



# **Model based design (MBD) – a free tool-chain**

**Simon Mayr, (Gernot Grabmair)**

Simon Mayr, [simon.mayr@fh-wels.at](mailto:simon.mayr@fh-wels.at), University of Applied Sciences Upper Austria , Austria

# Projects

## Projects dealing with Scilab/XCos



- **PROTOFRAME – Framework und frontend for semi-automated matching of real and virtual prototypes**
  - Work in progress
  
- **MOdoPS – MOdel based Design by OPen Source**
  - Project finished
  
  - Project result: Scilab/XCos example library

# Overview

- Model based design (MBD)
- Code generation from XCos
- Example (cart and pendulum)
- Conclusion

# Model based design (MBD)

## Definition



- Mathematical and visual method applied in designing embedded software to address problems associated with
  - > Complex control
  - > Signal processing
  - > Communication systems

# Model based design (MBD)

## Applications



- Common fields of application are:
  - > Motion control applications
  - > Industrial equipment
  - > Aerospace applications
  - > Automotive applications
  - > ...

# Model based design (MBD)

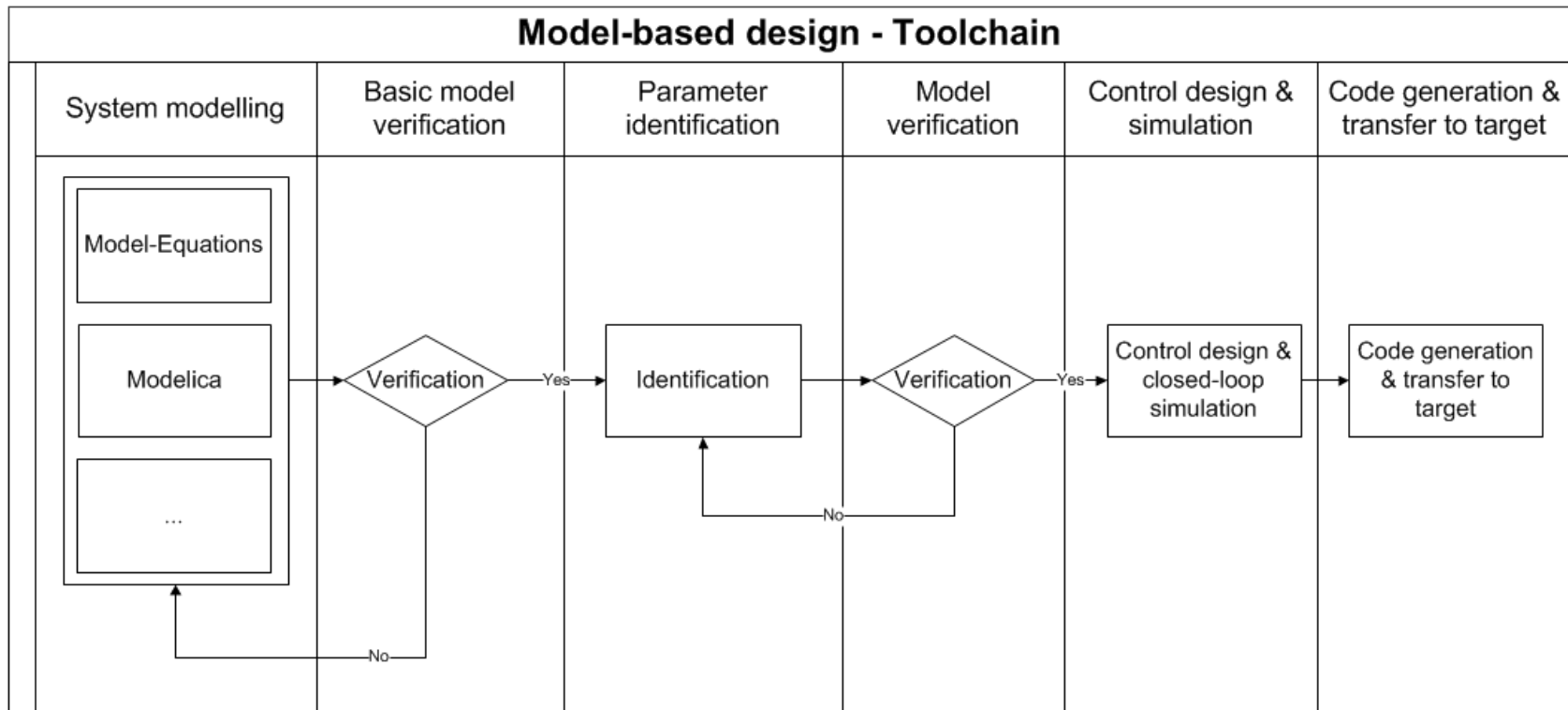
## Main steps of model based (controller) design



- System modelling and basic model verification
- Parameter identification and model verification
- Control design and closed-loop simulation
- Code generation and transfer to target

# Model based design (MBD)

## Main steps of model based (controller) design



# Model based design (MBD)

## Advantages



- Faster and more cost-efficient development
- Errors in system design can be located and corrected in early stage of the project, when financial and time impacts of the system redesign are relatively small
- Extension and/or modification of an existing system is relatively easy



# Model based design (MBD)

## Common commercial tool-chains



- Typical examples of commercial tools are:
  - Matlab/Simulink
  - Dymola
  - ...
- Advantages:
  - Advanced and well-proven software
  - Complete tool-chains
- Disadvantages:
  - Quite expensive
  - Unsuitable for small and medium-sized companies

# Model based design (MBD)

## Free tool-chain



- Scilab/XCos
  
- Advantages:
  - > Plant modeling
  - > Control design & simulation
  
- Disadvantages:
  - > Code generation is not implemented
  
- Solution:
  - > Use an external application to generate code from XCos diagram

# Code generator for Scilab/XCos

## State of the art



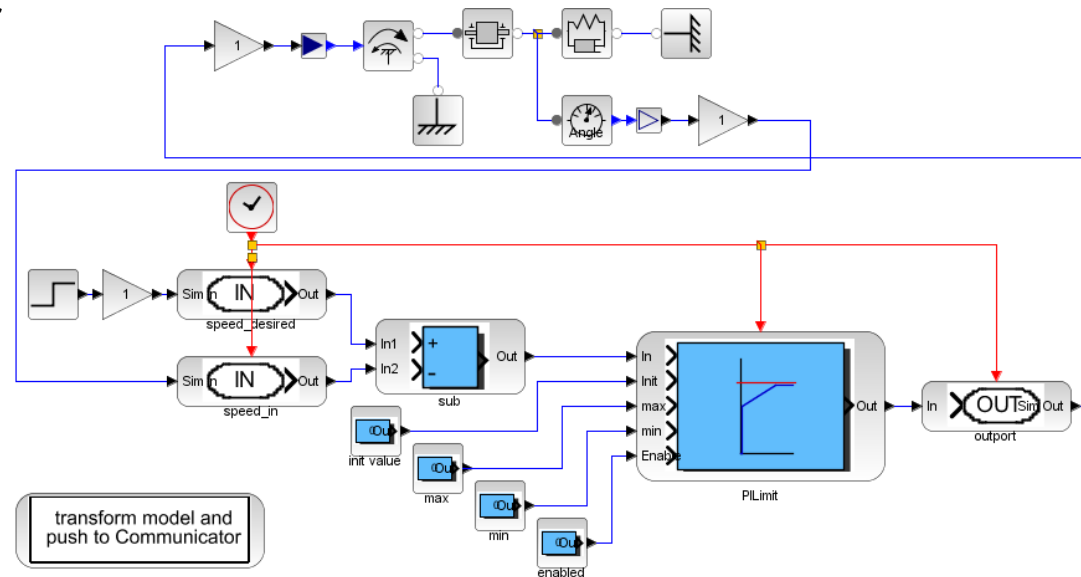
- Existing code generators for the outdated Scilab/Scicos:
  - RTAI [3]
  - Gene-Auto [4]
  - Scicos-FLEX [5]
  
- Code generators for Scilab/XCos:
  - Project-P [6]
  - **X2C from JKU-Linz (Upper Austria) [2]**

# Code generator for Scilab/XCos

## Code generator X2C



- The predecessor of X2C was developed more than 10 years ago at the JK-University Linz, Austria as a Simulink extension
- X2C natively includes into XCos and can be simulated in parallel with plant and controller



# Code generator for Scilab/XCos

## Code generator X2C



- X2C-blocks are full featured XCos-blocks extended with an parameter editor and the connection to the back-end for code generation
- In XCos simulation the X2C-blocks are implementing exactly the code which will run on the target
- Model transformation and code generation is executed by a simple mouse click. All non-X2C-blocks are ignored during this process.

# Code generator for Scilab/XCos

## Code generator X2C



- The central tool is the so called „Communicator“. It's the interface between simulation environment and target.
- The Communicator features
  - > Code generation
  - > Change parameters in the model or in the communicator, and the parameters on the target are updated instantly
  - > Scope (software oscilloscope)

# Code generator for Scilab/XCos Communicator and scope



Communicator -- model:mechRotSystemCtrlSm...

File Model

Status Setup Model

Model structure

- mechRotSystem
  - max
  - min
  - init value
  - enabled
  - PLimit**
  - sub

Block properties

Block type: PLimit

Parameters:

Name	Value	Changeable
Kp	3.1	<input checked="" type="checkbox"/>
Ki	3.1	<input checked="" type="checkbox"/>
ts_fact	1.0	<input type="checkbox"/>
method	zoh	<input checked="" type="checkbox"/>

Scope

File Data

Channel #1 Channel #2 Channel #3 Channel #4

Sample/Timing Info

SAMPLE ABORT  Single-shot

Sample time factor: 1

Sample time: 100 us

Total time: 122 ms

Trigger Configuration

Mode: AUTO

Source Config

Type: Address Address: 0 Type: int16

Edge: RISING

Level: 0.0

Delay [%]: 0

Channel Configuration

Enable	Visible	Color	Type	Block	Source Configuration
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Red	Block port	Block: SmGen	Port: u
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Green	I/O port	Type: Outport	Port: OutLT
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Blue	Address	Address: 500	Type: int16
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Yellow	Variable	Name: delay_us	Type: Ty
<input type="checkbox"/>	<input type="checkbox"/>	Yellow	Block port	Block: Add	Port: In1
<input type="checkbox"/>	<input type="checkbox"/>	Cyan	Block port	Block: Add	Port: In1
<input type="checkbox"/>	<input type="checkbox"/>	Pink	Block port	Block: Add	Port: In1

# Code generator for Scilab/XCos

## User defined X2C-blocks



- It's possible to generate user defined X2C blocks easily with the help of a dedicated block generator
- Inputs, outputs, control parameters and data types are specified by the user
- This information is used to generate a code template automatically
- The behavior of the block is included by the user
- This blocks can be used for simulation and implementation on target



# Code generator for Scilab/XCos

## X2C-block generator



**X2C Block Generator**

File

**Block Parameters**

Name: RLSCartPendulum  
Prompt: RLS Block Cart Pendulum  
Description:  
License: BSD 3-clause [Edit]  
Library: [Change configuration]  
ID: 559  
Author: Simon Mayr  
Revision: 0.1  
TexFile (optional):  
Date created: 2011-07-11  
Date changed: 2014-04-28  
Mask Inputs: uafilt, xfilt1, phifilt, phifilt1, m1w [Add, Remove, Modify]  
Mask Outputs: lp [Add, Remove, Modify]  
Mask Parameters: ts\_fact [Add, Remove, Modify]  
Visualizations: [Add Matlab, Add ScilabXcos, Remove, Modify]  
 Direct feedthrough

**Implementations**

Current Implementation: float32 [Add]  
Default Implementation: float32 [Remove, Set Default]

**Implementation Parameters**

Name: float32  
Display name: 32 bit floating point  
Description: 32 bit floating point implementation  
ID: 0  
Author: Simon Mayr  
Revision: 0.1  
Date created: 2014-04-28  
Date changed: 2014-04-28  
Controller Inputs: uafilt, xfilt1, phifilt, phifilt1, m1w [float32]  
Controller Outputs: lp [float32]  
Controller Parameters: pj\_old, lp\_old [Add, Remove, Modify]  
Conversion Function Type: NONE [Advanced]

# Code generator for Scilab/XCos

## Code template and user code



```

35 /* USERCODE-BEGIN:Description */
36 /* Description: */
37 /* USERCODE-END:Description */
38 #include "RLSCartPendulum_float32.h"
39
40 /* all used update functions to ram for c2000 */
41 #if defined(__ALL_UPDATE_FUNC_2_RAM_C2000__)
42 #pragma CODE_SECTION(RLSCartPendulum_float32_Update, "ramfuncs")
43 /* individual added update functions to ram for c2000 */
44 #elif defined(__ALL_UPDATE_FUNC_SEPARATE_SECT_DEF_C2000__)
45 #pragma CODE_SECTION(RLSCartPendulum_float32_Update, "RLSCartPendulum_float32_Update_Sect")
46 #endif
47
48 /* USERCODE-BEGIN:PreProcessor */
49 /* USERCODE-END:PreProcessor */
50
51 /******
52 ** RLSCartPendulum_float32_Update **
53 *****/
54 void RLSCartPendulum_float32_Update(RLSCARTPENDULUM_FLOAT32 *pTRLSCartPendulum_float32)
55 {
56 /* USERCODE-BEGIN:UpdateFnc */
57 /* USERCODE-END:UpdateFnc */
58 }
59
60 /******
61 ** RLSCartPendulum_float32_Init **
62 *****/
63 void RLSCartPendulum_float32_Init(RLSCARTPENDULUM_FLOAT32 *pTRLSCartPendulum_float32)
64 {
65 pTRLSCartPendulum_float32->ID = RLSCARTPENDULUM_FLOAT32_ID;
66 pTRLSCartPendulum_float32->lp = 0;
67 /* USERCODE-BEGIN:InitFnc */
68 /* USERCODE-END:InitFnc */
69 }
70
71 /******
72 ** RLSCartPendulum_float32_Load **
73 *****/
74 uint8 RLSCartPendulum_float32_Load(const RLSCARTPENDULUM_FLOAT32 *pTRLSCartPendulum_float32, uint8 data[])
75 {
76 uint8 loadSize;
77
78 data[0] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->pj_old) & 0x000000FF));
79 data[1] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->pj_old) >> 8) & 0x000000FF);
80 data[2] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->pj_old) >> 16) & 0x000000FF);
81 data[3] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->pj_old) >> 24) & 0x000000FF);
82 data[4] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->lp_old) & 0x000000FF));
83 data[5] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->lp_old) >> 8) & 0x000000FF);
84 data[6] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->lp_old) >> 16) & 0x000000FF);
85 data[7] = (uint8)((*(uint32*)&(pTRLSCartPendulum_float32->lp_old) >> 24) & 0x000000FF);
86 loadSize = (uint8)8;
87 /* USERCODE-BEGIN:LoadFnc */
88 /* USERCODE-END:LoadFnc */
89 return (loadSize);
90 }

```

```

35 /* USERCODE-BEGIN:Description */
36 /* Description: */
37 /* USERCODE-END:Description */
38 #include "RLSCartPendulum_float32.h"
39
40 /* all used update functions to ram for c2000 */
41 #if defined(__ALL_UPDATE_FUNC_2_RAM_C2000__)
42 #pragma CODE_SECTION(RLSCartPendulum_float32_Update, "ramfuncs")
43 /* individual added update functions to ram for c2000 */
44 #elif defined(__ALL_UPDATE_FUNC_SEPARATE_SECT_DEF_C2000__)
45 #pragma CODE_SECTION(RLSCartPendulum_float32_Update, "RLSCartPendulum_float32_Update_Sect")
46 #endif
47
48 /* USERCODE-BEGIN:PreProcessor */
49 // Inputs
50 #define UAFILT (*pTRLSCartPendulum_float32->uafilt)
51 #define XFILT1 (*pTRLSCartPendulum_float32->xfilt1)
52 #define PHI (*pTRLSCartPendulum_float32->phi)
53 #define PHIFILT (*pTRLSCartPendulum_float32->phifilt)
54 #define PHIFILT1 (*pTRLSCartPendulum_float32->phifilt1)
55 #define MIW (*pTRLSCartPendulum_float32->miw)
56 #define DIW (*pTRLSCartPendulum_float32->diw)
57 #define MP (*pTRLSCartPendulum_float32->mp)
58 #define BETA (*pTRLSCartPendulum_float32->beta)
59 #define G (*pTRLSCartPendulum_float32->g)
60 #define A1 (*pTRLSCartPendulum_float32->a1)
61 #define A0 (*pTRLSCartPendulum_float32->a0)
62
63 // Outputs
64 #define LP (pTRLSCartPendulum_float32->lp)
65
66 // Control parameters
67 #define LP_OLD (pTRLSCartPendulum_float32->lp_old)
68 #define FJ_OLD (pTRLSCartPendulum_float32->pj_old)
69
70 /* USERCODE-END:PreProcessor */
71
72 /******
73 ** RLSCartPendulum_float32_Update **
74 *****/
75 void RLSCartPendulum_float32_Update(RLSCARTPENDULUM_FLOAT32 *pTRLSCartPendulum_float32)
76 {
77 /* USERCODE-BEGIN:UpdateFnc */
78 float32 WJ, YJ, FJ_NEW;
79 YJ = 6*DIW*XFILT1 - 6*G*(MIW + MP)*PHIFILT - 6*BETA*UAFILT;
80 WJ = (PHI - PHIFILT - (A1/A0)*PHIFILT1)*(A0*(MP + 4*MIW));
81
82 FJ_NEW = FJ_OLD*(1-(PJ_OLD*WJ)/(1+WJ*WJ*PJ_OLD));
83 LP = LP_OLD*(PJ_OLD*WJ*(YJ-WJ*LP_OLD)/(1+WJ*WJ*PJ_OLD));
84
85 FJ_OLD = FJ_NEW;
86 LP_OLD = LP;
87 /* USERCODE-END:UpdateFnc */
88 }

```

# Experiment

## Cart and pendulum



- System modeling
- Plant simulation
- Parameter identification (pendulum length)
- Adaptive STC control
- Code generation
- Measurements

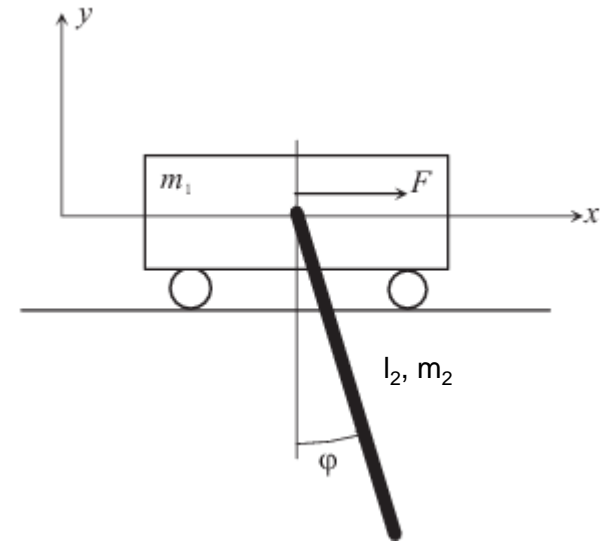
# Plant modeling

## Cart and pendulum

- Plant modeling can be either done by using mathematical terms (e.g. ODEs) or by using the Modelica-based Scilab-addon „Coselica“
- Plant modeling is shown by the cart and pendulum example.

$m_1$  ... vehicle mass  
 $d_1$  ... linear friction coefficient (cart)  
 $m_2$  ... pendulum mass  
 $d_2$  ... linear friction coefficient (pendulum)  
 $l_2$  ... pendulum length

$x$  ... distance (cart)  
 $v$  ... velocity (cart)  
 $\varphi$  ... angle (pendulum)  
 $\omega$  ... angular velocity (pendulum)



# Plant modeling

## Cart and pendulum

- Nonlinear system equations can be computed using the Lagrange formalism with the vector of generalized coordinates  $\mathbf{q} = [x, \varphi]^T$  and  $F = \beta u_A$ . Furthermore static friction  $F_C$  is ignored, because it's compensated.

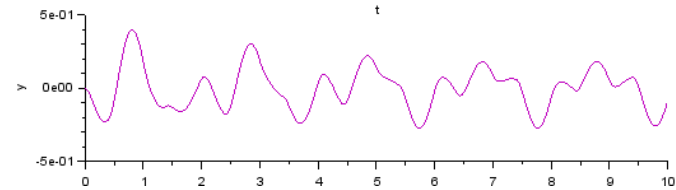
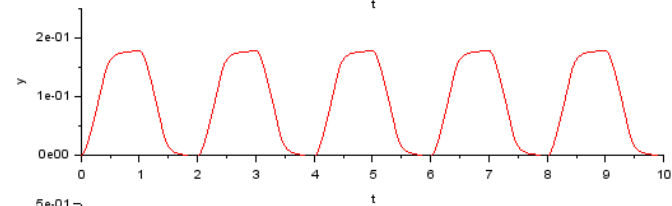
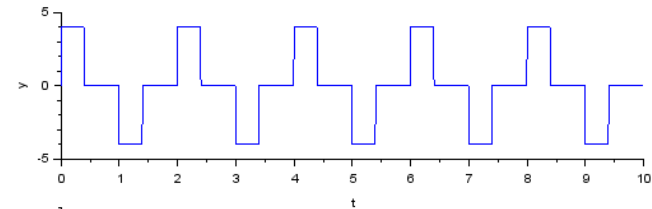
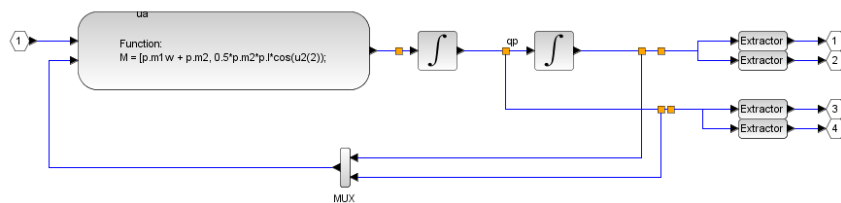
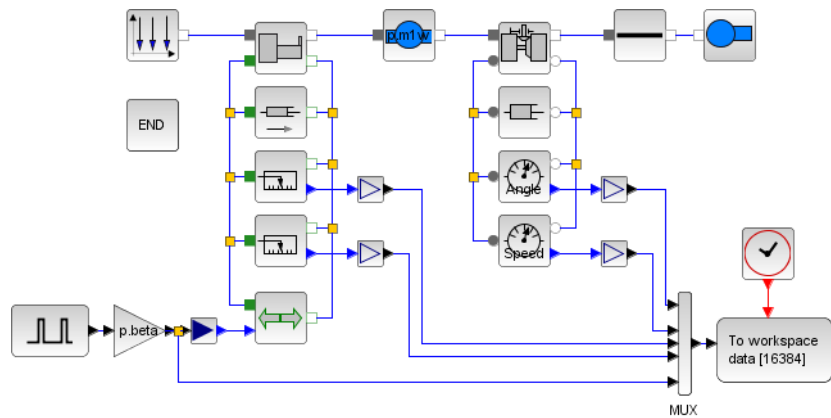
$$\begin{bmatrix} \tilde{m}_1 + m_2 & \frac{1}{2}m_2l_2\cos(\varphi) \\ \frac{1}{2}m_2l_2\cos(\varphi) & \frac{1}{3}m_2l_2^2 \end{bmatrix} \cdot \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}m_2l_2\sin(\varphi)\omega^2 - \tilde{d}_1v + \beta u_A \\ -\frac{1}{2}m_2gl_2\sin(\varphi) - d_2\omega \end{bmatrix}$$

- The linearized model (around  $q_S = [x_S, \varphi_S, v_S, \omega_S]^T = [0, k\pi, 0, 0]^T$ ,  $k = 0, 2, \dots$ ) can be written as

$$\begin{bmatrix} \Delta \dot{x} \\ \Delta \dot{\varphi} \\ \Delta \dot{v} \\ \Delta \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{3gm_2}{4\tilde{m}_1+m_2} & -\frac{4\tilde{d}_1}{4\tilde{m}_1+m_2} & \frac{6d_2}{l_2(4\tilde{m}_1+m_2)} \\ 0 & -\frac{6g(\tilde{m}_1+m_2)}{l_2(4\tilde{m}_1+m_2)} & \frac{6\tilde{d}_1}{l_2(4\tilde{m}_1+m_2)} & -\frac{12d_2(\tilde{m}_1+m_2)}{m_2l_2^2(4\tilde{m}_1+m_2)} \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ \Delta \varphi \\ \Delta v \\ \Delta \omega \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{4\beta}{4\tilde{m}_1+m_2} \\ -\frac{6\beta}{l_2(4\tilde{m}_1+m_2)} \end{bmatrix} \cdot u_A$$

# Plant modeling

## Coselica & ODE



# Parameter identification

- Identifying the unknown (but constant) pendulum length  $l_2$
- Assumption(s):
  - Pendulum friction is set to zero ( $d_2 = 0$ )
- The 4<sup>th</sup> line of the linearized model is used for identification
- To get rid of the time derivatives, the system equation is transformed into the laplace-domain

$$s^2 \hat{\varphi} l_2 (4\tilde{m}_1 + m_2) = -6\beta \hat{u}_A - 6g (\tilde{m}_1 + m_2) \hat{\varphi} + 6\tilde{d}_1 s \hat{x}$$

# Parameter identification

- We apply realizable stable filters  $F_0(s)$  and  $F_1(s) = sF_0(s)$  with free coefficients to the whole equation [1]

$$F_0(s) = \frac{\alpha_0}{s^2 + \alpha_1 s + \alpha_0} \qquad F_1(s) = sF_0(s)$$

- The inverse laplace transformation leads to one data line, linear in the unknown parameter (\* indicates the convolution operator in time-domain)

$$\left[ \alpha_0 \left( \varphi - \frac{\alpha_1}{\alpha_0} F_1 * \varphi - F_0 * \varphi \right) (4\tilde{m}_1 + m_2) \right] \left[ \theta_1 \right] = -F_0 * u_A 6\beta - F_0 * \varphi 6g (\tilde{m}_1 + m_2) + F_1 * x 6\tilde{d}_1$$

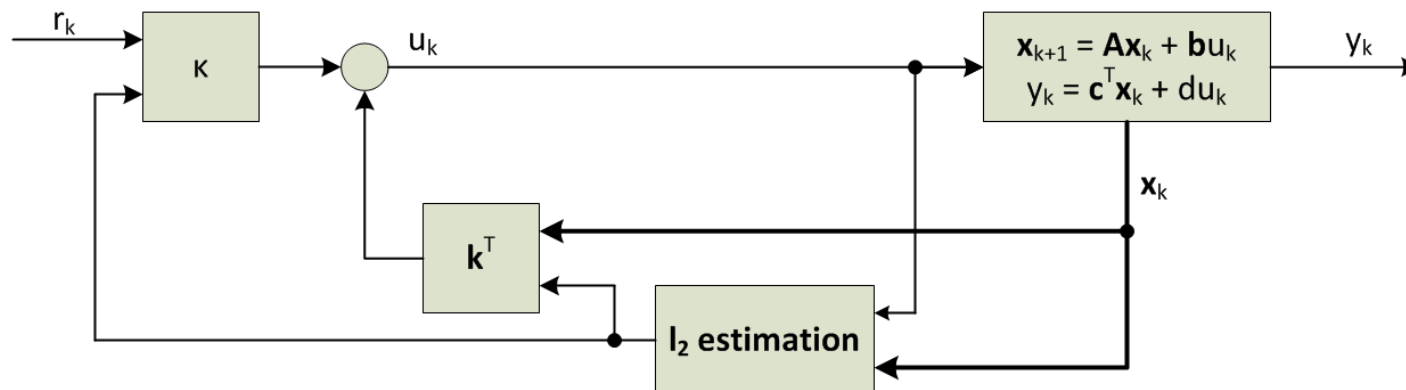
- Estimation of the parameter using recursive least square algorithm



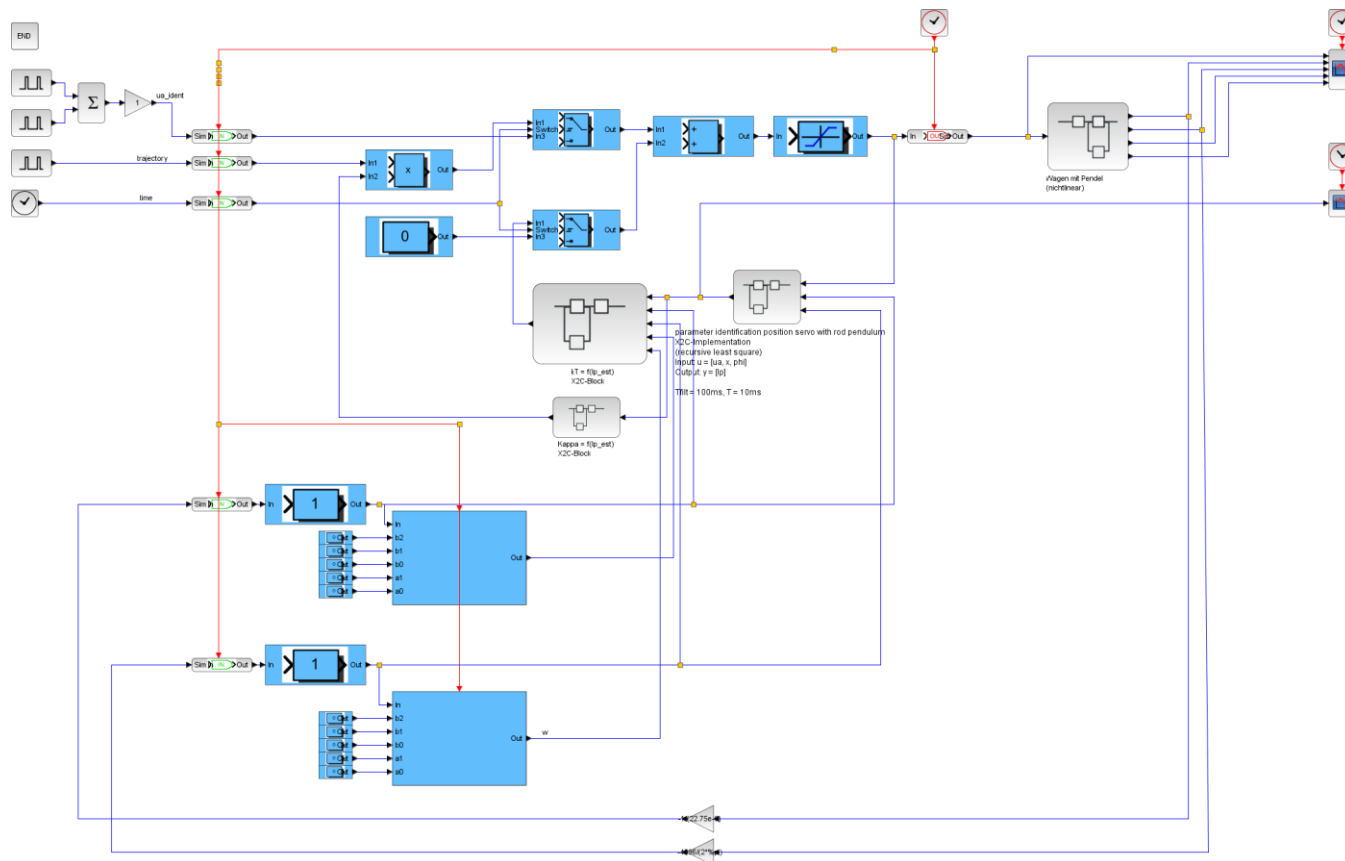
# Adaptive STC control

- Design a linear state control law parameterized in pendulum length  $l_2$

$$u_k = \mathbf{k}(l_2)^T \mathbf{x} + \kappa(l_2)r_k$$

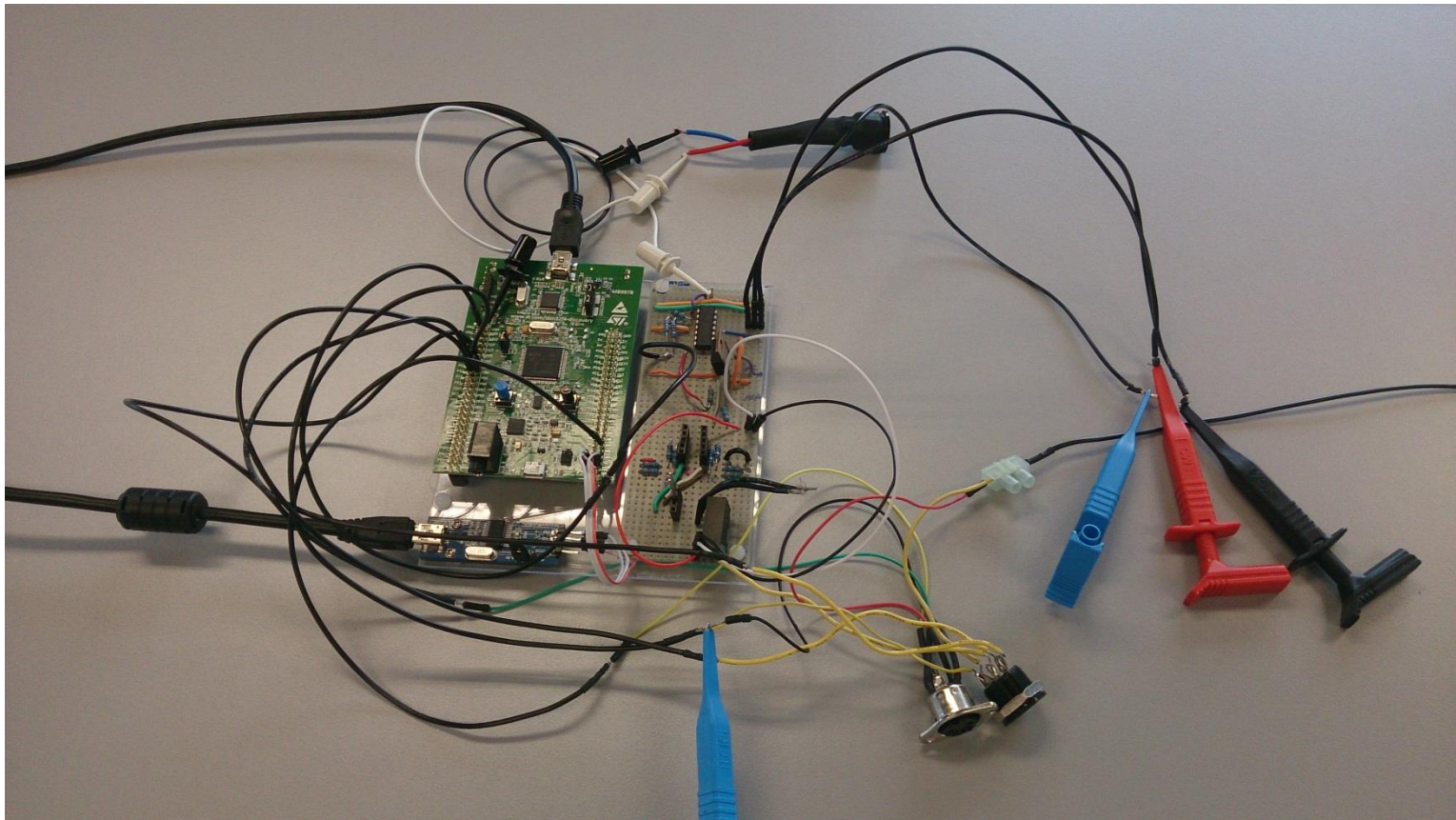


# STC measurements



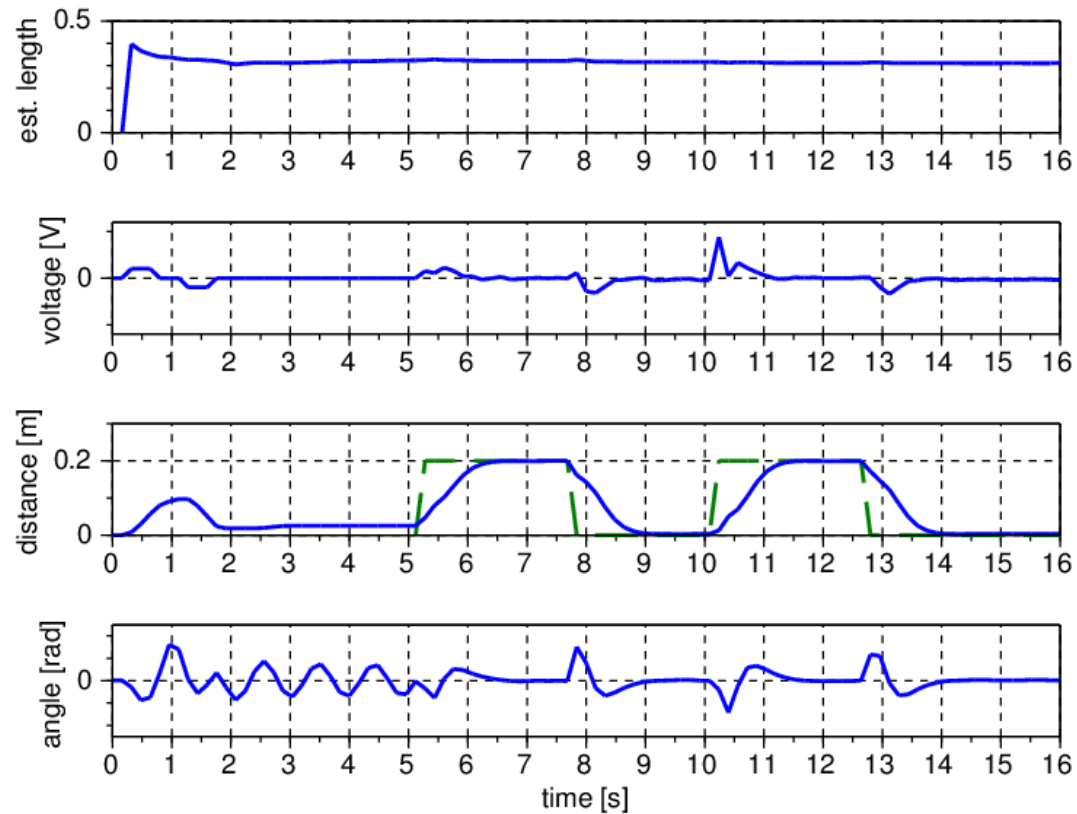
# STC measurements

## Discovery-board



# STC measurements

cart with pendulum – self tuning state control (STC)



# Summary and outlook

- Complete free (or low cost if hardware is included) tool-chain based on Scilab/XCos
  
- Ongoing development is targeted towards
  - efficient handling of vectorized signal lines in X2C
  - more block libraries
  - Industrial targets
  - adaption of the FMI (functional mockup interface) for model exchange

**Thank you for your attention!**

# References

- [1] JJE. Slotine, W. Li, Applied nonlinear control, Prantice-Hall, 1991
- [2] X2C in Scilab/XCos, 2013, <http://www.mechatronic-simulation.org>
- [3] Roberto Bucher, et al., RTAI-Lab tutorial: Scilab, Comedi and real-time control, 2006
- [4] Ana-Elena Rugina, et al., Gene-Auto: Automatic Software Code Generation for Real-Time Embedded Systems, DASIA 2008
- [5] Scicos-FLEX code generator, <http://erika.tuxfamily.org/drupal/scilabscicos.html>
- [6] Project-P code generator, <http://www.open-do.org/projects/p/>