Comparison of Programmed MATLAB Implementation and Graphically Modelled Simulink Implementation for ARGESIM Benchmark C11 ‘SCARA Robot’

Tamara Vobruba[1,*], Claudia Wytrzens[1], Andrea Kainz[1,2], Irene Hafner[3]

[1]Department of Analysis and Scientific Computing, Vienna University of Technology, Wiedner Hauptstraße 8-10, 1040 Vienna, Austria; **tamara.vobruba@tuwien.ac.at*

[2] AIT Austrian Institute of Technology, Tech Gate Vienna, Donau-City-Straße 1, 1220 Vienna, Austria

[3] dwh GmbH, Neustiftgasse 57-59, 1070 Vienna, Austria

Simulation Notes Europe SNE 24(3-4), 2014, 173 - 178
DOI: 10.11128/sne.24.bn11.10265
Received: February 15, 2014; Revised: October 3, 2014;
Accepted: October 20, 2014;

Abstract. This approach to ARGESIM Benchmark C11 ‘SCARA Robot’ compares a programmed MATLAB implementation with a graphically modelled Simulink implementation, esp. with respect to implicit models and state event descriptions.

The mechanical system results in an implicit second order differential equation. Therefore, different ways of solving the equation using MATLAB and Simulink are investigated. In this context, the question of using an explicit or an implicit model description arises. Based on the best fitting model description, a point-to-point motion of the robot controlled by a single axis PD-control is simulated. Furthermore, a collision avoidance maneuver of a defined obstacle is implemented in MATLAB as well as in Simulink. Finally, the two models are compared and the advantages and disadvantages of each model are pointed out.

Introduction

ARGESIM Benchmark C 11 [1] deals with motion of a three axis SCARA (Selective Compliance Assembly Robot Arm) robot. The equation of motion leads to an implicit second order system of differential equations for the joint vector q, where q_1 and q_2 are the joint angles and q_3 is the joint distance.

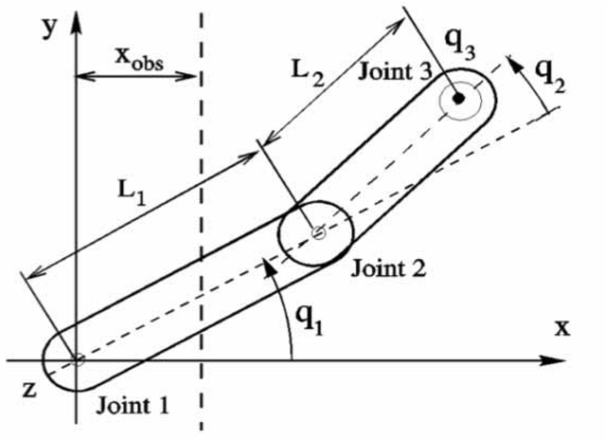

Figure 1: Schematic representation of the SCARA robot. [1]

The observed mechanical system is fully defined. The balance of power and the constraints result in a system of second order differential equations using variational calculus methods [3]. So the equation of motion can be written in a compact way as

$$\boldsymbol{M}\ddot{\boldsymbol{q}} = \boldsymbol{b}. \tag{1}$$

Here the Mass matrix M is a block-diagonal matrix as described in formula (2).

$$\begin{pmatrix} ma_{11} & ma_{12} & 0 \\ ma_{21} & ma_{22} & 0 \\ 0 & 0 & ma_{33} \end{pmatrix} \cdot \begin{pmatrix} \ddot{q}_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{2}$$

The elements of the Mass matrix $\boldsymbol{M}$ and the right-hand side $\boldsymbol{b}$ are determined as follows:

$$ma_{11} = \theta_1 + 2\theta_2 \cos(q_2) + \theta_3 \tag{3}$$
$$ma_{12} = ma_{21} = \theta_2 \cos(q_2) + \theta_3 \tag{4}$$
$$ma_{22} = \theta_3 \tag{5}$$
$$ma_{33} = m_{3L} + \theta_{3mot} u_3^2 \tag{6}$$

$$b_1 = T_1 + \theta_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2)\sin(q_2) \tag{7}$$
$$b_2 = T_2 - \theta_2 \dot{q}_1^2 \sin(q_2) \tag{8}$$
$$b_3 = T_3 - m_{3L} g\ , \tag{9}$$

where θ_i are the moments of inertia and T_i are the joint forces, whose calculation is based on the assumption that the two links are rods with homogenous mass distribution m_1 and m_2 along the length L_1 and L_2 of the rods.

The capabilities of MATLAB and Simulink to handle this system are explored. For all computations, MATLAB R2013b is used running on a Mac OS X Version 10.9.5.

1 Modelling Method

There are different tools in MATLAB and Simulink to solve systems of differential equations and therefore different methods of describing the system.

1.1 MATLAB

At first, the different methods of solving the model-equations numerically in MATLAB were investigated. The ODE-solver provided by MATLAB can only deal with first order differential equations [2], thus the three-dimensional second order system has to be transformed into a six-dimensional first order system.

$$\begin{pmatrix} ma_{11} & ma_{12} & 0 & 0 & 0 & 0 \\ ma_{21} & ma_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & ma_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix}. \tag{10}$$

The ODE-solver ode45 is able to deal with implicit and explicit differential equations [2]. In order to get an explicit model description, the mass matrix in system (2) is inverted. To prove the regularity of the matrix the determinant has to be examined. Per definition of each of the factors and addends it is a fact that

$$ma_{33} = m_{3L} + \theta_{3mot} u_3^2 \neq 0 \tag{11}$$
$$det\begin{pmatrix} ma_{11} & ma_{12} \\ ma_{21} & ma_{22} \end{pmatrix} = \theta_1\theta_3 - \theta_2^2 \cos(q_2) \neq 0. \tag{12}$$

So it is possible to invert the mass matrix in order to get an explicit system. The inversion of the matrix is calculated analytically and the explicit system is implemented in MATLAB.

The average runtimes of the explicit and the implicit model were compared and as a result – as illustrated in Table 1 - the explicit model is a bit faster and so it is used for the additional tasks.

System	runtimes
implicit (1st order)	4.7382s
explicit (1st order)	3.9431s

Table 1: Runtimes of the implicit and explicit model in MATLAB.

1.2 Simulink

In Simulink it is possible to implement second order differential equation – without transforming it into a first order differential equation – by using the 'Integrator'-block twice [2]. For this comparison the ode-solver ode45 is used in Simulink as well.

So the three dimensional square mass matrix in system (2) is inverted analytically and the resulting explicit second order system can be implemented in Simulink. It is notable that the Simulink model is a lot faster than both models in MATLAB.

System	runtime
explicit (2nd order)	0.2093s

Table 2: Runtime of the Simulink model.

2 Point-to-point Motion

Furthermore, the implementation of a point-to-point motion by the SCARA robot, controlled by a single axis PD-control, was performed. Therefore three more first order differential equations are added to the system of differential equations, which describe the electrical relationship of the armature of the robot servo motor,

$$\dot{I}_i = \frac{U_i - k_{T_i}u_i - R_i I_i}{L_i}, \qquad i = 1,2,3 \tag{13}$$

$$I_i \in [-I_{imax}, I_{imax}], \qquad i = 1,2,3 \tag{14}$$

where k_{T_i} is the motor constant, u_i is the gear ratio, R_i is the resistance and L_i is the inductance. The single-axis PD-control is involved to control the point-to-point motion of the robot by the control of the voltage U_i and represented by the equations

$$U_i = P_i(\hat{q}_i - q_i) - D_i\dot{q}_i, \qquad i = 1,2,3 \tag{15}$$

$$U_i \in [-U_{imax}, U_{imax}], \qquad i = 1,2,3. \tag{16}$$

Here $\hat{q}_i$ stands for the target point of the SCARA robot motion and therefore P_i describes the proportional gains and D_i derivative gains. The values of both of them are given for each controller.

2.1 Implementation in MATLAB

To implement the point-to-point performance of the SCARA robot in MATLAB the six-dimensional first order system – as described in Section 1.1 – has to be transferred into a nine-dimensional first order system by adding the three first order differential equations as given in formula (13).

Furthermore, in order to limit the values of $U_i$and I_i such that their values are located in the intervals given above, their numerical values are compared to the limits of the intervals by using logical expressions in MATLAB in the way that the expression

$$(U_i < -U_{imax}) \cdot (-U_{imax}) + (U_i > U_{imax}) \cdot U_{imax} + (-U_{imax} \le U_i \wedge U_i \ge U_{imax}) \cdot U_i \tag{17}$$

provides the right value for U_i, which is either $-U_{imax}$ if U_i is not located in the considered interval and less than $-U_{imax}$ or U_{imax} if U_i is larger than U_{imax} or U_i if it is situated in the interval.

In addition to this an event function is included to stop the integration at the target position using an accuracy of $acc = 0.001$ of reaching the target value.

2.2 Implementation in Simulink

The point-to-point motion of the robot is implemented in Simulink by adding the PD-control and the differential equation of the electrical current as given in formula (13) more or less as a system according to q which is running parallel to the system which describes q and its derivations. This parallel running system is shown in Figure 2.

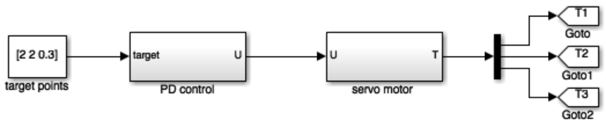

Figure 2: PD-control and servo motor subsystems for the implementation of the point-to-point motion in Simulink.

The outputs of the system are the values T_i, which are needed to calculate the right-hand side b_i of the dynamical equations.

Compared to the implementation in MATLAB, to control that the values of electrical voltage and current are located inside the intervals a ‘Saturation’- block is used, which proves to be a much simpler way.

Furthermore, for stopping the integration at the target position the same accuracy as in MATLB $acc = 0.001$ of reaching the target value is used. This is realised in Simulink by using a “Stop”- block.

2.3 Results

The point-to-point motion is – as already mentioned – implemented in MATLAB as well as in Simulink. Figure 3 shows the point-to-point motion of both models with initial value $q = [0,0,0]$ and target point $\hat{q} = [2,2,0.3]$.

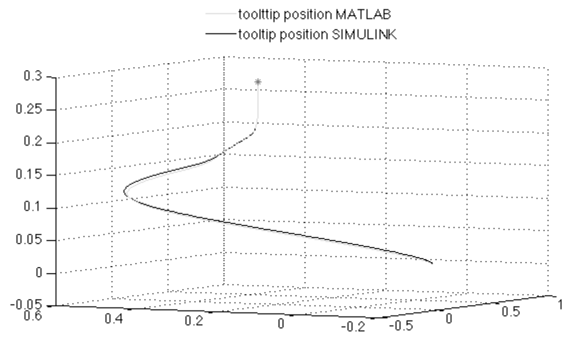

Figure 3: Comparison of the implementation of the point-to-point motion in MATLAB and Simulink.

Figure shows that the tooltip of the MATLAB model is following the same movement as the tooltip of the Simulink model. Both models are reching the target point, which is marked as a red dot in Figure 3. Furthermore, by comparing the simulation times (MATLAB: $t = 1.5719$, Simulink: $t = 1.5719$), which both models need to reach the target position, one can see that there is no significant difference. But if comparing the runtime one can observe that the Simulink simulation with about 0.2093 seconds is way faster than the runtime needed by MATLAB, which is about 3.9431 seconds. So for the simple point-to-point motion the Simulink model in terms of the simulation time and the runtime seems to be the better option as there is no difference in the simulation time but the runtime is a lot faster.

3 Obstacle Avoidance

In addition the model description is extended to simulate an avoidance manoeuvre of a given obstacle. The idea is to slow down the movement of the tooltip in the xy-plane, if the tooltip crosses a critical distance to the obstacle until the height of the obstacle is exceeded. So the given condition for slowdown is:

If $(x_{tip} - x_{obs}) \le d_{crit} \wedge (q_3 \le h_{obs})$
decelerate $\dot{q}_1, \dot{q}_2$ until $q_3 \ge h_{obs}$.

Furthermore the permissible interval for the armature voltage U_i is widened to

$U_i \in [-U_{imax}^{max}, U_{imax}^{max}],\ i = 1,2,3.$

3.1 Implementation in MATLAB

The implementation of the obstacle avoidance in MATLAB is realised by extending the point-to-point motion model by using a two-dimensional event function, which stops the integration just when the above described condition is fulfilled. The first component of the event function detects, if the distance between the tooltip and the obstacle in the *xy*-plane is critical. The second component detects, whether the tooltip position exceeds the height of the obstacle in z-direction. As in the point-to-point motion model, the conditions are written in a logical structure.

```
function [ value , isterminal , direction ]
= event_function1 (~ , w)
       value =[1 - double (( abs ( L1 * cos
       ( w (3))+ L2 * cos (w (3)+ w (4)) -
       xobs )< dcrit )
       && (w (5) < hobs )); 1- double (w (5)
       > hobs )];
isterminal =[1;1];
direction =[ -1; -1];

end
```

Using a nested function, the right hand side of the differential equation is state-depending. So detecting the first event, the velocities $\dot{q}_1$ and $\dot{q}_2$ in the right hand side of the differential equation are decelerated dramatically by multiplying them with the factor $dec = 0.00005$. Additionally the widened interval for U_i is used here as well.

By using a while-loop this procedure is repeated using the final values of the last step before the detection of the event as new initial values for the integration until the second event occurs, so until the height of the obstacle is crossed. After this event the point-to-point motion model from task b can be used, since no obstacle is present anymore.

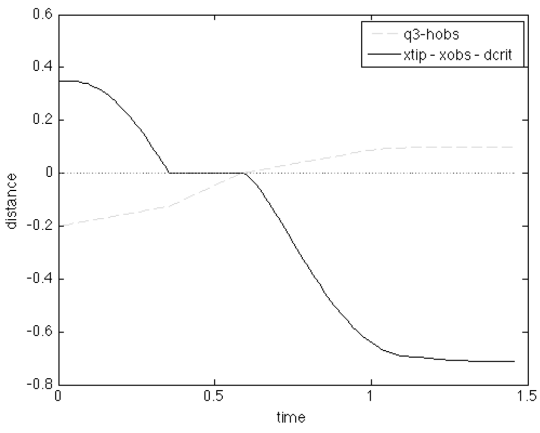

Figure 4: Distance between the joint vector and the obstacle in MATLAB.

In Figure 4 one can see that the motion of the tooltip has to be nearly stopped if it has a critical distance to the obstacle until the z-position of the tooltip exceeds the height of the obstacle and afterwards the point-to-point model is used. Here the initial values have to be set to zero, except the positions q_1, q_2 and q_3.

So the values for the velocities $\dot{q}_i$ and for the armature current $\dot{I}_i$, are set to zero. The reason for this is that otherwise the values of the last integration step of the extended collision avoidance model do not fulfil the initial value problem of the point-to-point motion model.

3.2 Implementation in SIMULINK

In Simulink, the point-to-point model is extended by distinguishing the different states as described above using a 'Switch'-block. This block switches between the values for $\dot{q}_1$, $\dot{q}_2$ and the values $\dot{q}_1 \cdot dec$ and $\dot{q}_1 \cdot dec$, where $dec = 0.00005$. Depending on the state this values are forwarded to the 'Integrator'-block and the right hand side of the differential equation.

The condition $(x_{tip} - x_{obs}) \leq d_{crit} \wedge (q_3 \leq h_{obs})$ is written in a logical structure using the "fcn"-block and a logical operator-block. The same condition is used in a second 'Switch'-block to distinguish between the permissible intervals for the armature voltage U_i. In Simulink it is not necessary to adjust the initial conditions after the height of the obstacle is exceeded.

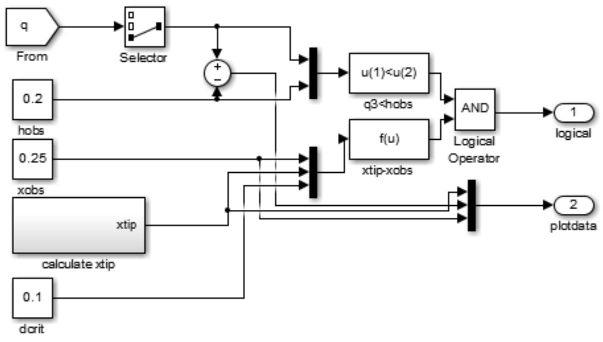

Figure 5: Condition for state – switch in Simulink.

In Figure 6 one can see a similar behaviour as in Figure 4 of the simulation in MATLAB. If the distance between the tooltip and the obstacle is critical the movement in the *xy*-plane is decelerated dramatically by multiplying by dec, until the height of the obstacle is crossed.

3.3 Results

There are some differences in the simulated results, so the movement of the tooltip differs, which can be observed in Figure 7. Until the first event occurs both simulations show very similar behaviour. But after the obstacle is passed the movements differ a lot.

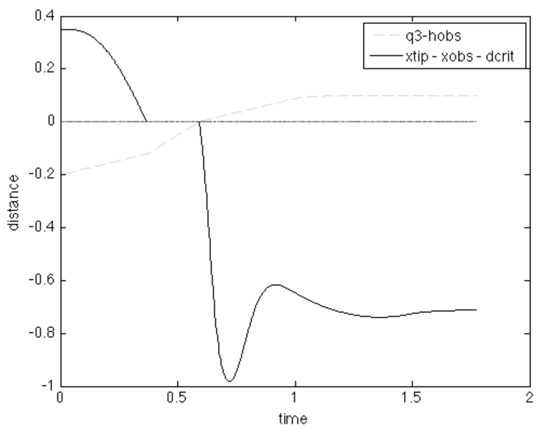

Figure 6: Distance between the joint vector and the obstacle in Simulink.

The reason for this deviance is that after the second event occurs the values of the velocities and the armature current are set to zero in the MATLAB model. In Simulink this is not necessary, therefore the velocity is bigger and the tooltip has to spin around to reach the target point - marked as a red dot in Figure 7.

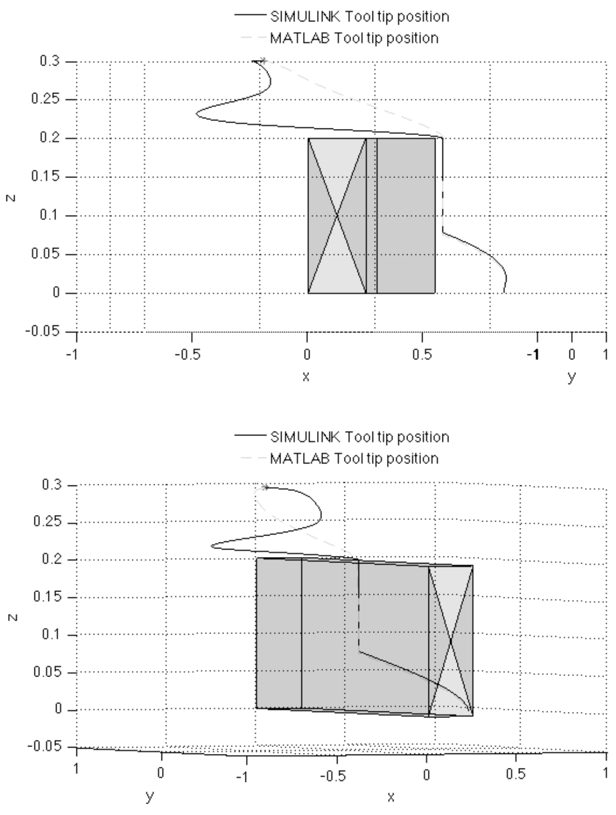

Figure 7: Tooltip position simulated in MATLAB and Simulink.

Furthermore, this also means that the simulation time of Simulink, which the tooltip needs to reach the target point, is much longer than of MATLAB.

Model	Simulation time
MATLAB	1.4555
Simulink	1.7736

Table 3: Simulation time of MATLAB and Simulink.

In Simulink it is possible to reduce the deceleration-factor to $dec = 0.005$, but the implementation in MATLAB cannot cope with this situation, because the slowdown of the movement in the xy-plane is too little. So the tooltip would not have enough time to exceed the height of the obstacle, which would mean that the tooltip would crash into the obstacle. So Simulink is much steadier in this situation than MATLAB.

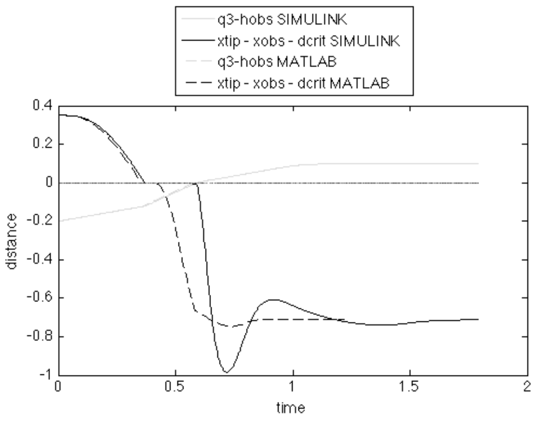

Figure 8: Distance between the joint vector and the obstacle in Simulink and MATLAB with $dec = 0.005$.

4 Discussion

As already mentioned two different kinds of comparisons have been done. On the one hand the explicit with the implicit implementation in MATLAB only were compared. Here the runtimes of both implementations were observed and resulting from this the explicit model was faster than the implicit one as already shown in Section 1.1.

Therefore, the further comparison of MATLAB and Simulink only refers to the explicit description of the given system. One big difference of those two kinds of implementation is that MATLAB cannot solve a second order differential equation system and therefore it had to be transferred into a first order differential equation system. But Simulink can handle such second order differential equation systems as given by connecting two 'Integrator'-blocks in series.

For the simple point-to-point motion there is no big difference in using either the MATLAB model or the Simulink model, both tooltips follow the same movements and the simulation times are also equal. The only difference is that the real runtime is a lot faster for the Simulink simulation than for the MATLAB one.

The last task which was implemented is the obstacle avoidance. The results of the two different implementations show that not only the simulation times and runtimes differ but also the movement of the tooltip. As the MATLAB model is quicker in solving this task it seems to be obvious to prefer the MATLAB implementation for the obstacle avoidance task. But one should not disregard that the Simulink model is steadier in respect of changing the factor dec for slowing down, especially in terms of increasing this factor and there is no need of changing the initial conditions in order to use the point-to-point model after crossing the height of the obstacle in the collision avoidance model.

Model sources

For this C11 Benchmark approach, all MATLAB model m-files, all Simulink model mdl-files, all parametrization m-files, and a short file description can be downloaded (zip format) by EUROSIM societies' members from SNE website, or are availably from the corresponding author.

References

[1] Ecker H, Comparison 11: SCARA Robot. EUROSIM-*Simulation News Europe*. 1998; 22: 30-32

[2] mathworks: Products and Services [Internet]. The MathWorks, Inc.: c1994-2014 [cited 2014 Nov 13]. Available from: http://uk.mathworks.com/products/index.html?s_tid=gn_loc_drop

[3] Hairer E, Wanner G. *Solving Ordinary Differential Equations II*. Germany: Springer; 2002. 491p.