Suitability of Different Real-Time Solvers for a Model-Based Engineering Toolchain using Industrial Rexroth Controllers

Nils Menager[1] Rüdiger Kampfmann[1] Niklas Worschech[1] Lars Mikelsons[1]

[1]Bosch Rexroth AG, Lohr a. Main, Germany {nils.menager, fixed-term.ruediger.kampfmann, niklas.worschech, lars.mikelsons}@boschrexroth.de

Abstract

Due to the increasing complexity of technical systems, model-based engineering is getting more and more important during the development process of new products. The code generation from models and the usage of this code on hardware targets is one important feature of model-based development. To execute this code on the hardware device, a simulation runtime is additionally required, which offers numerical methods to solve the model equations. To use generated code on a controller, the simulation has to be executed in real-time, which is a huge requirement for the solver. In this work, a Modelica-based open source toolchain for model-based engineering with Rexroth controllers is presented, which is used for virtual commissioning of a typical hydro-mechanical system on a standard Rexroth PLC. Therefore, instead of parameterizing the controller directly on the real system, the control algorithm on the PLC is connected to the system model, which is additionally executed on the controller in parallel to the existing PLC application. Doing this, the commissioning times can be reduced significantly, as the commissioning process can already be started during the build-up of the system using a simulation model of the system. As hydro-mechanical systems are in general mathematically stiff, the choice of the solver for the system model equations is not arbitrary. In this work, five different real-time solvers, beginning with a simple explicit Euler through to more complex linearly implicit methods, are tested on a single hydraulic axis. Furthermore, typical issues like state events as well as algebraic loops are discussed in context of real-time simulation requirements.

Keywords: Real-time simulation, Modelica, Hardware-In-The-Loop, code generation, model-based engineering, real-time solver

1 Introduction

The increasing complexity of technical systems nowadays requires a change from conventional development methods towards model-based engineering. This means, that the entire development process through to the commissioning of the system is supported by models. A consistent application of this approach reduces time and costs, for example by shorter iterations and avoiding multiple implementations. An important component of model-based engineering is code generation, meaning the generation of code out of simulation and engineering tools.

The generated code can be used for different fields of application. One important field is Rapid Control Prototyping. During the development process of a new technical system, generally, a simulation model of the plant and the controller is set up inside a simulation environment. Later, during the commissioning of the system, the control algorithm has to be implemented on the hardware controller. Here, until now, the existing model is not used any further. Instead, the controller architecture is implemented from scratch inside the development environment of the controller, which leads to some serious drawbacks. First, a re-implementation of existing code always means extra time and costs, which are not necessary. Second, as the existing code is re-implemented in another language (mostly PLC programming languages), it cannot be guaranteed, that the newly implemented controller behaves in the same way as the previously designed controller inside the simulation environment. Of course, a re-implementation of code always means a potential error source. To avoid these disadvantages, it is desirable to use the already existing model also on the hardware controller. This can be realized by generating code from the controller model.

Besides the use of controller models directly as controller on a hardware PLC, there are several other fields of application to use simulation models on industrial controller hardware. One is the usage of simulation models in parallel to the control algorithm on the controller for system diagnosis. The difference between the simulated behavior and a measurement on the real system may imply different errors inside the real system.

Furthermore, it is possible to detect even upcoming errors, which can reduce downtimes of systems and hence

reduces costs. It is imaginable to use the simulation model also for the control of the system using modern control strategies like Model Predictive Control. Here, with help of a dynamic model, which is set up during the development process anyway, the future behavior of the system is precalculated. This precalculation, in combination of a measurement of the current system behavior, is then used to determine an optimal control input to the system for the next time step.

There are already possibilities to generate code from simulation models and to run this code on real-time targets, for example using a toolchain based on MATLAB/Simulink. For Bosch Rexroth, this toolchain has some serious disadvantages. One is the fact, that the former described toolchain causes high costs due to the licence fees. Standard Rexroth customers have often no possibility to buy the software, which means that this toolchain is not accessible for them. Furthermore, the code generation module of Simulink is a black box. It is not possible to modify the code generation, for example if already existing basic functions (e.g. from an application programming interface) of the controller should directly be integrated into the generated code. Another disadvantage is the release frequency of MATLAB/Simulink (in general two new releases per year). As it remains unclear, whether there were changes inside the code generation module, all existing models have to be tested again with every new release of MATLAB/Simulink.

To avoid the disadvantages of the Simulink toolchain, an alternative toolchain based on Modelica models has been developed. This toolchain allows the user to execute Modelica models directly on Rexroth control hardware. To run the models on the controller, a simulation runtime is additionally necessary. Bosch Rexroth developes an own simulation runtime, written in C++. The simulation runtime manages the simulation, includes the numerical methods to solve the equations, is responsible for the data handling during the simulation and handles occuring events.

As the models should, for example, be used to control systems on real hardware targets, it is mandatory to run the execution of the controller model in real-time. The real-time simulation of a model is a huge requirement for the solver, as most of the common numerical methods (implicit methods like CVode and Radau) to solve the occuring equations cannot be used anymore, as they contain iterative elements, which make the execution time non-predictable. Hence, in this work, it is investigated, which numerical methods are suitable to simulate hydro-mechanical systems, which are in general mathematically stiff, under consideration of real-time requirements. Therefore, five different numerical ODE solver are compared regarding the suitability and accuracy of the solution.

1.1 Outline of this paper

This paper is structured as follows. In the second section, the toolchain for model-based engineering using Modelica models is described. The third chapter deals with numerical methods for real-time simulation. In this chapter, a short mathematical background on the methods is given. In the fourth chapter, a virtual commissioning of a commonly occuring hydro-mechanical system (single axis system) is performed on a Bosch Rexroth XM22 industrial controller. Therefore, the simulation model of the system is executed in parallel to the controller code on the PLC. This is realized using the toolchain described before. It is investigated, which of the real-time solver presented in chapter 3 can be used to simulate the system properly. The fifth section summarizes the results of the application on the test example and rates the different solver regarding their suitability for real-time simulations on industrial hardware controllers. This contribution ends with an outlook on further investigations.

2 Toolchain for model-based engineering

As already described in the introduction, one main toolchain used for model-based engineering is based on MATLAB/Simulink for the code-generation. Customers of small and medium-sized enterprises have often no possibility to use MATLAB/Simulink due to high licence fees. Furthermore, a toolchain based on a commercial tool has the disadvantage, that the models are in general encapsulated inside this tool. For an integrated, model-based engineering it is necessary to exchange models with other tools. Therefore, a tool independent description language is essential. Hence, an alternative toolchain has been developed. The requirements on this toolchain are discussed in the following section, while the realization of this toolchain is described after that.

2.1 Requirements on the toolchain

Bosch Rexroth offers both controller for industrial (e.g. Rexroth IndraControl XM22) and mobile (e.g. BODAS RC controller) applications. Hence, one requirement for the toolchain is to support both controller types without modifications on the controller. Further requirements result from the disadvantages already discussed in the introduction. The code generation should be modifiable and offer the possibility to add existing functions to the generated code. Additionally, the toolchain should not be mainly based on commercial tools, but on open standards. This is necessary to avoid external dependencies. Last but not least, the toolchain should be easy to use, so that engineers can intuitively make use of it.

DOI 10.3384/ecp15118883

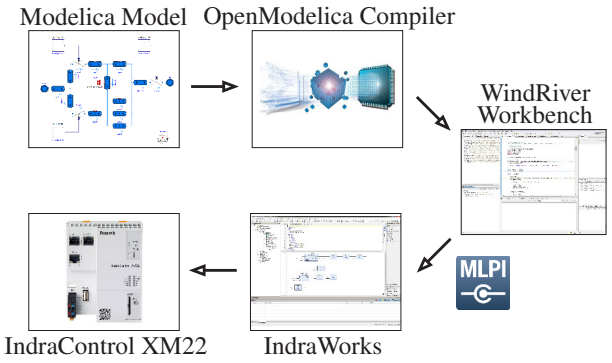

Figure 1. Structure of the Modelica-based toolchain

2.2 Realization of the toolchain

To avoid high costs and to allow an easy and uncomplicated re-use of models, the developed toolchain is based on Modelica models. Modelica is a modelling language, therefore, to generate executable code, a Modelica compiler is necessary. There exist both commercial (e.g. Dymola) and open source (e.g. OpenModelica) compiler. In this toolchain, of course, the open source OpenModelica compiler is used. One part of the compiler is the code generation. At this point, Bosch Rexroth has an own C++ code-generation. As this code generation module is self-written, it can easily be modified, for example if existing C/C++ libraries of the hardware should be used.

The generated C++-code from the code generation contains only the model. In order to execute this model, a simulation runtime is needed. The simulation runtime contains the numerical methods to solve the model equations and manages the simulation. This simulation runtime is also developed at Bosch Rexroth. Hence, it directly supports the generated code from the OpenModelica compiler. Both parts, the generated C++ code of the model and the C++ code of the simulation runtime, have then to be compiled for the control target. Therefore, a second compiler is needed. Each controller type, the industrial and the mobile controller, has its own operating system. Thus, a hardware-specific compiler is necessary.

The industrial controller used in this contribution is a Rexroth IndraControl XM22. This hardware is equipped with an Intel Atom x86 processor (1300 MHz). It works with the real-time operating system VxWorks. To generate executable code for this OS, the Windriver Workbench compiler is used. Using the Windriver Workbench compiler, both, the generated model code and the simulation runtime are compiled into a library, which is then, using the functionality of the *Motion Logic Programming Interface* (Engels and Gabler (2012)), integrated into a PLC project. The MLPI is an in-house developed interface (available in different languages as C/C++, C#, LabView, Matlab) to access controller functionalities from outside. This includes for example reading/writing controller parameters and variables, starting and stopping applications, triggering tasks or executing motion commands. Additionally, MLPI can be used to link externally implemented code to a PLC function block. For setting up Rexroth industrial PLCs, *IndraWorks* as development platform is used. The function block has input and output variables, which allow the data exchange between the simulation model and the PLC program. The PLC project running on the controller may then contain different function blocks, some of them implemented in IEC 61131 code and some of them implemented in C/C++. The structure of the toolchain is shown in Figure 1.

The basis of the mobile controller is the TriCore chip. Executable code for this target can be generated using the HighTec TriCore compiler. It is necessary to use a C-API, which contains all the essential functions needed for e.g. creating tasks or apply programs to tasks. The C-API and both, the generated code from the model and the simulation runtime, are compiled into a *.hex*-file using the HighTec TriCore compiler. This file can then be flashed onto the device using the development platform *BODAS service*.

Note, that this toolchain fulfills all the requirements discussed before. Due to the usage of the open source OpenModelica compiler, the toolchain is less cost-intensive. Furthermore, it is fully compatible to the main controller types (industrial and mobile controller) available at Bosch Rexroth without any modifications on the hardware. Because of the own in-house developed code generation module and simulation runtime, needed changes, for example to use already existing external libraries, can easily be integrated. As the generated code is integrated into the existing development platforms of the controller, the engineer can keep on working in his familiar tool, e.g. in case of the industrial controller, the generated code is simply connected to a function block instead of programming the code in IEC 61131 languages. All features of the development platform, like diagnosis or visualization features, can be used in its entirety.

3 Introduction on numerical real-time solver

The main focus in the development of ODE solvers, which are suitable for industrial problems, was to reduce the average computation time, while maintaining accuracy and robustness. Therefore, the widely spread solvers like Dassl or Radau use techniques like adaptive step-size control, i.e. only using small stepsizes, when necessary, or updating the Jacobian only, when convergence fails. This yields robust as well as effective solvers.

Unfortunately, the requirements for a real-time solver are tremendously different. A solver of this kind has to guarantee that one timestep is always finished in limited time, i.e the real-time cycle. Thus always the worst case runtime has to be considered. If a small stepsize is required at a certain step, for example in order to maintain stability, this stepsize can be used everywhere, because the real-time solver has to provide enough computation time for the smallest required stepsize. With the same arguments the Jacobian, if needed in the algorithm, can be updated in every step. But there still remains a problem: the common, i.e. implicit, solvers also require solving of at least one nonlinear system. These systems require an iterative algorithm like Newton's method. Unfortunately, convergence in a certain number of iterations cannot be guaranteed.

Hence summing up all requirements, a real-time solver is a fixed step solver without nonlinear systems. Obviously, explicit Runge-Kutta methods fullfill these requirements. Thus the easiest deputy of that family, the Euler forward, is one of the most spread solvers for real-time simulation. Unfortunately, explicit methods come along with limited stability issues. This is especially a problem while dealing with stiff systems, for example hydro-mechanical problems. In section 4 it is investigated, whether they can deal with a hydraulic single axis. Therefore, the forward Euler and the classical 4th-order Runge Kutta are tested. Additionally highly stable methods for real-time simulation are needed, in order to handle stiff problems. Usually implicit Runge Kutta methods come along with good stability issues, but also with nonlinear systems (Cellier and Kofman (2006)).

Linearizing the Runge Kutta methods yields the family of Rosenbrock methods, also known as linear implicit Runge Kutta methods. They exhibit the same stability properties, while avoiding nonlinear systems. In the next subchapter a short overview of the methods used is given.

Further problems for real-time solvers are algebraic loops and events, because they can also require iterative algorithms. State events are discussed in the following sections. Nonlinear algebraic loops always require iterative algorithms like Newton's method. This means that a worst case runtime cannot be guaranteed anymore. Thus the models simulated in this contribution are free of this kind of problem.

3.1 Rosenbrock methods

For the model equation, given in state space form:

$$\frac{\partial y}{\partial t} = f(y,t) \qquad y(t_0) = y_0$$

$$f\colon \mathbb{R}^n \times [t_0,\infty] \to \mathbb{R}^n \quad t_0 \in \mathbb{R} \quad y_0 \in \mathbb{R}^n$$

The simplest deputy of the Rosenbrock family is the linear implicit Euler, which is defined as:

$$\left(\frac{1}{h}I - \frac{\partial f}{\partial y}\left(t_n,y_n\right)\right)u = f\left(t_n,y_n\right) + \frac{\partial f}{\partial t}\left(t_n,y_n\right)$$

$$y_{n+1} = y_n + u$$

For higher order methods more stages are required. Therefore the efficient implementation of (Hairer and Wanner (2002)) is used. One step is given by:

$$\left(\frac{1}{h\gamma_{ii}}I - \frac{\partial f}{\partial y}\left(t_n,y_n\right)\right)u_i = f\left(t_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} a_{ij}u_j\right)$$

$$+ \sum_{j=1}^{i-1}\frac{c_{ij}}{h}u_j + \gamma_i h\frac{\partial f}{\partial t}\left(t_n,y_n\right) \qquad i = 1,\ldots,s$$

$$y_{n+1} = y_n + \sum_{j=1}^{s} m_j u_j,$$

whereas $\gamma_{ij}, \alpha_i, c_{ij}, m_i$ are constants. Using special sets of constants, methods of different orders can be obtained. In this contribution, Rosenbrock methods requiring one, two and three function evaluations per step are considered. The Rosenbrock method with one function evaluation is the linear implicit Euler described before and has therefore order one. One additional function evaluation, together with the constants of ROS3P (Lang and Verwer (2001)) yields a method of order three, while a method of order four (ROS4L) can be obtained with three function evaluations and the constants of the L-stable method described in (Hairer and Wanner (2002)). All methods used here are A-stable, the latter is even L-stable. Note, that using these Rosenbrock methods, the number of necessary function evaluations is reduced by one, due to a smart choice of the constants (Hairer and Wanner (2002)).

The occuring Jacobian is numerically approximated using forward finite differences and is updated once a step. For some applications, using coloured Jacobians, the number of function evaluations can be reduced (Braun et al. (2012)).

3.2 State events

For each state event a corresponding zero function and a boolean condition variable are generated. When a certain state event occurs, the corresponding function changes its sign and the condition variable becomes true. Usually in offline simulation, iterative algorithms like the Bisection method are used to localize the zero crossing. For real-time applications this algorithms cannot be used due to their iterative components. Therefore, two simple

DOI 10.3384/ecp15118883

methods were implemented to localize the state events without iterative parts.

The first approach is probably the simplest event handling. A change of sign of the zero function is just recognized and the corresponding condition variable is set to true. No zero search algorithm is executed. Afterwards, the equations are evaluated repeatedly until a consistent status is reached. This means, that all condition variables do not change anymore. This procedure is known as event iteration and is required because the occurrence of one event might trigger another event. This approach corresponds to replacing all state events in the Modelica model with the *noEvent operator*. Hence, the accuracy of this method is strictly limited. Also two or more occuring zero crossings within one step cannot be handled. The big advantage of this approach is that there is only little additional effort and simple models with state events can be simulated.

The second approach used is based on linear interpolation. If a zero function changes its sign, the zero crossing is located with linear interpolation. Afterwards, by using linear interpolation, the states at the time of the zero crossing are computed and the corresponding condition is set to true. Then also the equations are evaluated repeatedly until a consistent status is reached as mentioned above. Because only a smaller step than the desired stepsize was executed, the simulation time is not synchronous to the controller clock anymore. Therefore the stepsize of the next step is increased. This procedure only requires one integration and interpolation per step, so that again not much additional effort is added for the event handling. As mentioned above, using the absolute time has to be avoided. In this case, the time relative to the last succesful step is sufficient. For some problems, especially when the zero functions show nearly linear behaviour, this method provides quite good results. Also multiple zero crossings in one step can be handled. One big problem of this approach is that the occurrence of events in one step after another may lead to the fact, that the simulation time and the real time cannot get synchronous anymore.

The main disadvantage of this two approaches is that, without iterative algorithms, no certain accuracy can be guaranteed for the location of the zero crossings. This yields not only bad simulation results but also can cause inconsistent switching, i.e the event iteration is not converging and has to be aborted after a certain number of steps. In this case, the simulation cannot be proceeded. Also only one event per integration step can be handled. This drawbacks have to be avoided at the model side. It is obvious that not all models can be simulated in real time due to their complexity. Therefore, the models have to be simplified for online simulation. Summing up, for applications which should run a long time, like control algorithms, state events have to be strictly avoided. However, for applications which only run a certain time, like the plant model for the virtual commissioning presented in this contribution, state events can be tolerated, as long as the models can be simulated during the required time.

4 Virtual commissioning of a single axis system on a Rexroth PLC

In this contribution, a virtual commissioning of a hydromechanical system is performed, outlining the advantages of model-based engineering methods. As system, a single hydraulic axis is considered. The Modelica representation of this system is shown in Figure 2. The model contains a differential cylinder, a valve to control the volume flow, the pressure supply and the tank, and sensors to measure the piston position and the pressures in chamber A and B, respectively, of the cylinder. As already mentioned in section 3, for this contribution, the components of the system model are simplified with respect to events and hence do not contain any friction. The cylinder model does consider end stops, but the cyclinder is only moved inside its feasible range anyway. As controller, a P controller with velocity feed forward and active damping, is used, which is also available inside the simulation environment. This motion controller has three inputs and one output. As input, the pressures in cylinder chambers A and B and the current piston position of the cylinder are required. The calculated output is the valve command value. Overall, there are four control parameter to choose. As desired aim of the control, a velocity profile for the cylinder piston is predefined. To avoid a re-implementation of the control algorithm on the PLC, the described toolchain is used to transfer the simulation model of the controller on the PLC. The integration into the PLC project is realized using a function block, which includes the required inputs and outputs, which are used to pass the signals of the machine to the control algorithm and get the command values from the controller.

Until now, following the standard product development process, the two phases *system design* and *commissioning* are handled independently and consecutively. Hence, the commissioning of the system, which means the testing and optimization of the controller code, which is executed in real time on industrial control hardware, can be started not until the entire system is built up and supplied with electricity. At this time, the system is already built inside the customers buildings and therefore ties up capital and space. If the controller code can be tested in parallel to the system design and before the real system is built up, the overall project time can be significantly reduced.

In order to test the controller code without having the

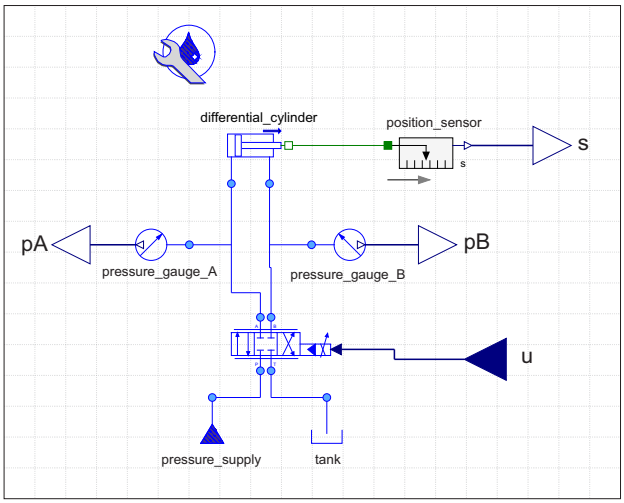

Figure 2. Simulation model of the single axis system

real system, the simulation model of the system is used to virtually commission the controller. There are several possibilities to perform the virtual commissioning. First, it is possible to run the simulation inside the simulation environment on a basic desktop computer. The hardware controller is then connected to the computer using a Hardware-In-The-Loop-setup. This approach has the disadvantage, that the simulation of the system is not executed in real time. Therefore, to synchronize the simulation and the hardware controller, the controller has to be slowed down to the simulation speed (Hofmann et al. (2015)). Hence, the real-time capability can not be verified using this strategy. A second possibility is to run the simulation on a special real-time hardware. With this approach, the real-time capability can be investigated, but additional hardware, which is only used for the commissioning, is necessary, which leads to high costs. This drawback can be avoided, if the PLC itself is used as real-time platform. As this hardware exists anyway to run the controller code, the simulation code of the plant, which is necessary for the virtual commissioning, can be executed in parallel on the same hardware. Thus, no additional hardware is needed in this case.

The simulation model of the system, which is available in Modelica, is also attached to a function block inside *IndraWorks* using the toolchain described in section 2.2. The three inputs of the controller are connected to the corresponding outputs of the function block containing the simulation model. In the same way, the output of the controller function block is connected to the input of the plant function block. To simulate the system behaviour properly, the calculation has to happen in real time. As the hydro-mechanical system is mathematically stiff, the used solver has to be chosen deliberately. The cycle time for the controller and the step size for the simulation is set to 1 ms.

4.1 Investigation on different solvers for the simulation

In a first step, before the virtual commissioning is performed, it is investigated, which solver is suitable to simulate the plant model and offers the best accuracy. Therefore, only the simulation of the system, without influences from the controller, is considered. This is necessary, because the controller can, under certain circumstances, compensate potential errors of the numerical method.

Hence, a special stimulus (sine with frequency $f = 0.5\,\text{Hz}$ and amplitude $\hat{y} = 2$) is applied to the input of the system, which is the input on the valve. The simulation results using the different solvers are finally compared to reference results. Reference results are obtained using the CVode solver in an offline simulation. Figure 3 shows the reference result for the output variable, the piston position of the cylinder.

Five different solver, the explicit Euler method, the explicit 4th order Runge-Kutta method as well as three different linear implicit Rosenbrock methods (order one, three and four), are used to simulate the single axis model.

4.1.1 Explicit methods

The Euler forward method is an explicit method and the easiest way to solve differential equations. Therefore, this numerical method is often used for real-time simulation. Because of the limited stability region (Cellier and Kofman (2006)), this method is not suitable for solving mathematically stiff systems like the hydro-mechanical single axis system. Using the forward Euler method in order to simulate, some variables attain physically nonsensical values (e.g. negative pressures). Hence, the simulation fails.

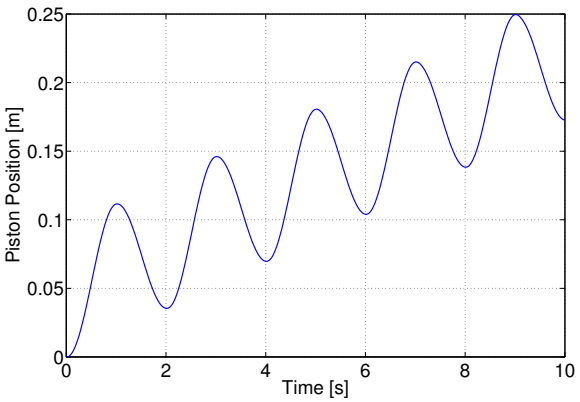

Figure 3. Reference result Piston Position generated with CVode

DOI 10.3384/ecp15118883

The explicit 4th order Runge Kutta method is also not suitable for stiff systems and shows the same behaviour as the forward Euler. Using this method, the simulation fails for the same reasons as mentioned above.

4.1.2 Rosenbrock methods

The Rosenbrock methods of order one, three and four can be used to simulate the system. In order to evaluate the accuracy of the method, the difference between the simulation result using each solver and the reference result is plotted. This difference can be seen in Figure 4. For the simulation, a step size of $h = 20\,\text{ms}$ is used.

4.2 Investigation on event handling using the example of a bouncing ball

The implemented event handling is tested in combination with the bouncing ball example. Both approaches described in chapter 3, with and without interpolation of the zero function, illustrate the physical effect, as soon as the ball hits the underground. But there are differences, when it comes to accuracy. The bouncing ball model is simulated for 0,75 s with a cycle time of 10 ms and the ROS4L. During this time, the ball hits the ground exactly once. Figure 5 shows the simulation results.

Without any interpolation, the event is detected 39 mm below the underground (red curve). When using a linear interpolation to determine the zero crossing more precisely, the penetration of the ball can be reduced to 10^{-9} mm (blue curve). Even though the good accuracy might result from the fact, that the zero function for this example does not differ much from a straight line, it can be seen, that the linear interpolation yields convenient results, at least for some problems.

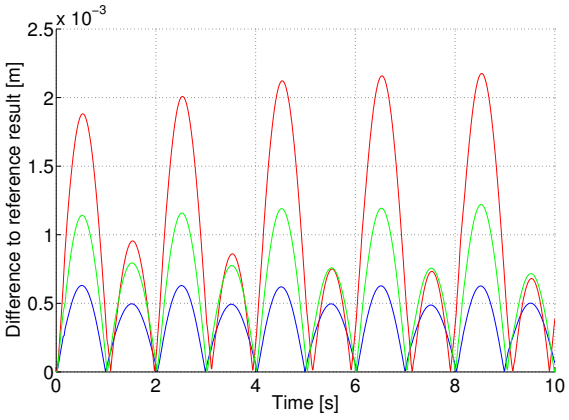

Figure 4. Difference between Rosenbrock methods and reference result [red: linear-implicit Euler; green: Rosenbrock method order 3 (ROS3P); blue: Rosenbrock method 4 (L-stable)]

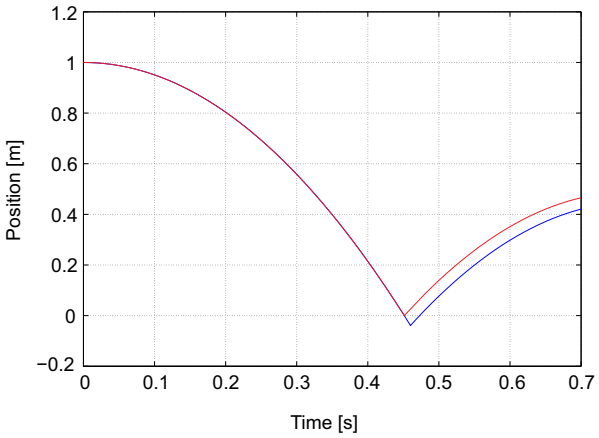

Figure 5. Bouncing Ball [red: with interpolation of zero function; blue: without interpolation of zero function]

In case, that more complex systems with multiple events are regarded, this simple state-event handling does not work. This applies for example for a complex stick-slip friction model. Hence, such effects are not included in the single axis model used in this contribution.

4.3 Practical application of the virtual commissioning

For the virtual commissioning, the stimulus described in section 4.1 is removed and the model is coupled with the controller on the PLC and then simulated in real time.

As the explicit methods fail to solve the plant model equations anyway, only the Rosenbrock methods are considered for the virtual commissioning. Within these methods, the main computational effort is not induced by the methods themselves, but through the function and Jacobian evaluations. Hence, in order to achieve the desired cycle time of 1 ms, the linear implicit Euler method is chosen for the virtual commissioning of the single axis, because it needs only one function and Jacobian evaluation per step. As shown in the passage above, this method yields the worst results, however the accuracy is still good enough for this problem.

Figure 6 shows the results of the virtual comission-ing. The actual position of the piston and the desired one show a very good agreement. So the parameterization of the controller is appropiate for this problem and the controller can be used on the real system. Using the method of virtual commissioning, the commissioning time can be reduced significantly. Even though the control algorithm is tested and parameterized after the virtual commissioning, a fine adjustment of the control parameters has to be performed, as soon as the control hardware is connected to the real machine. This is necessary, be-

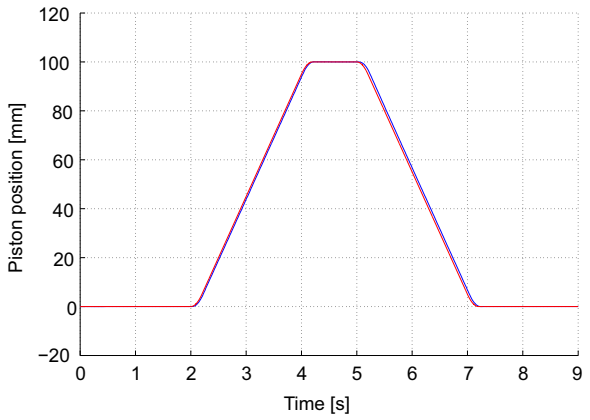

Figure 6. Comparison of the desired and actual behaviour after performing the virtual commissioning; used numerical method: Rosenbrock order 1

cause the behaviour of the virtual plant differs from the real system due to model inaccuracies.

5 Conclusion

In this work, a toolchain to run Modelica models on real hardware controller is presented. This toolchain allows the user to link both, the C/C++ code generated from the Modelica models and a simulation runtime to execute the simulation, to a PLC function block. Using this toolchain, it is possible to realize a virtual commissioning. The method of virtual commissioning allows the engineer to test and optimize the controller code before the real system exists. The controller on the PLC is therefore connected to the simulation model, which is running on the PLC, instead of the real system. The PLC acts in this case as both, a classical PLC for the control tasks and a real-time hardware target to simulate the virtual plant, at the same time. To run the simulation in real time, special numerical methods with real-time capability are necessary.

Therefore, different real-time solver were compared. In a first benchmark, the accuracy of the solver was analyzed on a mathematical stiff hydro-mechanical system (single axis system). It was shown, that explicit integration methods, such as the commonly used Euler forward and explicit Runge Kutta methods, fail to solve the occuring model equations and are therefore not usable for the simulation of plant models. The Rosenbrock methods presented in section 3.1, on the other hand, are in general suitable to solve stiff differential equations. As expected, the accuracy of the result depends on the order of the numerical method. The higher the order of the method, the smaller the error for a constant step size (see Figure 4).

When using simulation models on a hardware in real time, some issues have to be considered. While state events have to be strictly avoided, if the model, e.g. a control algorithm, is proposed to run within a real life application, they can be tolerated in models, which are only used for testing in a defined time range. This applies to the plant model used for the virtual commissioning.

6 Outlook

A different approach is, instead of designing a separate control algorithm like the P controller with velocity feed forward, to use the plant model directly for the control realizing a *Model Predictive Control*. Using this method, an optimal control problem has to be solved in every real-time cycle. The solution of the optimal control problem is equivalent to the optimal control input to the system in the next time step. The computation of the dynamic optimization problem is very time-consuming, which is a huge challenge.

Plant models, which contain numerous events, are still a problem for numerical real-time solvers. To improve the performance of the Rosenbrock methods in combination with state events is still an open task. As it is described, several state events like end stops (e.g. in the cylinder or the ground in the bouncing ball example) can be handled already now. Other events, especially deriving from friction, do not work properly today.

Especially if small cycle-times are required for the control, efficient code is essential to reduce simulation times. For the simulation of the models on the hardware target, the entire simulation runtime, which is also used for offline simulations, is utilized. This runtime contains features, like writing output files or dynamic state selection, which are not necessary for real-time simulations.

References

W. Braun, S. Gallardo-Yances, K. Link, and B. Bachmann. Fast simulation of fluid models with colored jacobians. In *Proceedings of the 9th Modelica Conference, Munich, Germany, Modelica Association*, 2012.

F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 2006.

E. Engels and T. Gabler. Universelle Programmierschnittstelle für Motion-Logic Systeme. In *Struktur, Funktionen und Anwendung in Forschung und Lehre, Tagungsband AALE*, 2012.

E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer, 2002.

A. Hofmann, S. Schweig, and L. Mikelsons. Virtuelle Inbetriebnahme mechatronischer Systeme unter Einbeziehung realer Industriesteuerungen von Bosch Rexroth. In *Tagungsband Mechatronik 2015, VDI Mechatroniktagung 2015 am 12.-13. März 2015 in Dortmund*, 2015.

J. Lang and J. Verwer. ROS3P - an accurate third-order Rosenbrock solver designed for parabolic problems. *BIT Numerical Mathematics*, 41(4):731–738, 2001.