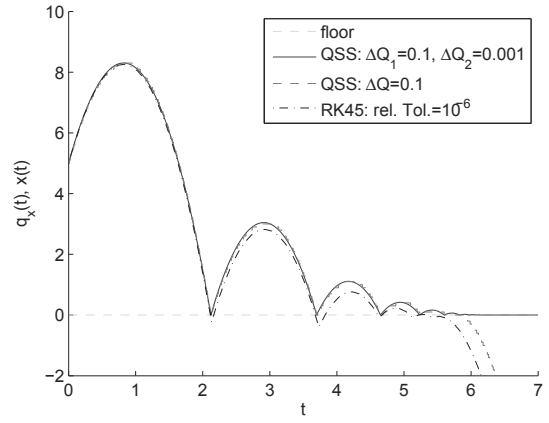

Figure 9: Simulation results of the Bouncing Ball.

Although the QSS method recognizes every event, some parameters effect that the ball does not come up from the floor at a definite time so that it has only the fall condition from this moment. Figure 10 shows the moment of the QSS ($\Delta Q = 0.1$), where the ball does not come up from the floor any more.

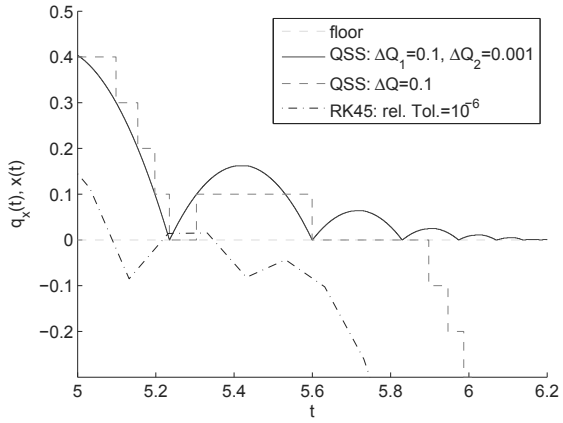

Figure 10: Simulation results of the Bouncing Ball in the interval (5,6.2).

Nevertheless with the QSS method it is also possible to reach the rolling condition. Figure 11 shows that with well chosen parameters ($\Delta Q_1 = 0.1, \Delta Q_2 = 0.001$) a periodic oscillation on the floor can be obtained.

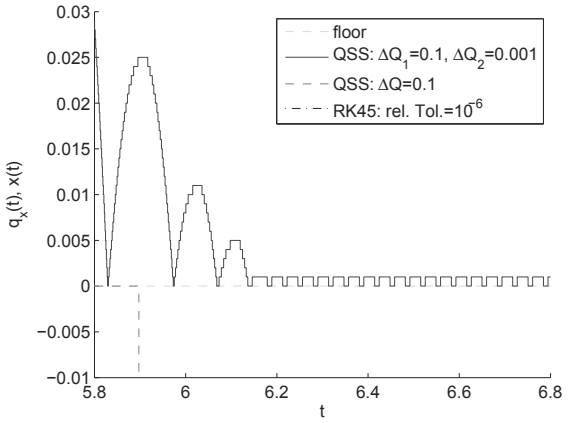

Figure 11: Simulation results of the Bouncing Ball in the interval (5.8,6.8).

Such a rolling condition, which is computed with the QSS method, is inconceivable for time discrete solvers unless they obtain help from additional approaches, like zero crossing detection or a reset of the height $x(t)$ in the moment of every event to handle this *Zeno Phenomenon*.

References

[1] Kofman E, Junco S. Quantized-State Systems: A DEVs Approach for Continuous System Simulation. *Transactions of the Society for Computer Simulation International - Recent advances in DEVS Methodology--part I*. 2001; 18(3): 123-132. ISSN: 0740-6797.

[2] Cellier FE, Kofman E. *Continuous System Simulation*. New York: Springer; 2006. 643 p. ISBN: 978-0-387-26102-7.

[3] *Control Tutorials for MATLAB and Simulink.* University of Michigan. Bill Messner, Dawn Tilbury. cited 2014 Sep 26. Available from: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SimulinkModeling

[4] Heinzl B, Rößler M, Körner A, Zauner G, Ecker H, Breitenecker F. BCP - A Benchmark for Teaching Structural Dynamical Systems. In Breitenecker F, Troch I, editors. Mathematical Modelling. *MATHMOD 2012 - 7th Vienna Conference on Mathematical Modelling;* 2012 Feb; Vienna University of Technology. Zürich: International Federation of Automatic Control. 896-901. ISBN: 978-3-902823-23-6.