Müller, Marcus; Fengler, Wolfgang; Amthor, Arvid; Ament, Christoph:

Model-driven development and multiprocessor implementation of a dynamic control algorithm for nanopositioning and nanomeasuring machines

URN: urn:nbn:de:gbv:ilm1-2014210283

Published OpenAccess: November 2014

Original published in:

Proceedings of the Institution of Mechanical Engineers / I. - London : Sage Publ (ISSN 2041-3041). - 223 (2009) 3, S. 417-429.
DOI: 10.1243/09596518JSCE673
URL: http://dx.doi.org/10.1243/09596518JSCE673
[Visited: 2014-10-14]

Model-driven development and multiprocessor implementation of a dynamic control algorithm for nanopositioning and nanomeasuring machines

M Müller, W Fengler, A Amthor and C Ament

Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 2009 223: 417
DOI: 10.1243/09596518JSCE673

Model-driven development and multiprocessor implementation of a dynamic control algorithm for nanopositioning and nanomeasuring machines

M Müller[1]*, **W Fengler**[1], **A Amthor**[2], and **C Ament**[2]
[1]Computer Architecture Group, Technische Universität Ilmenau, Ilmenau, Germany
[2]System Analysis Group, Technische Universität Ilmenau, Ilmenau, Germany

The manuscript was received on 21 August 2008 and was accepted after revision for publication on 12 December 2008.

DOI: 10.1243/09596518JSCE673

Abstract: This article presents a computationally intensive adaptive trajectory tracking control algorithm for dynamic control of nanopositioning and nanomeasuring machines. To realize the required high sample rate of the control algorithm, an embedded multiprocessor architecture has been chosen as development target. The model-oriented development approach studied here aims to narrow the gap between the control system design environment MATLAB/Simulink® and the actual distributed implementation on the custom platform by introducing a custom code generation target intending the utilization of automatic code generation facilities.

Keywords: control design, trajectory tracking, model-driven development, Simulink, multiprocessor implementation, code generation

1 INTRODUCTION

To measure and manipulate structures on the nanometre scale, high-resolution positioning stages are used, which are able to position a pattern in all three dimensions with a stationary accuracy below 1 nm. Currently the operating ranges of such stages are increased up to several hundred millimetres. To realize motion over long distances in vacuum, commonly ball bearing guides are utilized to bear the three motion axes. The position of the stage has to be controlled in all axes owing to external disturbances such as sound waves, thermal expansion of the mechanical components, ground motion, etc. Modified linear controllers as described in references [**1**] and [**2**] are state of the art. To perform well in nanopositioning, most of these control laws rely on high stiffness in combination with a stationary measurement strategy, but, with increasing operating range, this strategy becomes infeasible owing to the proportionally rising measurement time. Therefore, a dynamic measurement strategy is desirable. The control system has to be redesigned, since the dynamic behaviour of the classical controllers is wholly insufficient. To minimize the dynamic control error, the control system has to consider explicitly the dominant disturbance while moving, i.e. the friction that is introduced by the used ball bearing guides. Applying a nearly linear control law to such a system leads to tracking errors, limit cycles, and stick-slip motion [**3**]. In order to achieve high-precision dynamic positioning over wide velocity ranges, adaptive compensation of these non-linear effects is essential.

Since the combined goals of high precision and fast operation can be achieved only by using advanced control algorithms, a considerable computational load is produced that has to be coped with by the control system processing hardware within hard real-time restrictions. To deliver high performance for a specific application close to the controlled process, embedded computers are developed as specialized combined hardware–software systems. One of the major challenges in embedded system development stems from the fact that hard-

Corresponding author: Computer Architecture Group, Technische Universität Ilmenau, Helmholtz-Platz 1, Ilmenau, Germany. email: marcus.mueller@tu-ilmenau.de

ware properties already influence the design in the early stages, although the actual hardware-oriented implementation is one of the last stages of the development process. Model-driven development aims to describe a system independently of execution, interaction semantics, or hardware-specific implementation choices [**4**]. When using a model-driven approach to designing a real-time system that is to be realized as software targeted for an embedded platform, representations of hardware specifics have to be included in the modelling environment. State-of-the-art modelling environments such as MATLAB/Simulink offer certain target support libraries and means to generate target-specific code, but this mechanism is limited when custom, especially multiprocessor, hardware is considered. This article presents a case study of model-driven control algorithm development and hardware–software distribution and implementation on a custom embedded multiprocessor platform, and illustrates the efforts undertaken.

2 CONTROL ALGORITHM

To achieve the control engineering goals formulated in the introduction, the complex trajectory tracking control depicted in Fig. 1 was developed. The advantage of such a control scheme is its potential to speed up the dynamic behaviour of the controlled system, enabling the system to follow highly dynamic setpoints. The trajectory tracking control comprises four components. The dynamic setpoints for position, velocity, acceleration, and jerk are generated by a trajectory generation algorithm, which accounts for kinematic constraints of the experimental set-up. The friction compensation is realized by a modified Kalman filter as disturbance observer. As feedback controller, a PI state-space controller with an additional reference variable input is utilized. On account of the fact that the state velocity is immeasurable, a derivation algorithm was

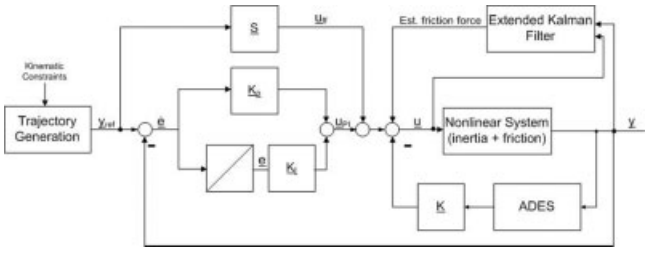

Fig. 1 Schematic diagram of the friction compensating control system

used in order to reconstruct the velocity out of the position signal.

2.1 Kalman filter

Neglecting the friction force, the dynamic behaviour of a single axis can be described on the basis of Newton's second axiom. Since the motor force has a linear relationship with respect to the applied current, the control algorithm controls the position via the current $i(t)$. The dynamic behaviour of the amplifier can be neglected, because its cut-off frequency is much higher than the sampling rate ($f_g > 10\,\text{kHz}$). Hence, the force/current relation is simply modelled as a gain k_A using the motor parameters provided by the manufacturer

$$k_A \cdot i = m \cdot a \tag{1}$$

where m is the mass and a is the resulting acceleration of the slider. In state-space notation, the system can be expressed as

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_A}{m} \end{bmatrix} \cdot i, \qquad y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{2}$$

where $x(t)$ is the position and $\dot{x}$ is the velocity of the slider. In order to design a non-model-based friction estimator, a Kalman filter approach is utilized. Considering an estimated friction force term, equation (1) is extended to

$$k_A \cdot i = m \cdot a(t) + \hat{F} \tag{3}$$

where $\hat{F}$ is the (immeasurable) friction force that resists the excited motion. After transforming the system into state-space notation, choosing x, v, $\hat{F}$, and $\dot{\hat{F}}$ as states and a discretization using a zero-order hold with the sample time T lead to the internal model of the friction observer

$$\begin{bmatrix} x_k \\ \dot{x}_k \\ \hat{F}_k \\ \dot{\hat{F}}_k \end{bmatrix} = \begin{bmatrix} 1 & T & -\frac{T^2}{2m} & -\frac{T^3}{6m} \\ 0 & 1 & -\frac{T}{m} & -\frac{T^2}{2m} \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \\ \hat{F}_{k-1} \\ \dot{\hat{F}}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{T^2}{2m} \\ \frac{k_A \cdot T}{m} \\ 0 \\ 0 \end{bmatrix} \cdot i(t)$$

$$x_k = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ \dot{x}_k \\ \hat{F}_k \\ \dot{\hat{F}}_k \end{bmatrix} \tag{4}$$

In the approach presented, the friction force is treated as an unknown state element. By measuring motion along with the applied force, it is possible to estimate the external friction force using the algorithm proposed by Kalman [**5**]. This method of friction cancellation has already been proven in other applications, e.g. by Ramasubramaniam and Ray [**6**, **7**].

2.2 PI state-space controller

According to Fig. 1, another main component of the proposed control system is the PI state-space controller. The controller is able to calculate the controller gains online in order to realize a gain scheduling strategy. As a system model, a moving mass with friction is considered. In equation (5), friction is assumed as a linear velocity-dependent force that resists the exciting force

$$k_A \cdot i - F_R(v) = m \cdot a \tag{5}$$

where m is the mass of the accelerated system, $k_A \cdot i$ is the applied force, x is the displacement, and $F_R(v) = F_C \cdot v$ is the (immeasurable) friction force. Choosing the position x and the velocity v as states leads to the following system model in state-space notation

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{F_C}{m} \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_A}{m} \end{bmatrix} \cdot i$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{6}$$

Using the present system model leads to the following dynamic matrix controlled system

$$\tilde{\mathbf{A}}_R = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{k_A(k_1+K_P)}{m} & -\frac{F_C}{m} - \frac{k_A k_2}{m} & \frac{k_A K_I}{m} \\ -1 & 0 & 0 \end{bmatrix} \tag{7}$$

Now the PI state-space controller can be designed with a pole placement strategy. The eigenvalues of the dynamic matrix are defined by

$$\det\left(s \cdot \boldsymbol{I} - \tilde{\mathbf{A}}_R\right) = \prod_{i=1}^{n} (s - \lambda_i) \tag{8}$$

Experimental study showed that two imaginary eigenvalues and one real eigenvalue are advantageous in the considered system. Comparing coefficients gives the following equations which determine all controller gains and the preliminary filter which is defined by $\boldsymbol{S} = [K_{ff0}\ K_{ff1}\ K_{ff2}]$

$$\det\left(s \cdot \boldsymbol{I} - \tilde{\boldsymbol{A}}_R\right) = \prod_{i=1}^{n} (s - p_0) \cdot (s - (p_r + p_i i)) \cdot (s - (p_r - p_i i))$$
$$k_2 = -\frac{-F_C + m(p_0 + 2p_r)}{k_A}$$
$$K_P = k_1 = K_{ff0} = \frac{(2p_0 p_r + p_r^2 + p_i^2)m}{2k_A}$$
$$K_{ff1} = \frac{F_C}{k_A} + k_2 \qquad K_{ff2} = \frac{m}{k_A} \tag{9}$$

For further information on the controller design, see reference [**8**]. As can be easily seen, all equations in system (9) have to be calculated online.

2.3 Algebraic derivate estimation scheme

The realized PI state-space controller needs the velocity of the system as input. On account of the fact that the velocity is not measurable with the utilized laser interferometers, the velocity signal has to be reconstructed out of the position signal. To realize this, an algebraic derivate estimation scheme (ADES) is used to determine a velocity signal with minimal noise and phase shift. In the following, a short overview of the algorithm is presented. For further information, see reference [**9**]. The measured noisy position signal $y(t)$ is approximated with a Taylor series of order N

$$y(t) \approx y_N(t) = \sum_{\kappa_0 = 0}^{N} \alpha_{\kappa_0} \frac{t^{\kappa_0}}{\kappa_0!} \tag{10}$$

The coefficients of the Taylor series α_{κ_0} correspond to the derivations of $y(t)$

$$\alpha_{\kappa_0} = \frac{\mathrm{d}^{\kappa_0} y}{\mathrm{d}t^{\kappa_0}}\Big|_{t=0} \triangleq y^{(\kappa_0)} \tag{11}$$

After a transformation into the complex variable domain, an arbitrary derivation of the position signal $y(t)$ can be calculated from the displacement history in discrete time. Including the sample time T_S yields

$$y^{(\beta)}(t) = \sum_{k=1}^{M} \Pi_{\beta,k} y(t - (k-1)T_S) \tag{12}$$

Equation (12) has to be calculated online. The weights $\Pi_{\beta,k}$ can be determined offline using the equation

$$\Pi_{\beta,k}=\frac{(N+\beta+\upsilon+1)!(N+1)!}{(MT_S)^j}\sum_{\kappa_1=0}^{N-\beta}\sum_{\kappa_2=0}^{\beta}\sum_{\kappa_0=0}^{\upsilon+\kappa_1+\kappa_2}\frac{(-1)^{N+\kappa_0-\kappa_1-\kappa_2}\left((k/M)^{N+\kappa_0-\kappa_1-\kappa_2+1}-((k-1)/M)^{N+\kappa_0-\kappa_1-\kappa_2+1}\right)}{\kappa_1!\kappa_2!\kappa_0!(N-\beta-\kappa_1)!(\beta-\kappa_2)!(N-\kappa_1-\kappa_2)!(\upsilon-\kappa_0+\kappa_1+\kappa_2)!(N-\kappa_1+1)(N+\kappa_0-\kappa_1-\kappa_2+1)} \tag{13}$$

where β is the order of the derivative, M defines the length of the displacement history in discrete time, and υ is a smoothing parameter. The derivation of these filter coefficients is beyond the scope of this paper; for further information, see reference [**10**]. The capability of ADES to determine a low-noise velocity signal out of the position signal is shown in Fig. 2. It is clear that the first derivative determined by ADES has nearly no phase shift compared with the derivative calculated with a TD1 transfer function. Also, the noise rejection of ADES is superior.

2.4 Trajectory generation algorithm

Another integral part of the proposed trajectory tracking control is the trajectory generation algorithm. This component generates a setpoint trajectory for position, velocity, acceleration, and jerk of the considered moving system. In order to maximize the precision, this algorithm contributes the mechanical and electrical constraints of the experimental set-up. Inputs for the algorithm are the start point, the endpoint, and the appropriate kinematic constraints.

To guarantee a continuously differentiable behaviour in all generated derivates, the jerk is a cubic function of time with three zero points (see equation (14)). Consequently, the acceleration is the primitive of the jerk (see equation (15)), the velocity is the primitive of the acceleration (see equation (16)), and the position is the primitive of the velocity (see equation (17))

$$j=\begin{cases}\frac{h}{2}t(2t-T_1)(t-T_1) & \text{for } 0\leqslant t<T_1\\ 0 & \text{for } T_1\leqslant t<T_1+T_2\\ -\frac{h}{2}(t-2T_1-T_2)(t-T_1-T_2)(2t-3T_1-2T_2) & \text{for } T_1+T_2\leqslant t<2T_1+T_2\end{cases} \tag{14}$$

$$a=\begin{cases}\frac{h}{4}t^2(t-T_1)^2 & \text{for } 0\leqslant t<T_1\\ 0 & \text{for } T_1\leqslant t<T_1+T_2\\ -\frac{h}{4}(t-2T_1-T_2)^2(t-T_1-T_2)^2 & \text{for } T_1+T_2\leqslant t<2T_1+T_2\end{cases} \tag{15}$$

$$v=\begin{cases}\frac{h}{120}t^3\left(6t^2-15T_1t+10{T_1}^2\right) & \text{for } 0\leqslant t<T_1\\ \frac{h{T_1}^5}{120} & \text{for } T_1\leqslant t<T_1+T_2\\ -\frac{h}{120}(t-2T_1-T_2)^2\left(6t^2-9T_1t-12T_2t+4{T_1}^2+9T_1T_2+6{T_2}^2\right) & \text{for } T_1+T_2\leqslant t<2T_1+T_2\end{cases} \tag{16}$$

$$x=\begin{cases}\frac{h}{120}t^4\left(2t^2-6T_1t+5{T_1}^2\right) & \text{for } 0\leqslant t<T_1\\ \frac{h{T_1}^5}{240}(2t-T_1) & \text{for } T_1\leqslant t<T_1+T_2\\ -\frac{h}{240}(2t^6-18T_1t^5-12T_2t^5+65{T_1}^2t^4+90T_1T_2t^4+30{T_2}^2t^4-120{T_1}^3t^3 & \\ -260{T_1}^2T_2t^3-180T_1{T_2}^2t^3-40{T_2}^3t^3+120{T_1}^4t^2+360{T_1}^3T_2t^2+390{T_1}^2{T_2}^2t^2 & \\ +180T_1{T_2}^3t^2+30{T_2}^4t^2-64{T_1}^5t-240{T_1}^4T_2t-360{T_1}^3{T_2}^2t-260{T_1}^2{T_2}^3t & \text{for } T_1+T_2\leqslant t<2T_1+T_2\\ -90T_1{T_2}^4t-12{T_2}^5t+14{T_1}^6+62{T_1}^5T_2+120{T_1}^4{T_2}^2+120{T_1}^3{T_2}^3+65{T_1}^2{T_2}^4 & \\ +18T_1{T_2}^5+2{T_2}^6 & \end{cases} \tag{17}$$

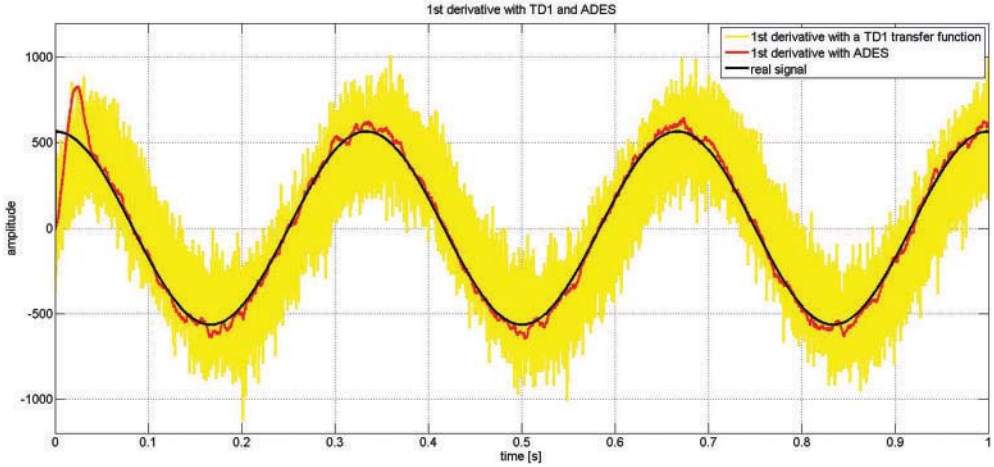

Fig. 2 First derivative calculated with a TD1 transfer function (fg = 50 Hz) and ADES

The whole motion can be divided into three intervals. In the first interval the system is accelerated to the maximum velocity, and in the third interval the system is slowed down to zero velocity. The jerk and acceleration functions in these two intervals are identical but mirror inverted. Between the acceleration and brake interval the system moves with maximum velocity. The whole kinematic behaviour of the movement is determined by the durations of the three intervals. Owing to the fact that all trajectories are symmetric, all interval durations could be described with two time parameters T_1 and T_2. At start-up the algorithm calculates T_1 and T_2 as well as the shape parameter h once. All parameters are determined such that all kinematic constraints are fully exhausted. Thus, the overall moving time is minimized. In online mode, equations (14) to (17) are carried out at every time step. For further information regarding the trajectory algorithm, see references [**11**] and [**12**].

2.5 Control system performance

The experimental set-up is a two-dimensional fine positioning stage (see Fig. 3). It was constructed by members of the Collaborative Research Centre 622 at the Technische Universität Ilmenau.

Each axis is driven by two ULIM3-2P-66 linear voice coil actuators of IDAM. The motors are powered by proprietary developed amplifiers, which provide current with the required precision. Commutation of the motors is achieved by the control system upon magnetic field intensity measurements provided online by Hall sensors. The operating range of this positioning stage is 200 × 200 mm^2. Each axis is supported by two R6-300-RF-SQ-HA linear guideways by Schneeberger. The position is measured by a laser interferometer of the SP-2000 type (manufactured by SIOS Messtechnik GmbH) with a resolution of less than 0.1 nm. A rapid prototyping system in combination with MATLAB/Simulink is utilized for data acquisition and control execution. All the described components of the trajectory tracking control were implemented as C S-functions in MATLAB/Simulink. The control algorithm uses a sampling rate of 6.25 kHz and operates on the amplifiers with a 16-bit resolution. For the present study, only the outer axis of the demonstrator is used, while the inner axis is mechanically jammed at the position shown in Fig. 3.

Figure 4 shows the reference trajectory for a linear motion (with return fare) of 10 000 000 nm of one axis of the demonstrator shown in Fig. 3. These trajectories are generated by the proposed trajectory generation algorithm. The used kinematic constraints

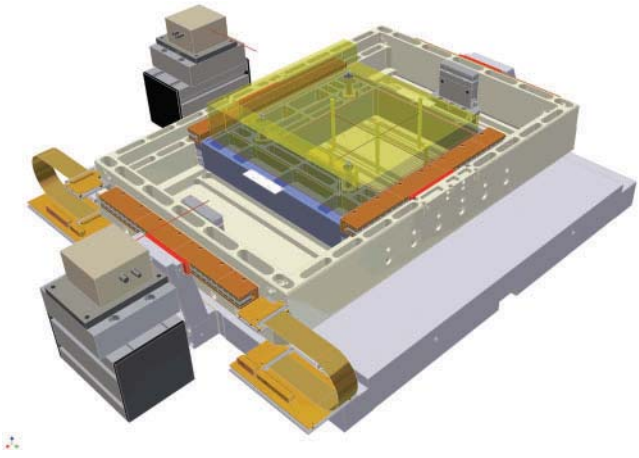

Fig. 3 Two-dimensional fine positioning stage

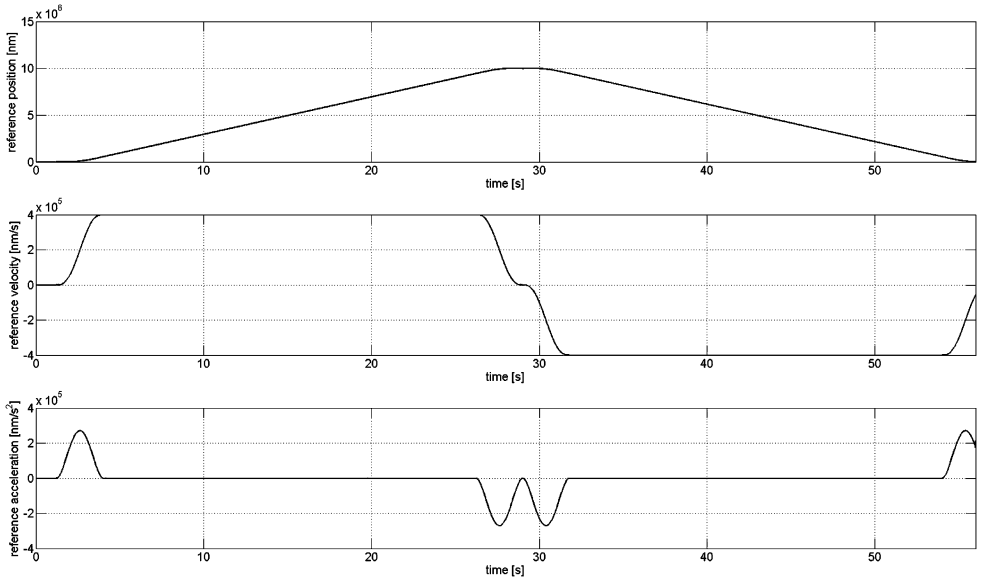

Fig. 4 (a) Reference position, (b) reference velocity, (c) reference acceleration

are $v_{max} = 400\,000\,\text{nm/s}$, $a_{max} = 2\,000\,000\,\text{nm/s}^2$, and $j_{max} = 300\,000\,\text{nm/s}^3$. In Fig. 5 the tracking error with and without trajectory tracking control is depicted. Using only a classical controller leads to a considerable tracking error of almost 2000 nm at peak. Using the trajectory tracking control, the error is reduced significantly. These results clearly show the capability of the proposed control system to enhance the performance of a position control system on the nanometre scale. A disadvantage of the trajectory tracking control is the high computational cost. To realize this control system for all three axes of a nanopositioning and nanomeasuring machine, a multiprocessor real-time system is essential.

3 MODEL-DRIVEN MULTIPROCESSOR IMPLEMENTATION

As indicated in the previous section, the algorithm was designed in MATLAB/Simulink, a platform widely used for control engineering. The goal of the approach presented here is to maintain Simulink as the main implementation platform. Code generation for a single Texas Instruments C6XXX DSP is supported using the Real Time Workshop® Embedded Coder™ and the toolbox Embedded IDE Link™ to Texas Instruments Code Composer Studio™. Nevertheless, the characteristics of the custom multiprocessor hardware exceed the means provided by the target libraries provided by Real

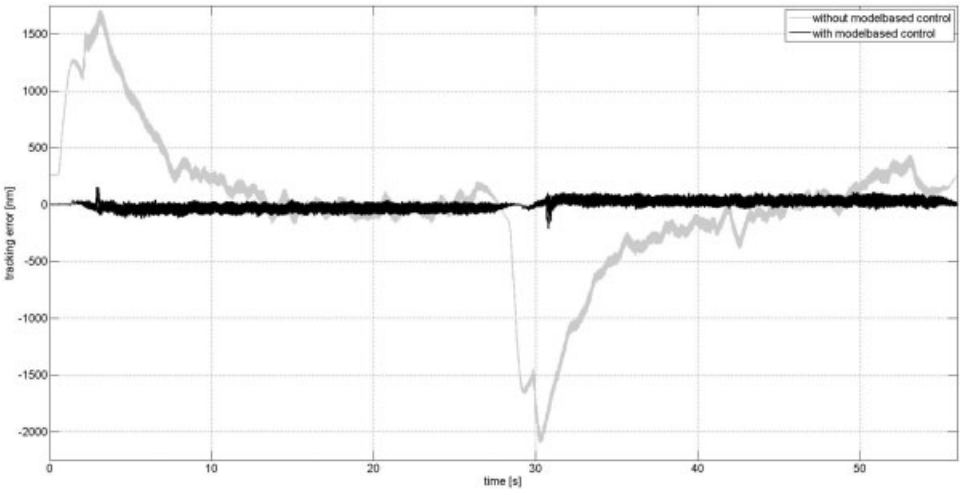

Fig. 5 Tracking performance with and without trajectory tracking control

Time Workshop. To utilize this hardware nonetheless, the Matlab-based implementation process has to be enhanced, first by means of optimal distribution of the algorithm on the given hardware, and second by elements introducing inherent platform characteristics into the Simulink model and the code generation process. Figure 6 illustrates the main steps of the process and the influence of model usage.

To capture the properties of both the given target hardware and the given algorithm model, an additional model has been built using the description language SysML, which is an extension of a subset of the unified modelling language (UML) designed for system engineering [**13**]. It provides the means to relate structure and behaviour of both hardware and software. Referring to hardware, these are the number and type of processing nodes, the topology, and the communication protocol. Referring to software, functional blocks with data dependencies are concerned. Further important properties are specifications and constraints, the most important of which in this case are processor speed, run times, and deadlines, according to which the validity of a hardware/software mapping and scheduling solution is decided. Owing to run time performance and the limited scope of the embedded implementation of control algorithms, only offline software partitioning and scheduling are conducted here as part of the development process. This produces a static distribution of functional units between hardware nodes and defines the respective execution intervals. To utilize this distribution information in the Simulink model, certain target specific blocks have to be included that allow the association of software to hardware partitions. These blocks are the accessible part of a platform abstraction that transparently integrates the specific hardware and communication characteristics into the Real Time Workshop code generation process. The following steps are fully supported by available tools. Running and profiling the code execution on the target system allows validation of distribution and scheduling and drives the optimization of the solution.

3.1 Hardware-specific characteristics

The hardware platform selected to study the deployment of MATLAB/Simulink models exemplified by the given algorithm is a multi-DSP unit that has been developed by the Computer Architecture Group of the Technische Universität Ilmenau. It is the result of the model-driven development approach presented in reference [**14**] and the past years' research effort in embedded system design regarding hard-

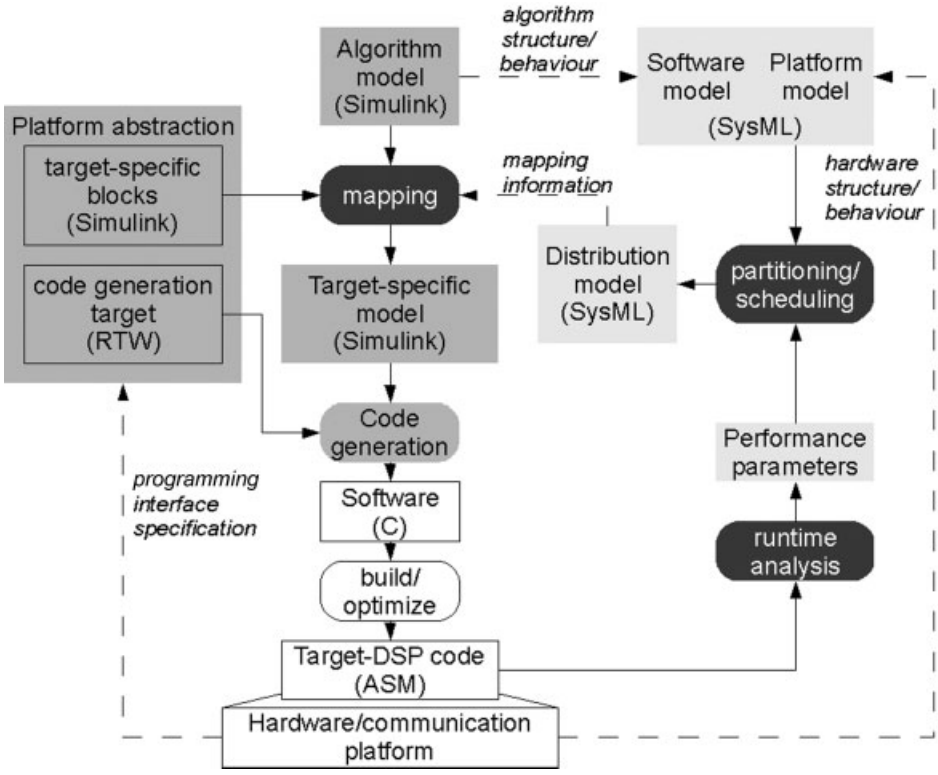

Fig. 6 Model-driven implementation process

ware–software codesign and synthesis with the background of high performance measure and control applications.

In its current version, it is a six-core multiprocessor board based upon off-the-shelf piggy-back modules carrying the TMS320C6713 digital signal processor (see references [**15**] and [**16**]). The module's main features utilized in this project are the asynchronous parallel bus interface, known as the external memory interface (EMIF), and the host port interface (HPI). The connection of EMIF to HPI is the key concept to interprocessor data communication in this system. The underlying hardware architecture, first presented in reference [**17**], is a master–slave system aligned in a bus structure with one master node and four slave nodes (see Fig. 7 for a picture of the hardware realization). The master node *Master* is the controller of the interprocessor bus communication with the slave nodes *Slave0* to *Slave3* over the parallel 16-bit wide HPI bus. As indicated in the picture, a sixth node can be added to the system, being connected to the master node in a similar EMIF-to-HPI fashion. This node, *Comm*, is to carry a USB2.0 interface board for connection to a host PC.

The hardware and communication structures have a major influence on how to organize the software distribution and scheduling. A parallel bus structure physically connects all slave HPIs to the master EMIF; logically, the slave HPI registers are mapped into distinct EMIF memory sections (see reference [**17**]). Address resolution allows accessing each slave selectively, but also broadcasting to all slaves collectively for time-efficient system-wide data distribution. The HPI accesses are actively conducted by the master node only, leaving the slave processor operation uninfluenced, and thus removing all communication concerns from the slave nodes. Signalling between master and slaves is realized by interrupts. The master node additionally controls the hardware interface to the I/O hardware backplane and has to perform the rate-monotonic tasks associated with data acquisition and process control at a current rate of 6.25 kHz, a requirement derived from the highly dynamic system behaviour [**1**, **2**]. In the given timeframe it has to communicate with the sensors and actuators and is additionally responsible for interprocessor data distribution by managing the data flow between the slave nodes, as well as for the real-time execution control of the distributed algorithms. The actual algorithm execution is conducted on the slave nodes. The HPI-based concept serves for the high-volume rate-monotonic communication that is required by the closed-loop control applications crucial for high-precision operation.

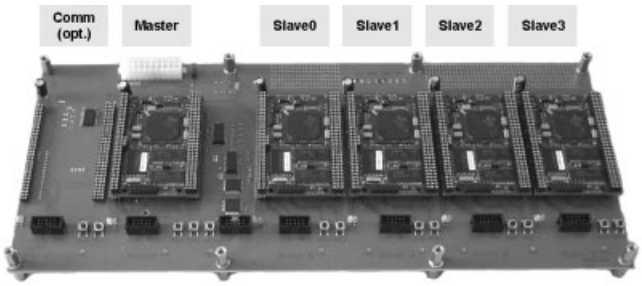

Fig. 7 Hardware with five DSP modules in master/slave structure

3.2 Controller algorithm model

The algorithm can be described using the SysML enhanced functional flow block diagram (see reference [**13**]) pictured in Fig. 8. The model shows the activities that have to be executed to perform one run of the control loop. Five major functional units can be distinguished, each capturing one of the components introduced in section 2. The data dependencies between the functions are indicated by the «*object flow*» edges. These dependencies induce the control and synchronization sequence, since the model semantics will allow the execution of an activity only if all input data are available.

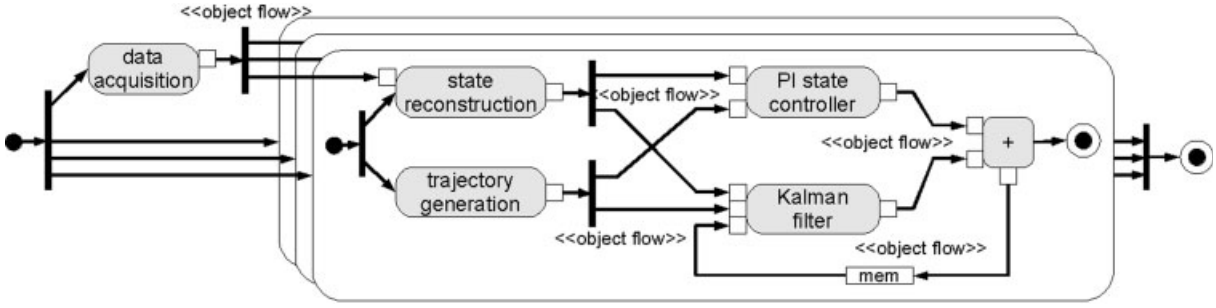

Fig. 8 Control algorithm in SysML notation

Profiling the execution of the algorithm's functional units on a target processor provided timing properties such as the HPI bus accesses for interprocessor data transfer. The timings that will be used to create a scheduling solution are shown in Table 1.

A strictly sequential execution of the algorithm system for three axes on a single C6713 DSP would involve the data acquisition followed by 3 times the control system functions, without HPI communication and interrupts, and would result in a loop time of at least 389 μs. This value indicates a maximum control sample rate of 2.57 kHz, which is far below the required operation rate of 6.25 kHz (see section 2).

Analysing the diagram and run time information reveals the points of interest for a coarse-grain software distribution (distribution on the function block level). Since the complete system works on three axes, the full system will inherit an obvious threefold parallelism. Also, a critical execution path emerges. This is the sequence of functions with the longest run times which have to be executed sequentially owing to dependencies. Disregarding communication costs, the critical path resembles the shortest possible execution time. Setting up concrete schedules will show how these concerns influence distribution and scheduling decisions.

3.3 Platform abstraction and code generation target

Platform abstraction provides a layer at which the characteristic features of the target hardware are accessible on the system modelling level while concealing the underlying complexity of the target-specific implementation. The abstraction layer presented here introduces Simulink model elements and an underlying code generation target.

While current multiprocessor targets for industrial computing hardware allow the parameterization of functional blocks with an ID for the core to be run on, this approach introduces a set of three blocks to map data flow properties to hardware units: a *source block*, which indicates a data flow source on a certain processing node, a *channel block*, which expresses a transfer to a new hardware partition, and a *sink block*, which concludes a data flow. These blocks are parameterized with the distribution information derived from the SysML scenarios.

The target blocks do not alter the functional behaviour of the algorithm, but act as markers for the code generation process. During generation, a coarse-grain data flow paradigm is implemented, in the context of which a data flow scheduler is created. This scheduler realizes the data transfers with HPI communication routines, synchronizes data, and triggers function blocks on slave nodes according to data availability. The scheduler is mapped to the master, whereas the function blocks are distributed among the slave nodes.

4 RESULTS

Figure 9 shows a diagram that provides means for illustrating distribution, execution intervals, data transfer delays, and interrupt signals. The hardware nodes are displayed at the top, each with an execution timeline. Function blocks are marked with the abbreviations from Table 1 and are aligned to the timeline of the node they are run on. The transfer delays are proportional to the number of values. Of the various scenarios researched, the two displayed in the figure show the most distinct features and provide the most different results.

The left-hand side of Fig. 9 shows the straightforward mapping of the algorithms for three axes on respective nodes. After data acquisition, the values for the position data are distributed and the calculations for each axis are started. Completing the sequential execution of the control algorithm, the slaves trigger the master to transfer back the results. Compared with the fully sequential single-processor execution of the three-axis control algorithm (see section 3.2), the direct mapping of one axis per processing node reduced the run time by 58.61 per cent to 161 μs. This results in a control sample rate of 6.21 kHz, which still misses the required operation rate.

On the right-hand side of Fig. 9, a scenario is shown that provides the optimal schedule for the given algorithm on the hardware. The DAQ function is mapped to the master via the required I/O interface. To avoid lengthening the rest of the critical

Table 1 Timing information for execution on target hardware

Data acquisition (DAQ)	Trajectory generation (TG)	State reconstruction (ADES)	PI state controller (PIC)	Kalman filter (KF)	Summation (sum)	HPI r/w access	Interrupt
41 μs	24 μs	52 μs	9 μs	30 μs	<1 μs	1 μs/64-bit	$\ll 1$ μs

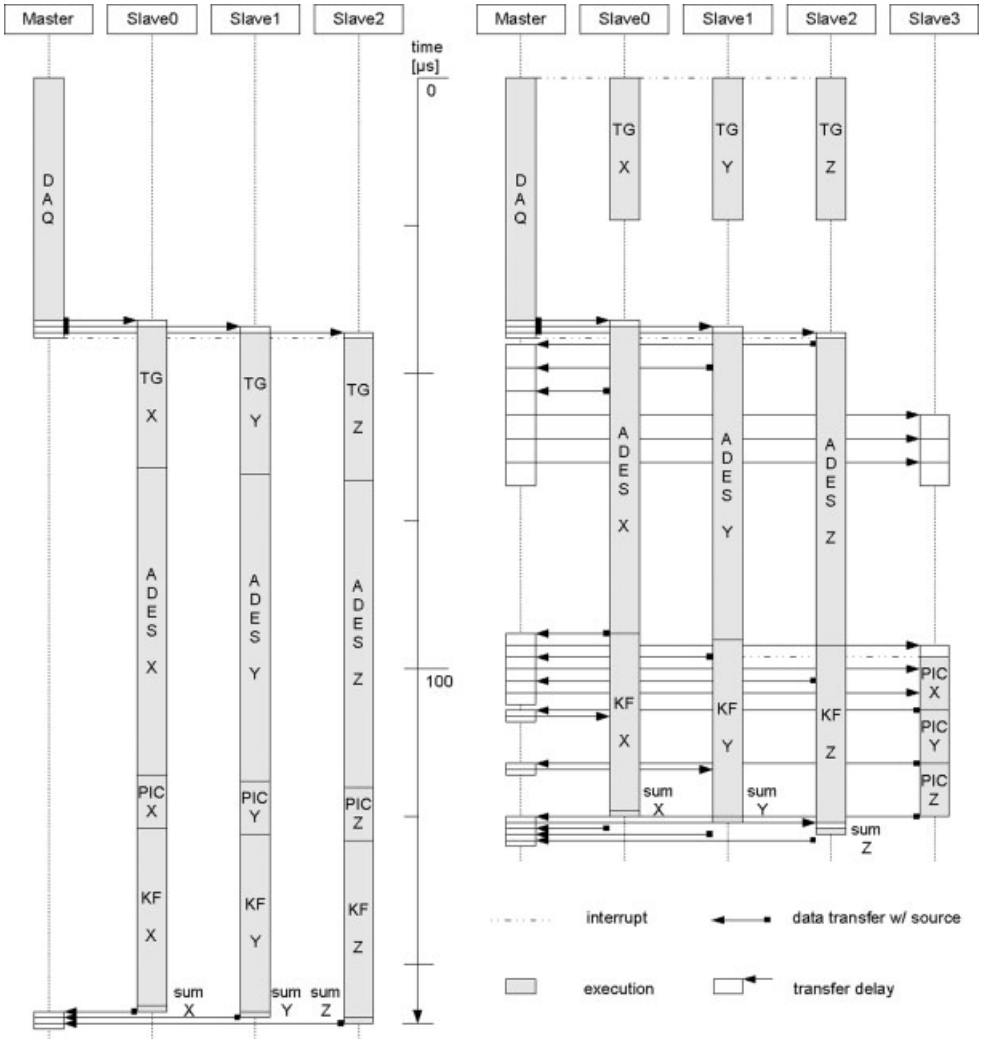

Fig. 9 Distribution and scheduling solutions

path described by the sequence ADES–KF–SUM by communication delays, it is mapped to one slave per axis. Since the trajectory generation is independent of input data, it can be executed simultaneously to data acquisition. After data distribution, the state reconstruction is started on slaves 0 to 2. The PI state controller and the Kalman filter depend on both trajectory data and state data, but can be executed simultaneously and independently from each other. Since the filter is part of the critical path, the PI state controller functions are remapped to the spare slave node. The communication effort to transfer the required data to master and the fourth slave can easily be recognized. Conveniently, the execution time of the ADES allows the distribution of the 3×4 trajectory values without inducing additional delay. Once the ADES modules are finished, the state values are distributed as well, again without interrupting the slaves, after which the individual PI state controller blocks can be executed. The Kalman filter run time masks these transfers, the execution of the PI controller instances, and the result transfers back to the respective slaves to execute the following summation. The result is stored on the respective slave as Kalman filter input for the following loop and copied back to the master for output. It is possible to transfer almost any data parallel to critical path execution on the slaves, inducing a communication overhead of 4.8 per cent relative to the critical path execution time. The overall run time of this scenario is 130 µs. With this optimal distribution and scheduling solution there is a reduction in the run time by 66.58 per cent compared with the sequential single-processor execution (see section 3.2). The calculated sample rate of 7.69 kHz could be achieved by fully exploiting the function block parallelism of the control algorithm, as well as parallelizing data transfers and execution.

Figure 10 shows an excerpt of a target-specific model of the second distribution scenario, taken

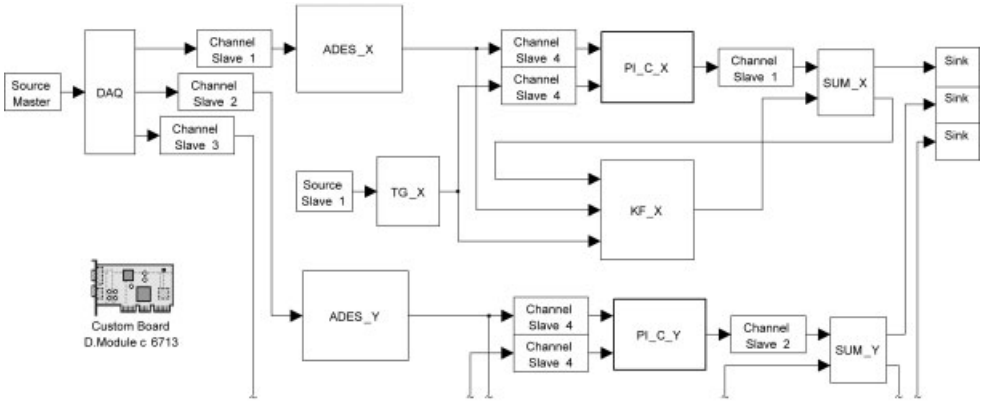

Fig. 10 Algorithm model with target-specific blocks in Simulink

from the Simulink environment. Only the modules for the *x* axis are depicted; the modules of the other axes are arranged downstream of the DAQ block outputs in a similar fashion. The first model element of interest is the *custom board*, a block that defines interface parameters for the target code generator. In this case it is customized to configure the DSP modules. Second, the source, channel and sink blocks can be recognized, several instances of which have been inserted into the data paths of the model, each parameterized with the ID of the node on which the downstream algorithm module is processed. This example shows to what extent algorithm engineers are required to redesign and enhance their model to fulfil the requirements of the code generation and hardware platform. The following steps of building a software project, compiling, and deployment (see Fig. 6) are conducted with full tool support.

5 CONCLUSION AND FUTURE WORK

In this study, a state-of-the-art control algorithm and an approach for the implementation on custom multiprocessor hardware have been presented. The emphasis was placed on the utilization of code generation facilities supported by the widely used design environment MATLAB/Simulink, thus providing a way to narrow the gap between model-driven hardware/software design and target implementation during control engineering.

The control algorithm system presented here has the capability to provide the qualities needed for nanometre accuracy during dynamic movement, as expressed by both the state reconstruction results and the achievable tracking error in comparison with respective classic approaches. Nevertheless, it has proven to be complex in computational terms. Exploiting the potential of the complete control algorithm system for coarse-grain parallelization, an approach towards multiprocessor application was chosen.

A custom Embedded Coder® target has been developed to move most of the hardware specifics into the background code generation process. The introduced set of three platform-specific model elements allows the mapping of function blocks and data transfers to hardware nodes without diverting the development focus from the functional view on the algorithm. Determining the optimal distribution strategy is solved in a different branch of the development process. It has been shown that, with SysML, an additional modelling formalism can be utilized firstly to capture the hardware and software characteristics influencing the design decisions and secondly to describe a hardware/software distribution and scheduling scenario – semantics exceeding the modelling features of MATLAB/Simulink. Additional effort in creating a multiprocessor-deployable model is clearly introduced during the setting up of an optimal distribution based on the timing constraints of hardware and software, but this optimization problem can be solved automatically. Future work will address an appropriate solution on the system design level that can be integrated or interfaced with a MATLAB/Simulink-based control design process.

When considering the computational capacity of multi-DSP hardware, it is foreseeable that it will not be able to fulfil the future combined requirements of shorter control cycles for even more elaborate control algorithms. Therefore, future hardware development will focus on heterogeneous FPGA-based architectures for better exploitation of the run time characteristics and internal parallelism of the algorithm components and building of more effective communication structures. Since alternate execution platforms pose different requirements and con-

straints, an alternate platform abstraction will be necessary to utilize the corresponding deployment tool chain.

The next step for the project presented here is to interface the hardware unit running the distributed control algorithm with the presented positioning stage to validate the achievable quality of the trajectory tracking control system for multiple axes.

ACKNOWLEDGEMENTS

This work is supported by Deutsche Forschungsgemeinschaft (German Research Council) under grant SFB 622.

REFERENCES

1 **Hausotte, T.** *Nanopositioning and nanomeasuring machine* (in German). Dissertation, Technische Universität Ilmenau, 2002.
2 **Hausotte, T., Jäger, G., Manske, E.,** and **Sawodny, O.** Control system of a nanopositioning and nanomeasuring machine. In Proceedings of the 9th International Conference on New Actuators, Bremen, Germany, 2005.
3 **Armstrong-Héouvry, B., Dupont, P.,** and **De, C.** A survey of models, analysis tools and compensation methods for control of machines with friction. *Automatica*, 1994, **30**(7), 1083–1138.
4 **Henzinger, T. A.** and **Sifakis, J.** The discipline of embedded systems design. *Computer*, 2007, **40**(10), 32–40.
5 **Kalman, R. E.** A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Engng*, 1960, **82**, pp. 35–45.
6 **Ramasubramaniam, A.** and **Ray, L. R.** Friction cancellation in flexible systems using extended Kalman–Bucy filtering. In Proceedings of the 2003 American Control Conference, Denver, Colorado, USA, 2003, pp. 1062–1067.
7 **Ramasubramaniam, A.** and **Ray, L. R.** Stability and performance analysis of non-model-based friction estimators. In Proceedings of Conference on *Decision and control*, Orlando, Florida, USA, 2001, pp. 2929–2935.
8 **Föllinger, O.** *Regelungstechnik,* 7th edition, 1992 (Huethig Verlag, Heidelberg).
9 **Reger, J., Sira-Ramirez, H. J.,** and **Fliess, M.** On non-asymptotic observation of nonlinear systems. In Proceedings of IEEE International Conference on *Decision and control*, Sevilla, Spain, 2005.
10 **Zehetner, J., Reger, J.,** and **Horn, M.** Realtime implementation of an algebraic derivative estimation scheme. *Automatisierungstechnik*, 2007, **55.**
11 **Pfeiffer, F.** and **Johanni, R.** A concept for manipulator trajectory planning. *IEEE J. Robotics Automation*, 1987, **3**(2), 115–123.
12 **Sawodny, O., Aschemann, H.,** and **Lahres, S.** An automated gantry crane as a large workspace robot. *Control Engng Practice*, 2002, **10**, 1323–1338.
13 Object Management Group. OMG Systems Modeling Language (OMG SysML^{TM}), V1.0. [Online]. Available from: http://www.omg.org/spec/SysML/1.0/PDF (accessed 17.11.2008).
14 **Fengler, W., Däne, B.,** and **Duridanova, V.** Design methodology for an embedded system for high-performance computing. In *Real-time programming 2003*, Proceedings of WRTP 2003, 14–17 May 2003, Lagow, Poland (Eds M. Colnaric, M. Adamski, and M. Wegrzyn), 2003, pp. 99–104 (Elsevier, Oxford).
15 D. Module. C6713 Board Revision 2.0 Technical Data Sheet Document Revision 1.1. D.SignT Digital Signalprocessing Technology GbR, Kerken, Germany, September 2004.
16 **Dahnoun, N.** *Digital signal processing implementation using the TMS320C6000 DSP platform*, 2000 (Prentice Hall, Old Jappan, New Jersey).
17 **Däne, B.** and **Berger, F.** A multiprocessor DSP system for a high throughput control application. EDERS-2004. The European DSP Education and Research Symposium, Birmingham, UK, 16 November 2004.

APPENDIX

Notation

a	acceleration
$\tilde{\mathbf{A}}_{\mathrm{R}}$	dynamic matrix of the controlled system
F_{C}	Coulomb force
$F_{\mathrm{R}}(v)$	friction force
$\hat{F}$	estimated friction force
$\hat{\boldsymbol{F}}$	deviation of the estimated friction force
h	shape parameter
i	applied current
$\boldsymbol{I}$	unit vector
j	jerk
k	control variable
k_A	motor constant of the voice coil motors
k_1	first state gain
k_2	second state gain
K_{ff0}	first gain of the preliminary filter
K_{ff1}	second gain of the preliminary filter
K_{ff2}	third gain of the preliminary filter
K_{I}	integrator gain
K_{P}	proportional gain
m	mass of the slider
M	lengths of the FIR filter
N	order of the Taylor series
s	Laplace variable
T	sample time

T_S	sample time	β	order of the derivative
v	velocity	κ_0	control variable
x	position	κ_1	control variable
$y^{(\beta)}$	deviation of order β of the original signal	κ_2	control variable
		λ	eigenvalue
		υ	smoothing parameter
α_{κ_0}	coefficients of the Taylor series	$\Pi_{\beta,k}$	discrete weights of the FIR filters