Model based control design – a free tool-chain

Gernot Grabmair[1], Simon Mayr[2], Martin Hochwallner and Markus Aigner[3]

***Abstract*— In this study we present a new free tool-chain for model based control design for mechatronic plants applicable to small embedded systems based among other software on the open simulator Scilab-XCos. After a very short introduction of model based design terms this article focuses on the code generator and the other programs of the tool-chain. The design concept is demonstrated by an adaptive self tuning control (STC) of the cart and pendulum system in gantry crane configuration in simulation and on a real laboratory experiment.**

Index terms: open source, code generation, Scilab-XCos, model based control, parameter identification, embedded systems

I. Introduction

Model-based design (MBD) is a mathematical and visual method of addressing problems associated with designing complex control, signal processing and communication systems. It is used in many motion control, industrial equipment, aerospace, and automotive applications. Model-based design is a methodology applied in designing embedded software. During the past years there is a growing interest of more and more medium to small size engineering companies in order to cut down development time and costs. Common tool-chains are quite expensive commercial solutions due to the origin of MBD in aerospace and automotive industries.

Commercial code generators for Matlab-Simulink (M&S) – one of the most complete tool-chains in MBD –, Dymola, etc. do exist. On the other hand, INRIA and others provide free code generators for the outdated Scilab-Scicos – an open source pendant of M&S, e.g., [2], [3], and some more. Scilab-XCos made a major development step concerning the user interface in the last two years but unfortunately the former free code generators do not work anymore. To the best knowledge of the authors there is only one commercial implementation for the new Scilab-XCos suitable for embedded systems.

The main idea presented in this paper is the MBD control development for mechatronic plants with a complete free (or low-cost if target hardware is included) tool-chain from the modeling and control design to the hardware realization using an integrated development environment (IDE). Possible fields of application for such a low-cost development tool-chain are teaching courses and companies interested in testing this new technology or dealing with MBD projects of moderate complexity.

[1]Gernot Grabmair is with the Upper Austrian University of Applied Sciences as Professor for Electrical and Control engineering, Wels, Austria, gernot.grabmair at fh-wels.at

[2]Simon Mayr is with the Upper Austrian University of Applied Sciences, member of the Control engineering group, Wels, Austria, simon.mayr at fh-wels.at

[3]are with the LCM GmbH, Austria, Martin.Hochwallner at lcm.at

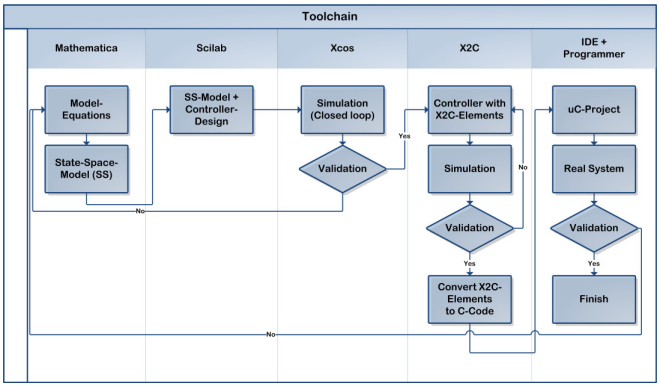

Fig. 1. MBD and the tool-chain.

This paper is organized as follows. Starting with a short description of the parts of the whole tool-chain we focus on the details of the code-generator itself. Afterwards, an adaptive control law for the cart-and-pendulum system is derived as a non-trivial application. Finally, the model based design process is demonstrated by the implementation of this controller on a low-cost embedded system board.

II. MBD – the tool-chain

For the major development steps of a model-based controller design (plant modeling and system identification (1), control design (2) and simulation (3), code generation (4) and transfer to the target (5)) the reader is referred to Fig. 1.

(1) An ideal tool-chain allows to model a mechatronic plant from an engineering point of view, i.e., the plant model can be directly built from physical blocks (e.g., mass, spring, damper, resistor, capacitor, ...). This model can be simulated and additionally, the system equations (e.g., ODEs) are available in symbolic form. This makes symbolic analysis and symbolic control design possible. The physical parameters of the real plant are determined by system identification with real plant measurements and verified by simulation experiments. (2) A controller structure is chosen based on the type of plant, performance requirements, etc., and the controller parameters are calculated with the help of the identified plant parameters. Often the used plant model is somehow idealized in order to obtain smaller control structures. (3) The fulfillment of the control performance requirements is verified by closed loop simulation experiments. For this purpose, the verification plant model should be as near as possible to the real plant. (4) Realizable control code for the target automation system must be generated. In order to avoid programming errors and remove the requirement of target system specific knowledge automatic code generation from

the controller verification simulation, see Fig. 2, is at least desirable. (5) Finally, the target system has to be programmed and tested on the real plant. The real experiments can be measured and analyzed.

The proposed tool-chain: (1) The derivation of a plant model in symbolic form as a basis for control design can be done per hand or with, e.g., commercial (Mathematica, Maple) or free (Maxima, SymPy, Sage) tools but is not considered in this paper. An obtained signal based type of plant model (e.g., ODEs) can be implemented in the standard Scilab-XCos. Scilab is a free and open source software for numerical computation maintained by Scilab Enterprises similar to the commercial Matlab from Mathworks. The graphical dynamical system modeler in Scilab is called XCos and the counterpart to Simulink from Mathworks. From an engineering point of view, the out-of-the-box incorporation of the Modelica based Coselica (see [7]) is a valuable tool for designers used to standard electrical and mechanical system blocks and especially of interest for a more realistic verification plant model of step (3). Linear control design and system identification is done inside Scilab (part of step (1) and step (2)). The major contribution of this paper is to the steps (4) and (5) and is discussed in more detail in the following section. Concerning step (4) there are several possibilities to generate C-code from an existing XCos (or the former SciCos) schematics. In order to name a few:

- Gene-Auto, [3] : The Gene-Auto project has created an open-source tool set which converts an application described in a high-level modeling language like Scicos to C-code. Unfortunately, this tool – apart from a commercialized toolbox – only works with the previous version of Scilab, called Scicos.
- Realtime Linux as target system, [2]: It is one of the longer known tool-chains, but it supports the outdated Scicos and is not really applicable to small embedded systems, i.e. microcontrollers.
- Scicos-FLEX, [4]: It is somehow a port of [2] for some microcontrollers, but outdated too.

Since all solutions are outdated (or commercialized) we make use of our own generator X2C. As target for step (5) an ARM Cortex-M device is discussed and the free EmBlocks, see [8], is used. There, a general hardware project with the input-output mapping for the STM32F4-discovery board (approx. 15$) has been implemented, that directly includes the auto-generated controller code. In order to transfer the program to the target, the discovery on-board debugging probe or a free JTAG device, e.g., [9], can be used. Additionally, there is ongoing development in order to establish an industrial control system (PLC from Bernecker and Rainer, b&r) as an industrial target. Further, the X2C Scope utility allows taking measurements (free-run and triggered) and the X2C Communicator parameter modifications and interaction directly from/on/with the target system. It is worth mentioning that the range of features of Scilab can be extended with add-ons. For example there exists a add-on called “plotting library” that helps the user to make plots with a similar syntax as in Matlab – useful for step (5).

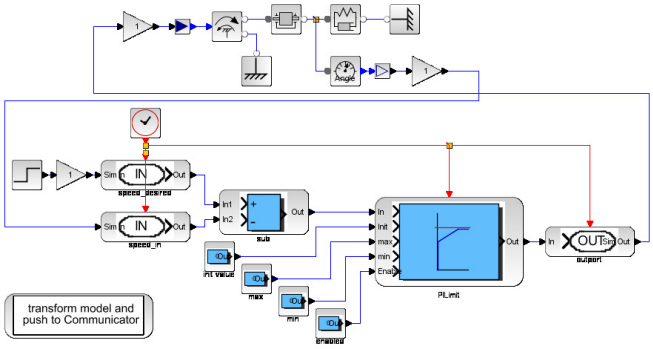

Fig. 2. Typical system model with plant (top) and controller (bottom).

A. The code generator X2C

The predecessor of the code generator X2C, see [5], was originally developed more then ten years ago at the JK-University Linz, Austria as a Simulink extension generating assembler code for TI-DSPs. Later, the system was extended to generate C-code and to largely comply with MISRA (S2C), see [6]. This long history and the simple effective concept of the code generator system stand for stability of its main kernel.

X2C natively includes into XCos and can be simulated in parallel with plant and the controller, see Fig. 2. For the plant blocks of the XCos library are used, Coselica in this case. The controller part is modeled using dedicated X2C-XCos blocks. These blocks are full featured XCos blocks extended with an enhanced parameter editor and the connection to the back-end for e.g. code generation. All the glue code needed for these X2C-XCos blocks is fully auto generated from the X2C block’s model. It is possible and also available to automatically generate blocks for other simulation environments. Special blocks are used to model and specify the targets in-ports and out-ports which may correspond to, e.g., analog, digital or PWM input or output ports (IOs).

This system model shall be simulated to provide detailed feedback about the expected performance of the overall system for further optimization. In this simulation the blocks are implementing exactly the code which will run on the target. Thus effects like quantization, fixed point arithmetic artifacts and discretisation are included in the simulation model. In simulation the developer has easy access to all signals of the plant and controller to analyse the behaviour or to inject faults for testing.

To move on to the target implementation the model transformation and code generation is executed by a single mouse click. The model transformation will step through the XCos model ignoring all non-X2C-XCos blocks to detect all X2C-XCos blocks and their associated clock domain and hence the sample time. X2C features the use of multiple sample times for blocks. The result is an abstract model implemented in Java holding all relevant information about the blocks, their parameters and their connections. Further

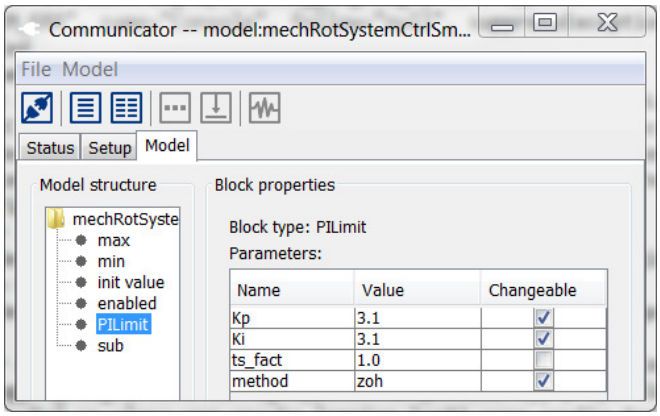

Fig. 3. The X2C Communicator with loaded model showing block's parameter.

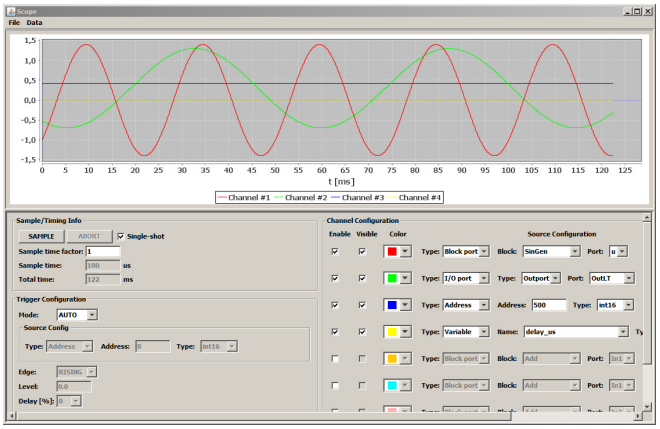

Fig. 4. X2C Scope: online access to IO ports, block ports, variables and registers of the target.

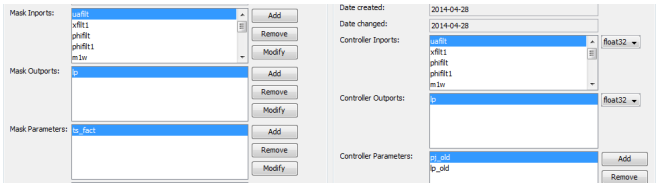

Fig. 5. X2C Block Generator: generate templates for own blocks.

on the code generator is applied on this abstract model using methods from graph theory to check, to partition and order the model and to generate the final code. The generated code is written as substantially MISRA[6] conform ANSI C code in object oriented style. During this process the parameters specified in the blocks, usually in the continuous time or frequency domain, are automatically converted to implementation specific discrete time domain parameters.

During design of the code generator attention has been paid on generation of human friendly, readable code. The so generated code facilitates code review, debugging and potential long term maintenance.

The central tool for the developer is the so called Communicator, see Fig. 3. The Communicator is the home of the code generator (fixed-point, single and double data typing is available) and the interface to the modeling / simulation environment and the target. An optional part of X2C is a quite basic operating system for various target platforms. This operating system features the communication protocol to connect the Communicator, a flash algorithm to deploy the developed control software, to change parameters online, to record data and more. In case of application of the whole operating system no specific target IDE or hardware programmer besides the target compiler is necessary, because the host-target link is established by standard serial (UART, USB) or network connections.

For effective development rapid feedback on changes in design is valuable. With X2C the way from an adapted model to a running system on the target is short even when the whole tool chain of code generator and compiler has to be applied. A highlight of X2C is that in many cases this is not necessary. With X2C it is reality that e.g. parameters can be changed in the model or in the Communicator and the parameters on the target are updated instantly. That means manually tuning controller parameters becomes a task of selecting the block and parameter in the XCos diagram and using the keyboard or mouse wheel to tune the parameter while watching the plant and the automatically updated Scope for feedback. The Scope, see Fig. 4, is featuring functionality like a conventional oscilloscope. Through the Scope the developer has access to online data of block ports, I/O ports, variables and registers of the target in a way like in the simulation environment.

1) User defined blocks: Own code fragments can be incorporated into simulation and code generation with the help of a dedicated block generator. The user specifies the in- and outputs of the block and the X2C Block Generator generates the necessary commented template files (in ANSI-C) afterwards. The behaviour of the block is included into the templates by the user again, and can be used for simulation and implementation. In the demonstration application this feature is applied in order to implement a recursive least square algorithm for the adaptive controller.

B. EmBlocks and hardware programming

Most microcontroller developers require full insight into the target programming and a (JTAG) hardware probe for full on-chip debugging features instead the small operating system approach already mentioned. Towards this end, we use a free microcontroller IDE for target programming (step (5) from above), see [8], that supports a vast number of targets. EmBlocks is a full-featured free embedded IDE with stack, register, etc. views and a number of supported hardware probe interfaces. In Fig. 6 the initialization part of the heavy commented X2C auto-generated code in EmBlocks is presented. As soon as standardized names for block IOs in XCos are used only one general template project is required per target, where the hardware specific code parts and especially the input-output mapping take place. As soon as this is achieved for a specific target the user needs just to press the compile and program button. As already mentioned, the small operating system can be used alternatively.

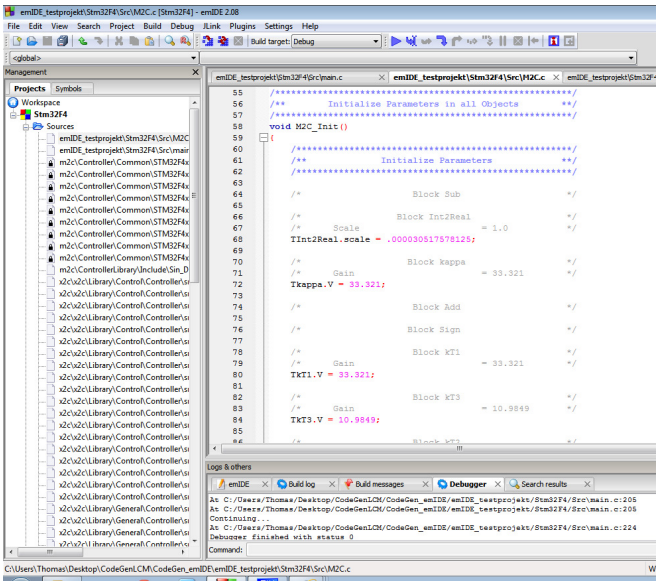

Fig. 6. The EmBlocks IDE with the generated initialization code.

C. Different Targets and step (5)

As for any other code generator the user acceptance relies among other things essentially on the number of supported targets. This is the reason why we favor two target approaches, namely the tight integration by the small operating system approach best fitted for student use and rapid controller prototyping on the real experiments, and the quite versatile approach where all target hardware specific part is done in the IDE (for the embedded systems programming engineer). The tight integration approach requires some more target specific details if a new target is made but releases the user from target hardware interactions. The second IDE-based approach is very easily extendable to new targets. At the moment some TI targets and one for a STM32F0/1 do exist using the small operating system approach.

For the STM32F4-discovery target discussed in this paper we have chosen the free EmBlocks route for the first time. The reason is the feature rich 32bit Flash CortexM4 based MCU very well fitted for general control purposes:

- up to 180 MHz/225 DMIPS, with DSP instructions, floating point unit (FPU; single precision) and advanced peripherals
- 2 DAC 12bit, 3x 12bit ADC (24 channels)
- synchronized PWM-timers, quad-encoder channels
- available discovery-board (approx. 15$; programmer and debugging probe already on-board; just requires USB and serial (for scope) connection)

Towards industrial targets (Programmable Logic Controller, PLC) there is ongoing development concerning the application to a b&r Powerpanel (PP400) using the IDE approach with the target specific Automation Studio.

III. Application: Cart and Pendulum System

As a reference application the well known cart and pendulum system in gantry crane configuration is presented, see

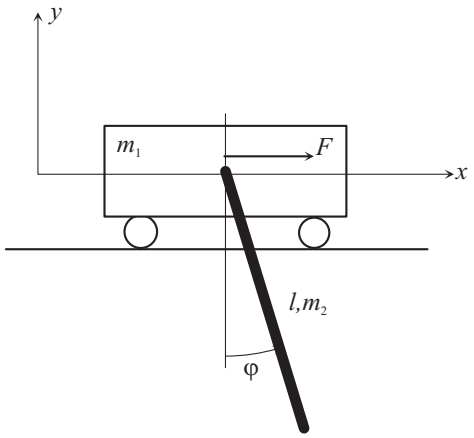

Fig. 7. The well known cart and pendulum system.

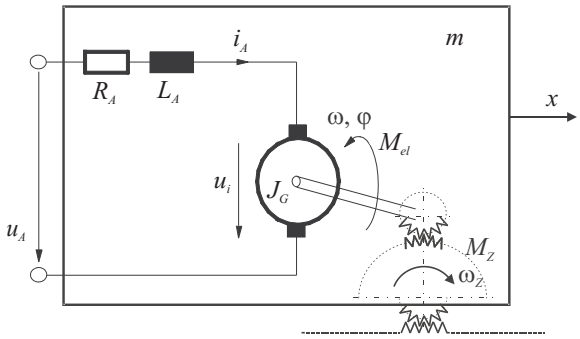

Fig. 8. Armature equivalent circuit and power conversion of a dc drive with external excitation.

Fig. 7. As common to real world crane applications at least the equivalent length to the center of mass of the load is unknown. Instead, we assume the equivalent length of the pendulum rod l_2 as unknown but constant. The cart with mass m_1 is driven by a dc drive with external excitation, see Fig. 8.

In the following, we assume a very small electrical time constant ($\tau_{el} = \frac{L_A}{R_A}$, with armature inductance L_A and resistance R_A) compared to the mechanical one. By means of system reduction, i.e, $L_A \to 0$, we introduce equivalent parameters integrating the whole drive-train (drive constant k_m, inertia J_A, transmission ratio n, gear pinion radius r, and some mechanical damping d_1) into the mathematical model of the cart. Other essential parameters are explained in Tab I.

$$\begin{aligned} \tilde{m}_1 &= m_1 + J_A \left(\frac{n}{r}\right)^2 \\ \tilde{d}_1 &= d_1 + \frac{n^2 k_m^2}{r^2 R_A} \end{aligned} \tag{1}$$

The model equations of the nonlinear reduced system are written in the form $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{Q} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} - \mathbf{D}(\mathbf{q}, \dot{\mathbf{q}})$, with the mass matrix $\mathbf{M}(\mathbf{q})$, centrifugal- and Coriolis terms $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$, dissipative terms in the vector $\mathbf{D}(\dot{\mathbf{q}})$and generalized forces $\mathbf{Q}$. $\mathbf{q}^T = [x,\, \varphi]$ denotes the vector of generalized coordinates and $[v,\, \omega]$ the corresponding velocities. Further, we use the abbreviation $\beta = n\, k_m\, (r\, R_a)^{-1}$ and the static

Name	Description
m_1	mass cart
m_2	mass pendulum
d_1	friction coefficient cart
d_2	friction coefficient pendulum
l_2	length pendulum
J_A	moment of inertia of drivetrain
n	transmission ratio
r	radius pinion
k_m	motor constant
R_A	terminal resistance

TABLE I
SYSTEM PARAMETERS.

friction F_C is ignored in the model equations, because it's compensated in all measurements by the well-known approach.

$$\begin{bmatrix} \tilde{m}_1 + m_2 & \frac{1}{2} m_2 l_2 cos(\varphi) \\ \frac{1}{2} m_2 l_2 cos(\varphi) & \frac{1}{3} m_2 l_2^2 \end{bmatrix} \cdot \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} m_2 l_2 sin(\varphi) \omega^2 - \tilde{d}_1 v + \beta u_A \\ -\frac{1}{2} m_2 g l_2 sin(\varphi) - d_2 \omega \end{bmatrix} \quad (2)$$

For the calculation of a linear state controller the system has to be linearized. The linearized model (around $\mathbf{q_S} = \begin{bmatrix} x_S & \varphi_S & v_S & \omega_S \end{bmatrix}^T = \begin{bmatrix} 0 & k\pi & 0 & 0 \end{bmatrix}^T, k = 0, 2, ...$) can be written as

$$\begin{bmatrix} \Delta\dot{x} \\ \Delta\dot{\varphi} \\ \Delta\dot{v} \\ \Delta\dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{3gm_2}{\bar{m}_2} & -\frac{4\tilde{d}_1}{\bar{m}_2} & \frac{6d_2}{l_2\bar{m}_2} \\ 0 & -\frac{6g\bar{m}_1}{l_2\bar{m}_2} & \frac{6\tilde{d}_1}{l_2\bar{m}_2} & -\frac{12d_2\bar{m}_1}{m_2 l_2^2 \bar{m}_2} \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ \Delta\varphi \\ \Delta v \\ \Delta\omega \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{4\beta}{\bar{m}_2} \\ -\frac{6\beta}{l_2\bar{m}_2} \end{bmatrix} \cdot u_A \quad (3)$$

with the substitutions $\bar{m}_1 = \tilde{m}_1 + m_2$ and $\bar{m}_2 = 4\tilde{m}_1 + m_2$ and unknown but constant pendulum equivalent length l_2. This completes the modelling part of step (1) and has to take place outside the tool-chain.

A. System identification and adaptive control design

As already mentioned, the pendulum equivalent length l_2 is assumed unknown but constant.

In order to get rid of the time derivatives the fourth equation of the linearized system model is transformed from the time-domain to the frequency-domain. For brevity, initial states are assumed equal to zero and $d_2 = 0$.

$$s^2 \hat{\varphi} l_2 \bar{m}_2 = -6\beta \hat{u}_A - 6g\bar{m}_1 \hat{\varphi} + 6\tilde{d}_1 s\hat{x} \quad (4)$$

We apply the realizable stable filter with free coefficients α_i to both sides

$$F_0 = \frac{\alpha_0}{s^2 + \alpha_1 s + \alpha_0} \quad (5)$$

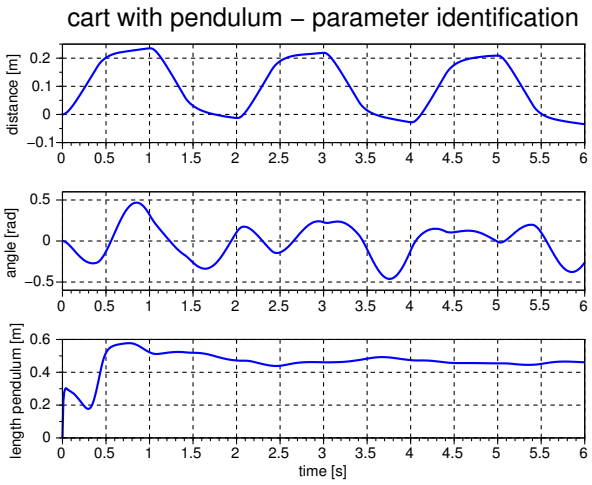

Fig. 9. Offline parameter identification with RLS in Scilab-XCos on true measurements, l_2=46cm.

$$\frac{s^2}{s^2 + \alpha_1 s + \alpha_0} \hat{\varphi} \alpha_0 l_2 \bar{m}_2 = -\frac{\alpha_0}{s^2 + \alpha_1 s + \alpha_0} \hat{u}_A 6\beta - \frac{\alpha_0}{s^2 + \alpha_1 s + \alpha_0} \hat{\varphi} 6g\bar{m}_1 + \frac{s}{s^2 + \alpha_1 s + \alpha_0} \hat{x} \alpha_0 6\tilde{d}_1 \quad (6)$$

see [10]. With standard polynomial division one eliminates the non-strictly proper transfer function. Therefore, it is sufficient to implement the two filters $F_0(s)$ and $F_1(s) = sF_0(s)$. Both filters have common denominator and can be realized as one dynamical system if applied to the same signal. The inverse Laplace transform leads to one data line of the algebraic equation system for identification

$$\left[\alpha_0 \left(\varphi - \tfrac{\alpha_1}{\alpha_0} F_1 * \varphi - F_0 * \varphi\right) \bar{m}_2\right] \left[\theta_1\right] = -F_0 * u_A 6\beta - F_0 * \varphi 6g\bar{m}_1 + F_1 * x 6\tilde{d}_1 \quad (7)$$

linear in the unknown parameter $\theta_1 = l_2$, whereby $*$ indicates the convolution operator in time-domain. For implementation purposes the filters have to be discretized. The filter on plant input u_A is discretized naturally by a zero-order hold but the filters on the states have to be approximated. A trapezoidal approximation, i.e., the Tustin method, is usually sufficient without requiring an unnatural small sample time compared to system dynamics. The unknown parameter can be estimated using a standard recursive least square algorithm. For identification results with true plant measurements logged into a file and the RLS algorithm in XCos the reader is kindly referred to Fig. 9. This completes the identification part of step (1). For the adaptive self-tuning control (STC) approach (step (3)) the filters and a standard RLS algorithm is implemented in combination with a linear state control law parametrized in l_2, see below.

B. Experiment - Cart and pendulum

In order to demonstrate the physical modelling capabilities the Coselica toolbox is used for plant modeling, see

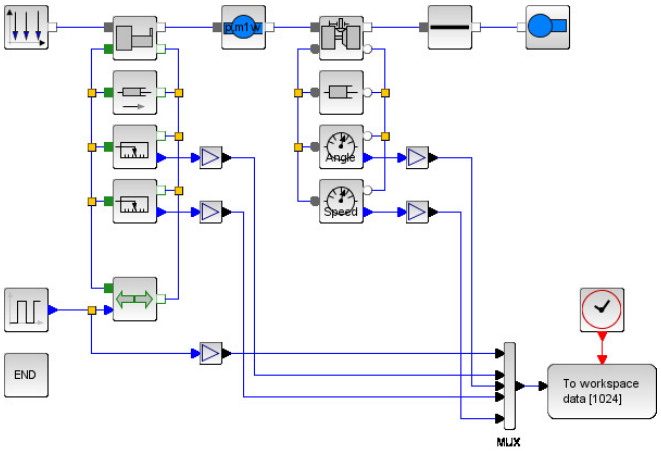

Fig. 10. Cart with rod pendulum - Coselica.

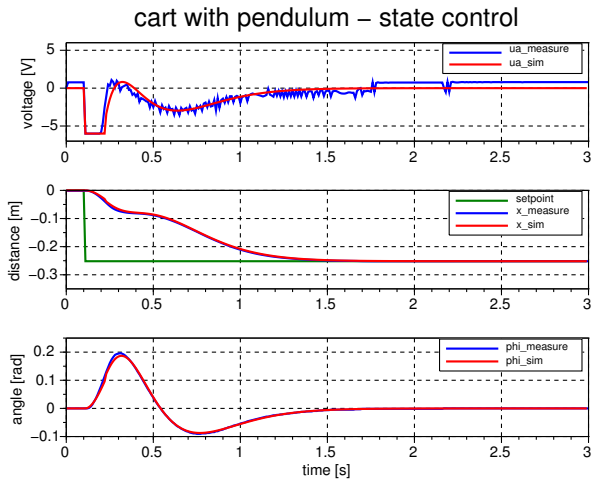

Fig. 11. Comparison of simulation and true experiment with standard state control and known length l_2.

Fig. 10. The quality of the overall model compared to the true experiment can be seen in Fig. 11 where a standard linear state controller based on fixed identified parameters is implemented in simulation (step (3)) and real experiment (steps (4) and (5)). Velocities are obtained by first order derivative approximations.

1) Adaptive STC Control: In order to met the proposed goal STC control with unknown but constant l_2 is applied to the simulated plant. The first 1.5s are used for settling the online RLS identificator and the filters as derived above. Then the parametrized control law $u_A = -\mathbf{k}\,(l_2)^T\,\mathbf{x}$ is activated, see Fig. 12.

$$\mathbf{k} = \begin{bmatrix} -208.4 l_2 \\ -8.4 + 201.8 l_2 - 138.9 l_2^2 \\ 9 - 106.9 l_2 \\ (17.3 - 71.3 l_2) l_2 \end{bmatrix}$$

As expected, the control law forces the pendulum angle to zero at rest whereas a quite good positioning performance is achieved for the cart.

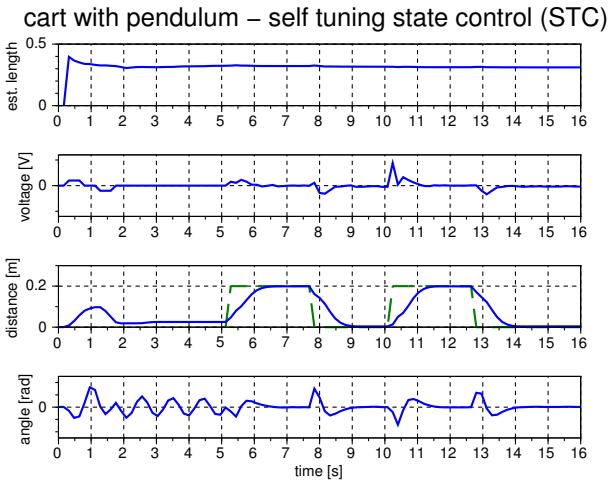

Fig. 12. STC control experiment (slashed lines means inactive signals), l_2=33cm, setpoint dashed and activated at approx. 5s.

IV. Conclusion

The presented work will be available for free – to a large extend under a BSD license – within the Q2 of 2014, see the references [5].

Ongoing development is targeted towards efficient handling of vectorized signal lines, Scilab-code to C-code conversion, more block libraries, industrial targets, IEC 61131-3 code generation (PLC), adaption of the FMI (functional mockup interface) for model exchange with various commercial simulators, and some state machine concept. For most parts we plan to adapt again existing free tools.

We gratefully acknowledge the support from the Austrian funding agency FFG in the COIN-project ProtoFrame and the internal funding in the project MOdoPS. Part of this work was conducted within the research project “sustainable and resource saving electrical drives through high energy and material efficiency” sponsored within the EU program “(Regio 13)” and within the COMET K2 center ACCM. Further, we are indebted to our master students Höglinger Thomas and Würflinger Florian for testing and measurement preparation.

References

[1] Scilab: Free and Open Source software for numerical computation, Scilab Enterprises, 2012, http://www.scilab.org/.
[2] Roberto Bucher, et al., RTAI-Lab tutorial: Scilab, Comedi, and real-time control, 2006.
[3] Ana-Elena Rugina, et al., Gene-Auto: Automatic Software Code Generation for Real-Time Embedded Systems, DASIA 2008.
[4] Scicos-FLEX code generator, http://erika.tuxfamily.org/.
[5] X2C in Scilab-Xcos, 2013, http://www.mechatronic-simulation.org/.
[6] MISRA Consortium, 2013, http://www.misra.org.uk/.
[7] Reusch, 2013, http://www.kybdr.de/software.
[8] EmBlocks the free IDE for embedded programming, 2013, http://www.emblocks.org/.
[9] Open On-Chip Debugger, a open JTAG interface, 2013, http://openocd.sourceforge.net/.
[10] JJE. Slotine, W. Li, Applied nonlinear control, Prentice Hall, 1991.