ARGESIM Benchmark C11 'SCARA Robot': Comparison of Basic Implementations in EXCEL and MATLAB

Olga Rekova[1,2], Nicole Pelzmann[1,2], Philipp Mandl[1,2], Maximilian Hoffmann[1,2],

Horst Ecker[2*], Andreas Körner[1], Martin Bicher[1,3], Felix Breitenecker[1]

[1]Mathematical Modelling and Simulation Group, Inst. of Analysis and Scientific Computing
[2]Inst. of Mechanics and Mechatronics, [3]Inst. of Information Systems Engineering
TU Wien, Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria; **horst.ecker@tuwien.ac.at*

SNE 29(3), 2019, 149 - 158, DOI: 10.11128/sne.29.bne11.10488
Received: November 10, 2018; Revised: March 13, 2019;
Revised: July 25, 2019; Accepted:July 30, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. This SCARA robot benchmark study documents an EXCEL implementation and compares it with a MATLAB implementation on basis of Euler and Heun ODE solvers with integrated event handling. Aim is to check whether a spreadsheet tool like EXCEL can be used as simulator for continuous models as given for the SCARA robot. The benchmark study is organized in four parts. The first part describes the proper transformation and preparation of the implicit model description into an explicit state space with integrated control and state restrictions for use with explicit Euler and Heun solver. The second part documents the model implementations in EXCEL and in MATLAB for simulation of point-to-point movement. The third part concentrates on model extension for obstacle avoidance and proper implementation in EXCEL and in MATLAB. The fourth part compares the simulation results on a numerical basis and discusses advantages and disadvantages of the implementations. An appendix shows snapshots from the EXCEL implementations.

Introduction

ARGESIM Benchmark C11 'SCARA Robot' is based on a mechanical model for a three-axis SCARA robot (**S**elective **C**ompliance **A**ssembly **R**obot **A**rm).

The three degrees of freedom are constituted by two vertical revolute joints (angles q_1, q_2) and one vertical prismatic joint (distance q_3) as shown in Figure 1.

Such a system can be fully described by an implicit second-order system of differential equations:

$$M\ddot{\vec{q}} = \vec{b} \tag{1}$$

Here $\ddot{\vec{q}} = (\ddot{q}_1, \ddot{q}_2, \ddot{q}_3)^T$ represents the second derivative of the joint vector, and M is the mass matrix, which has a block-diagonal form and can be inverted symbolically:

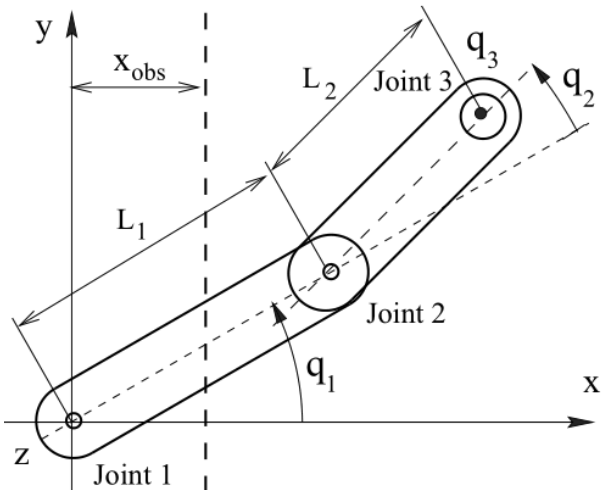


Figure 1: Three-axis SCARA robot, three degrees of freedom q_1 (rotational), q_2 (rotational), q_3 (translational). [1].

$$M = \begin{bmatrix} ma_{11} & ma_{12} & 0 \\ ma_{21} & ma_{22} & 0 \\ 0 & 0 & ma_{33} \end{bmatrix} \quad (2)$$

The components of the mass matrix are given in the definition of the benchmark [1], but can also easily be derived using Lagrangian mechanics (neglecting the rotational kinetic energy of the load m_3L and of the motor for the vertical axis):

$$\begin{aligned} ma_{11} &= \Theta_1 + 2\Theta_2 cos(q_2) + \Theta_3 \\ ma_{12} &= \Theta_2 cos(q_2) + \Theta_3 \\ ma_{21} &= ma_{12} \\ ma_{22} &= \Theta_3 \\ ma_{33} &= m_{3L} + \Theta_{3mot} u_3^2 \end{aligned}$$

Here Θ_i are the moments of inertia. These moments are calculated based on the assumption that the two physical links are two rods of mass m_1 and m_2 with homogeneous mass distribution along their length L_1 and L_2.

The right-hand side $\vec{b} = (b_1, b_2, b_3)^T$ is made up of the following equations, whereby T_1 and T_2 are the joint torques and T_3 is the joint force - inputs for the uncontrolled systems:

$$\begin{aligned} b_1 &= T_1 + \Theta_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2{}^2) sin(q_2) \\ b_2 &= T_2 - \Theta_2 \dot{q}_1{}^2 sin(q_2) \\ b_3 &= T_3 - m_{3L} g \end{aligned}$$

For operation, servo motors for each axis drive the robot following a specific control scheme (joint torques and joint force are proportional to the current of the respective motor). The electrical relationship of the armature of a robot servo motor is given by a first order differential equation for the current I_i of the servo motors, whereby the current I_i must be limited to I_{ai}:

$$\dot{I}_i = g_{I,i} = \frac{U_{ai} - k_{T_i} u_i \dot{q}_i - R_{ai} I_{ai}}{L_{ai}},\ i = 1,2,3$$

$$I_{ai} = [-I_i^{max} \le I_1 \le I_i^{max}],\ i = 1,2,3 \quad (3)$$

Here k_{T_i}, u_i, R_{ai}, and L_{ai} are parameters, the control voltages U_i and U_{ai} resp. result from PD control for point-to-point movement with target joint position vector $\hat{\vec{q}} = (\hat{q}_1, \hat{q}_2, \hat{q}_3)^T$:

$$U_i = P_i(\hat{q}_1 - q_i) - D_i \dot{q}_1], i = 1,2,3$$

$$U_{ai} = [-U_i^{max} \le U_1 \le U_i^{max}], i = 1,2,3 \quad (4)$$

1 Explicit State Space Model

Challenge of this SCARA robot benchmark report was a proper implementation in the spreadsheet tool EXCEL and the comparison with a (basic) MATLAB implementation. EXCEL is no simulator, it does not provide ODE solvers or other simulation tools. But simple ODE solvers can be easily implemented by means of recursive cell update in columns. Consequently, solver choice was explicit Euler ODE solver and explicit Heun ODE solver, for an explicit state space

$$\dot{\vec{x}} = \vec{f}(\vec{x}) \quad (5)$$

given on a grid $t_0, t_1, ..., t_n$, $\vec{x}_i = \vec{x}(t_i)$ with constant stepsize $h = t_{i+1} - t_i$ by

$$\vec{x}_{i+1}^E = \vec{x}_i + h \cdot \vec{f}(\vec{x}_i) \quad (6)$$

$$\begin{aligned} \vec{x}_{i+1}^H &= \vec{x}_i + \frac{h}{2} \cdot (\vec{f}(\vec{x}_i) + \vec{f}(\vec{x}_i + h \cdot \vec{f}(\vec{x}_i))) \\ &= \vec{x}_i + \frac{h}{2} \cdot (\vec{f}(\vec{x}_i) + \vec{f}(\vec{x}_{i+1}^E)) \end{aligned} \quad (7)$$

To formulate the robot with servo motors and control in explicit state space form (5), first the second derivatives in (1) were replaced by three additional states, and the currents for the servo motors in (3) were also integrated in the state space, resulting in a 9 by 9 implicit system:

$$A(\vec{x}) \cdot \dot{\vec{x}} = \vec{g}(\vec{x}) \quad (8)$$

$$A = \begin{bmatrix} ma_{11} & ma_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ ma_{21} & ma_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ma_{33} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} \dot{\vec{x}} &= (\ddot{q}_1, \ddot{q}_2, \ddot{q}_3, \dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{I}_1, \dot{I}_2, \dot{I}_3)^T \\ \vec{x} &= (\dot{q}_1, \dot{q}_2, \dot{q}_3, q_1, q_2, q_3, I_1, I_2, I_3)^T \\ \vec{g}(\vec{x}) &= (b_1, b_2, b_3, \dot{q}_1, \dot{q}_2, \dot{q}_3, g_{I,1}, g_{I,2}, g_{I,3})^T \end{aligned}$$

As the mass matrix (2) can be inverted symbolically, also the matrix $A(\vec{x})$ in (8) can be inverted symbolically, so that the so-called semi-linear implicit state space de-

scription (8) can be made explicit:

$$\dot{\vec{x}} = A(\vec{x})^{-1} \cdot \vec{g}(\vec{x}) = \vec{f}(\vec{x}) \tag{9}$$

The explicit state space description (9) can now directly be inserted in the algorithm for the Euler solver (6) or Heun solver (7), if the limitations for the servo motor currents due to (3) are not necessary (otherwise for the limitations a special implementation must be used). The simple Euler solver (6) is of approximation order 1 and has a limited area of stability, so that an appropriate small step size must be used. The Heun solver (7) – which starts with an Euler step and improves the result by the trapezoidal rule – is of approximation order 2 and has also a limited area of stability, but allowing slightly bigger step sizes than Euler solver.

As conclusion, *Task A – Implicit Model Handling* is performed by transformation into an explicit state space description for proper use with explicit ODE solvers.

2 Implementation of Point-to-point Motion

The second task *Task b - Point-to-Point Movement* requires a proper implementation and simulation in the time domain for a point-to-point movement of the robot arm. This benchmark study compares an EXCEL implementation and a MATLAB implementation based on explicit Euler solver and explicit Heun solver using the explicit state space description (9), with modifications for the limitations of the currents. For better comparison, the algorithmic formulations in EXCEL and in MATLAB are as 'near' as possible, and no EXCEL macro features and no MATLAB modules are used.

2.1 Euler implementation – EXCEL

EXCEL is no simulator, it does not provide ODE solvers or other simulation tools. But simple ODE solvers can be easily implemented by means of recursive cell update in columns.

Figure 9 (see last section) shows parts of the spreadsheet implementation. There, the first row denotes time and states `t q1dot q2dot q3dot q1 q2 q3 I1 I2 I3` in columns `P Q R S T U V Z AA AB`, and the second row contains the initial values – all zero.

The following rows are recursive updates for time $t_{i+1} = t_i + h$ in column `P`, and due to (6) Euler integration $x_{i+1} = x_i + h \cdot f(x_i)$ in columns `P Q R S T U V Z AA AB` for the states $\dot{q}_1, \dot{q}_2, \dot{q}_3, q_1, q_2, q_3, I_1, I_2, I_3$, in EXCEL notation for instance given

for time t

```
P3:  = P2 + h
P4:  = P3 + h
P5:  = P4 + h
...  ...
```

for state $\dot{q}_1$

```
Q3:  =Q2+h*f1(Q2 R2 S2 T2 U2...)
Q4:  =Q3+h*f1(Q3 R3 S3 T3 U3...)
Q5:  =Q4+h*f1(Q4 R4 S4 T4 U4...)
...  ...
```

for state q_1

```
T3:  =T2+h*f4(Q2 R2 S2 ...)=T2+h*Q2
T4:  =T3+h*f4(Q3 R3 S3 ...)=T3+h*Q3
T5:  =T4+h*f4(Q4 R4 S4 ...)=T4+h*Q4
...  ...
```

Here the formula functions `f1,f2,...,f9` correspond to the derivative vector $(f_1, f_2, ..., f_9)^T$. `f1,f2,f3` are relatively complicated, as they result from symbolic inversion of the mass matrix (2) – they need auxiliary variables for simplification (see Figure 10, last section); `f4,f5,f6` are trivial, as they are only integrating the velocities (see above); and `f7,f8,f9` are complicated because of the state limitations due to (3) and (4) – the classical problem of space variables which must be limited. The following code snippet shows the EXCEL formula for the Euler update of $x_1 = \dot{q}_1$:

```
q1dot,i = q1dot,i-1+h*
(-ma22/(ma12,i*ma21,i-ma11,i*ma22)*
(u_1*3^0,5/2*kt_1*Ia1,i-1+Th_2*
(2*q1dot,i-1*q2dot,i-1+q2dot,i-1^2)*
SIN(q2,i-1))+ma12,i/
(ma12,i*ma21,i-ma11,i*ma22)
*(u_2*3^0,5/2*kt_2*Ia2,i-1-Th_2*
q1dot,i-1^2*SIN(q2,i-1)))
```

The index `i` hereby implies the time step, the values for these variables have to be calculated for each time step in the EXCEL sheet – in the EXCEL formulas all variables with a time index are replaced with the respective cell name. Variables without an index are constant and are named cells.

In order to implement the case distinction for the current of the motors an interim result for the current is calculated, based on the ODE and the limitations for current and voltage due to (3) and (4). These interim

calculations allow to limit the integration on the derivative, instead on the output: Since these interim results could result in values above the given threshold for these quantities an `IF`-statement is used to check if they are above their maximal or below their minimal values. If the interim results are outside the allowed region the given boundary values are used for the calculations, and if they are within the allowed region the interim values are used – see following code snippet:

```
U1,i = P_1*(q1_t-q1,i)-D_1*q1dot,i
U1a,i = IF(ABS(U1,i)>U_1maxreg;
U_1maxreg*SIGN(U1,i);U1,i)
I1,i = I1,i-1+h*((U1a,i-1-kt_1*
u_1*q1dot,i-1-R_a1*Ia1,i-1)/L_a1)
I1a,i = IF(ABS(I1,i)>I_1max;I_1max*
SIGN(I1,i);I1,i)
```

2.2 Euler Implementation – MATLAB

MATLAB is a powerful numerical programming environment for any tasks, also for 'manual' programming of dynamic simulations. SIMULINK is a MATLAB extension for graphical modelling and simulation of dynamic systems based on input/output relations, equipped with a powerful so-called ODE suite with many different ODE solvers – from explicit Euler solver to implicit stiff system solver with state event detection.

In MATLAB only a subset of this ODE suite is available, which does not include Euler solver and Heun solver – but both solvers – named ODE1 and ODE2 – can be downloaded from MathWorks as m-file [2]. As basic ODE1 solver and basic ODE2 solver only terminate on time conditions and do not provide state event termination, for the tasks of this benchmark both solvers had to be modified – for *Task b - Point-to-Point Movement* using a simple `IF`-statement for terminating the integration loop (state update loop) as soon as a certain state is reached. For *Task c - Collision Avoidance* more complex `IF`-statements are necessary to distinguish and terminate the different simulation phases.

Generally, ODE solver libraries require precise formulation of the derivative function $\vec{f}(\vec{x})$ and perform the integration steps (the state updates) unconditionally - so also MATLAB's ODE suite does. That means that derivatives can be limited in the formulation of the function, but states cannot be limited directly.
In case of the SCARA robot, the state variables for the current must be limited in advance, which requires a modification of the integration step. Consequently, also for the MATLAB Euler integration it is needed to make use of `IF`-statements with logic queries to implement the equations of the current and the voltage of the motors in MATLAB. These `IF`-statements are embedded within the ode solver's `FOR`-loop.

First results for the point-to-point motion of the tool tip in 3D space using the Euler solver with a step size of 0.0004 show a reliable behaviour (see Figure 2). The kinematic restrictions result in a bend of the path toward the end and the target position for q_1 and q_2 is reached before that of q_3.

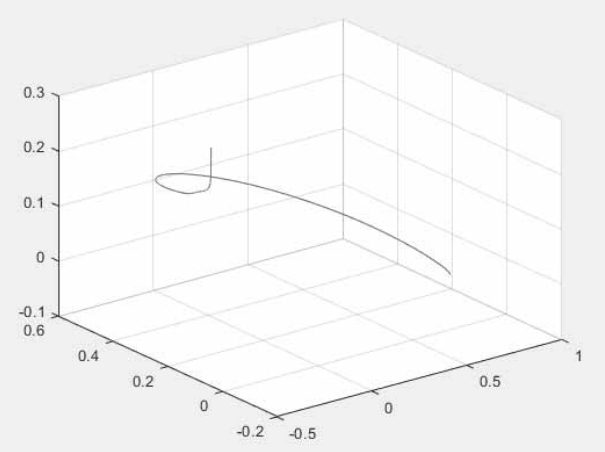


Figure 2: Point-to-point motion of the tool tip in 3D space (Euler solver, MATLAB, step size 0.0004).

2.3 Heun Implementation – EXCEL

The Heun solver requires two evaluations of the derivative function vector $\vec{f}(\vec{x})$, the first $\vec{f}(\vec{x}_i)$ for the Euler approximation (6) by $\vec{x}^E_{i+1} = \vec{x}_i + h \cdot \vec{f}(\vec{x}_i)$, and the second $\vec{f}(\vec{x}^E_{i+1})$ for the Heun correction due to (7).

For both slope vectors, in the EXCEL implementation now auxiliary variables (new columns) are used: `K1_1,...,K1_9` for $\vec{f}(\vec{x}_i)$, and `K2_1,...,K2_9` for $\vec{f}(\vec{x}^E_{i+1})$ (see Figure 10, last section). Using these auxiliary variables, which also double the auxiliary variables for components of the inverted mass matrix, the Heun step for the first state `q1_dot` (being $x_1(t_{i+1})$) becomes:

```
K1,1,i = q1ddot,i-1 = -ma22/
  (ma12,i-1*ma21,i-1-ma11,i-1*ma22)*
  (u_1*3^0,5/2*kt_1*Ia1,i-1+Th_2*
```

```
   (2*q1dot,i-1*q2dot,i-1+q2dot,i-1^2)*
   SIN(q2,i-1))+ma12,i-1/
   (ma12,i-1*ma21,i-1-ma11,i-1*ma22)*
   (u_2*3^0,5/2*kt_2*Ia2,i-1-Th_2*
   q1dot,i-1^2*SIN(q2,i-1))
K2,1,i = q1ddot,pred,i-1 = -ma22/
 (ma12,i-1*ma21,i-1-ma11,i-1*ma22)*
 (u_1*3^0,5/2*kt_1*Ia1,pred+Th_2*
 (2*(q1dot,i-1+h*K1,1,i-1)*
 (q2dot,i-1+h*K1,2,i-1)+
 (q2dot,i-1+h*K1,1,i-1)^2)*
 SIN(q2,i-1+h*q2dot,i-1))+
 ma12,i-1/
 (ma12,i-1*ma21,i-1-ma11,i-1*ma22)*
 (u_2*3^0,5/2*kt_2*Ia2,pred,i-1-Th_2*
 (q1dot,i-1+h*K1,1,i-1)^2*
 SIN(q2,i-1+h*q2dot,i-1))
q1dot,i = q1dot,i-1+(h/2)*
 (K1,1,i-1+K2,1,i-1)
```

In the above EXCEL formula, the last two lines represent the Heun update due to (7).
Predicted values for limited variables as currents and voltages are calculated separately within their boundaries, so in above formula no conditional statements are necessary (but again the number of auxiliary variables represented in new columns increases):

```
U1pred,i = P_1*
 (q1_t-(q1,i+h*K1,1,i))-
 D_1*(q1dot,i+h*K1,1,i)
U1a,pred,i = IF(ABS(U1,pred,i)>
 U_1maxreg;U_1maxreg*
 SIGN(U1,pred,i);U1,pred,i)
I1a,pred,i = IF(ABS(I1,i+h*K1,1,i)>
 I_1max;I_1max*SIGN(I1a,i+h*K1,1,i);
 I1a,i+h*K1,1,i)
```

Despite the mathematical simplicity of the Heun method the necessity of many auxiliary variables the complexity of the calculations in EXCEL increases. A better, but advanced EXCEL technique would be the use of EXCEL macros for the derivative functions.

2.4 Heun Implementation – MATLAB

The implementation of the Heun method in MATLAB directly follows the Euler implementation, but using two evaluations of the derivative function with following use of trapezoidal rule due to (7) in the state update loop. Again the provided Heun solver has to be modified with respect to the state limitations for the currents – with formula very similar to the above sketched EXCEL formula.

3 Obstacle Avoidance

The third task *Task c - Collision Avoidance* requires extension of the model description for handling a collision avoidance manoeuvre. The obstacle, a box, is situated at a certain x-position x_{obs}, and has a certain height h_{obs}. If the tool tip of the robot gets too near to the obstacle in the xy-plane (nearer than a critical distance d_{crit}), motion in xy-plane must stop, and the robot can move only upwards in z-direction (q_3-direction) as fast as possible, until the height of the obstacle is reached. This collision avoidance manoeuvre is given by condition formula

$$(d = x_{tip} - x_{obs}) \leqslant d_{crit} \ \wedge \ q_3 < h_{obs} \qquad (10)$$

3.1 Euler and Heun Implementation – EXCEL

For EXCEL implementation, same the principles as in *Task b - Point-to-Point Movement* are used, but with more complicated control actions depending on conditions. Consequently, several *IF*-statements as well as new auxiliary variables are added to implement the collision avoidance manoeuvre, including Cartesian coordinates for the positions.

During each step the distance d in x-direction between the tool tip and the obstacle due to (10) is calculated. As soon as this distance is smaller than the critical distance and the tool tip is not above the obstacle height target x-position and target y-position are set to the current x-position and current y-position, as well as the boundaries for voltages of the motors are changed to the emergency maximum. This new target position as well as voltage maxima are kept until the tool tip has risen above the obstacle height.

To realize the switch of the target position to the current position an auxiliary variable `d_mod` is calculated which is equal to the actual distance to the obstacle as long as it is bigger than the critical distance, but frozen after the distance falls below the critical value and the tool tip is below the obstacle height. By referencing to this `d_mod` and to the current z-position of the tool tip, the variable boundaries can be changed comfortably to their emergency maximums after the distance falls below the critical distance, and all other calcula-

tions are taken from formulas used in the implementation for *Task b - Point-to-Point Movement*. The EXCEL formulas for this controlling `d_mod` are:

```
dmod,i = IF(AND(dmod,i-1<
 d_crit;q3,i-1-h_obs<0);
 d_mod,i-1;d,i)
q1_t,i = IF(AND(q3,i-h_obs<0;
 d_mod,i<d_crit);q1,i-1;q1_t,i-1)
U1a,i = IF(AND(q3,i-h_obs<0;
d_mod,i<d_crit);IF(ABS(U1,i)>
U_1max;U_1max*SIGN(U1,i);U1,i);
IF(ABS(U1,i)>U_1maxreg;U_1maxreg*
SIGN(U1,i);U1,i))
```

The method of freezing `d_mod` is chosen to avoid switching back to the original target position before the tool tip has reached a height above the obstacle and thereby oscillating around the critical distance.

The Heun implementation in EXCEL follows the Euler implementation, but using for state update the Heun solver with additional variables as given in *Task b - Point-to-Point Movement*.

3.2 Euler and Heun Implementation – MATLAB

The MATLAB implementation chooses a separation of the dynamics in three phases: PD-controlled movement to target position until obstacle detection, obstacle avoidance movement until non-critical heights, and PD-controlled movement to target position. For the first phase, the ODE1 solver from *Task b - Point-to-Point Movement* is used, extended by movement stop at obstacle detection due to (10); the second phase is governed by an modified ODE1 solver which uses the (simpler) collisions avoidance control, until a non-critical height is reached, and the third phase can make use of the ODE1 solver from *Task b - Point-to-Point Movement*. In MATLAB all results are concatenated:

```
y1 = ode1_1(@(t,y) reach_target(t,y),
     tspan,y0);
y2 = ode1_2(@(t,y) avoid_obs(t,y),
     tspan,y1(length(y1),:));
y3 = ode1_3(@(t,y) reach_target(t,y),
     tspan,y2(length(y2),:));
```

In the implemented ODE solvers `IF`-statements (given below) stop the integration loop on occurence of a specific event. The first `IF`-statements stops the integration loop of `ODE1_1` if the tool tip drops below the security threshold due to (10). The second `IF`-statements stops the integration loop of `ODE1_2`, if the tool tip has exceeded the obstacle heights, and the integration loop of `ODE1_3` stops when the tool tip reaches the target position:

```
if  1 - (abs(L1*cos(Y(4,i+1)) +
  L2*cos(Y(4,i+1)+Y(5,i+1)) - xobs)
  <=dcrit && (Y(6,i+1)<hobs)) == 0
if  (1 - ((Y(6,i+1) - hobs)>hsafe)) == 0
if  (abs(Y(4,i+1) - 2) < 0.001
  & abs(Y(5,i+1) - 2) < 0.001
  & abs(Y(6,i+1) - 0.3) < 0.001) - 1 == 0
```

To detect the obstacle, the tool tip position is calculated and an `IF`-statement checks whether the position exceeds any restriction. Afterwards when the event is detected the position of the tool tip is locked in *x*-direction and *y*-direction, and the second ode solver started.
To detect the end of the obstacle, another `IF`-statement compares the current tool tip height with the obstacle height and stops the solver as soon as the tool tip exceeds the obstacle height. Thereafter the last ODE solver takes over and lets the robot move freely until the tool tip arrives at the designated position.

The Heun implementation simply replaces the modified Euler ODE1 solver by the modified Heun ODE2 solver for construction the three different Heun solvers `ODE2_1`, `ODE2_2`, and `ODE2_3` for the three phases.

4 Results – Comparison – Discussion

In order to compare the different solutions between MATLAB and EXCEL on the one side, and between Euler solver and Heun solver, all simulations are performed with the same step size *h*. Results from EXCEL simulations are imported into MATLAB and plotted with MATLAB plot features.

The choice of a proper step size is a critical task. Euler solver and Heun solver are explicit solvers, so they have limited stability regions, and so the step size is also limited – on the other hand side small step sizes result in a very big number of rows in the EXCEL implementations - a minimum of 3000 in the calculated simulations.

4.1 Results point-to-point motion

The reliable results in Figure 2 for the point-to-point motion of the tool tip in 3D are calculated with a step size of $h = 0.0004$. This step size requires about 7000 rows in EXCEL.

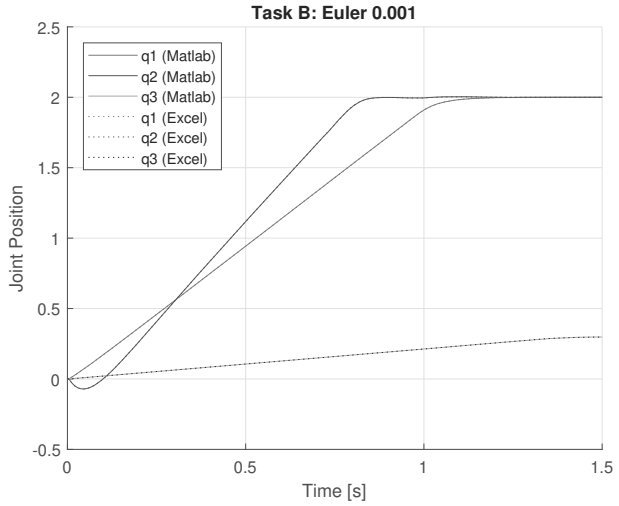


Figure 3: States q_1, q_2 and q_3 over time for *Task b - Point-to-Point Movement*, Euler solver with step size 0.001. EXCEL solutions (dotted lines) and MATLAB solutions (solid lines) show negligible differences.

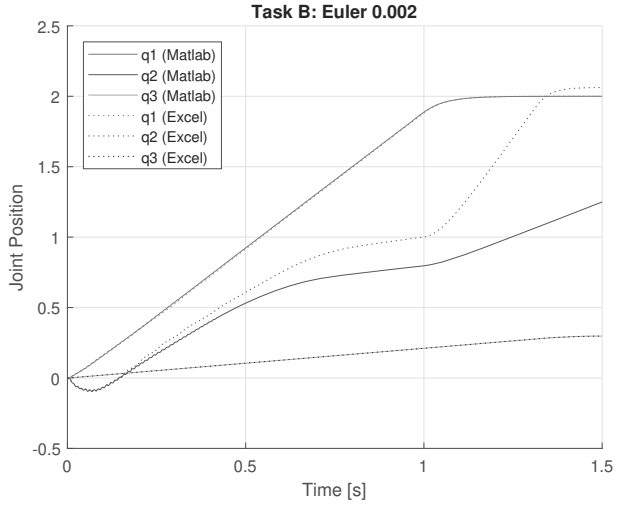


Figure 4: States q_1, q_2 and q_3 over time for *Task b - Point-to-Point Movement*, Euler solver with step size 0.002. EXCEL solutions (dotted lines) and MATLAB solutions (solid lines) differ significantly.

Time domain results for the joint coordinates with same step size $h = 0.0004$ and step size up to to step size of $h = 0.001$ (only 3000 rows necessary) coincide for EXCEL and for MATLAB implementation – 'classical' correct results for this benchmark, as given in Figure 3.

Experiments with the step size in *Task b - Point-to-Point Movement* indicate, that the step size $h = 0.001$ is a 'critical' maximal allowable step size. Figure 3 compares the MATLAB results and the EXCEL results for this step size $h = 0.001$ and shows graphically a good coincidence; also a numerical comparison results in a minor expected deviation.

But for a step size of $h = 0.002$ and bigger the solutions differ more significantly, as documented in Figure 4 graphically, and as checks of the numerical differences proof. Interestingly, the differences of the EXCEL solutions with $h = 0.001$ and $h = 0.002$ are bigger than the differences of the MATLAB solutions with $h = 0.001$ and $h = 0.002$. A possible reason is a more sensitive behaviour of EXCEL due to accumulating round-off errors – a topic for further investigation and better error parameter tuning in EXCEL.

Usually the use of a higher order ODE solver lets expect more accurate results with the same step size, or results with same accuracy using a bigger step size. Unfortunately this expectation does not hold for the Heun solver in case of the investigates model, although he has order 2. The stability region of the Heun solver extends only in the imaginary direction for the eigenvalues. This would allow bigger step sizes for oscillating behaviour, being not the case in the investigated model. Consequently also for the Heun solver the step size $h = 0.001$ is the critical maximal possible step size. A bigger step $h = 0.002$ results in differences similar to that of the Euler solver with step size $h = 0.002$, and additionally the EXCEL solutions are more stronger affected by round-off errors, so that EXCEL results with Heun and step size $h = 0.002$ are worse than EXCEL results with Euler and step size $h = 0.002$, especially EXCEL results for state q_2 seem to be definitely wrong.

4.2 Results collision avoidance

Generally, the results for *Task c - Collision Avoidance* are reliable for the MATLAB implementation and for the EXCEL implementation, if the step size is chosen properly.

Figure 5 displays the results for the tool tip position x_{tip} and for $q_3 - h_{obs}$, the distance to the obstacle in z-direction over time for Euler solver with step

size $h = 0.001$: the tool tip is approaching the obstacle, after detection stopping movement in x-direction (also moving 'back' a little), and continuing x-direction movement after reaching the security height. The solid lines for the MATLAB solutions overlap the dotted lines for the EXCEL solutions, as the results are almost the same. These results are similar to other benchmark solutions already published, with slight differences at begin of the collision avoidance manoeuvre because of differences in implementing the manoeuvre.

Again the step size $h = 0.001$ turns out to be the maximal allowable one. Euler solver with step size $h = 0.002$ results in differences between MATLAB implementation and EXCEL implementation for the tool tip position x_{tip} already in the first phase (approaching the obstacle), increasing in the phase of collision avoidance manoeuvre, and couriously overshooting in the third phase (Figure 6). The use of the Heun solver does not improve the accuracy, in contrary: the deviations between MATLAB implementation and EXCEL implementation for step size $h = 0.002$ are worsening.

For completeness, Figure 7 shows the motion of the tool tip in 3D space. There, due to the momentum of the system the tool tip overshoots the critical distance at first and then returns to the newly set target position in the xy-plane resulting in a slight bend of the motion next to the obstacle,

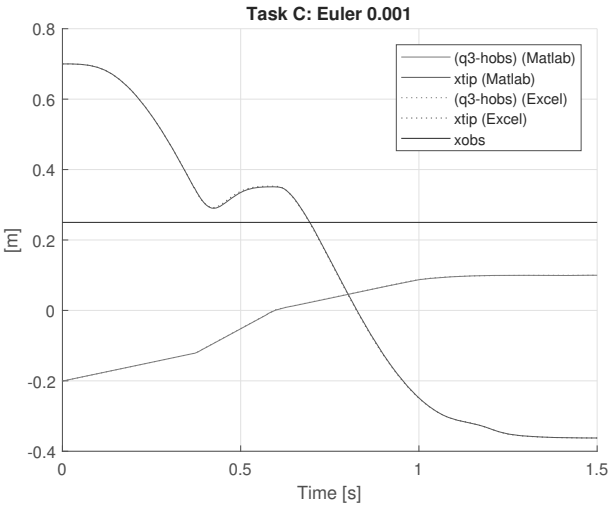


Figure 5: Tool tip position in x-direction x_{tip} and distance to the obstacle in z-direction $q_3 - h_{obs}$ for *Task c - Collision Avoidance*, calculated with Euler solver and step size $h = 0.001$.MATLAB and EXCEL solutions are almost congruent.

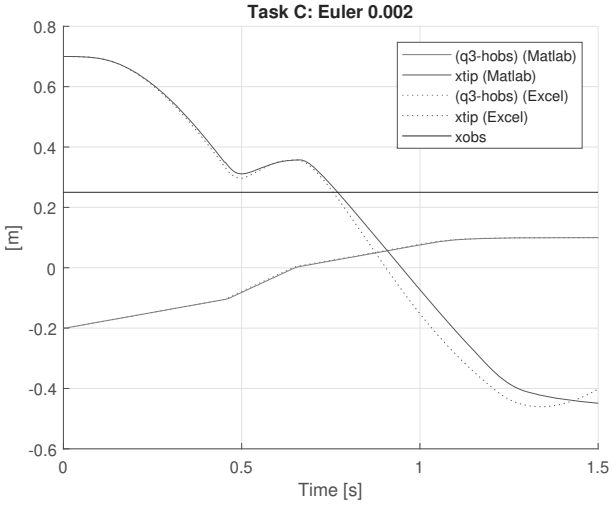


Figure 6: Tool tip position in x-direction x_{tip} and distance to the obstacle in z-direction $q_3 - h_{obs}$ for *Task c - Collision Avoidance*, calculated with Euler solver and step size $h = 0.002$. MATLAB solutions and EXCEL solutions differ, the EXCEL solution shows a slight overshoot.

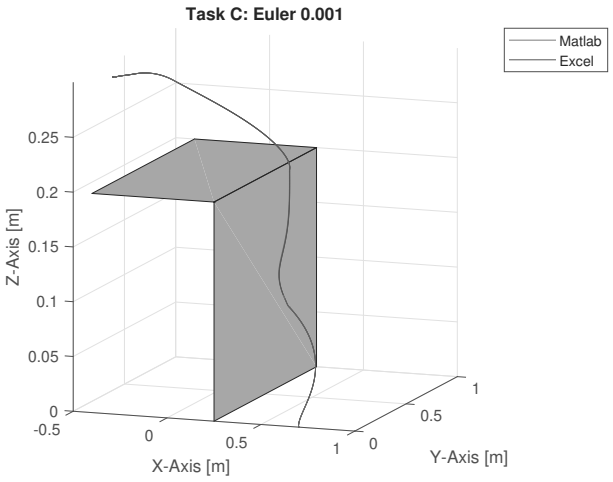


Figure 7: Motion of the tool tip in 3D space for *Task c - Collision Avoidance* using Euler solver with a step size $h = 0.001$. Due to the momentum of the system the tool tip overshoots the critical distance at first and then returns to the newly set target position in the xy-plane resulting in a slight bend of the motion next to the obstacle.

4.3 Comparison MATLAB - EXCEL

MATLAB is a powerful numerical programming environment for any tasks, also for 'manual' programming of dynamic simulations. SIMULINK is a MATLAB extension for graphical modelling and simulation of dynamic systems based on input/output relations, equipped with a powerful so-called ODE suite with many different ODE solvers – from explicit Euler solver to implicit stiff system solver with state event detection.

In MATLAB only a subset of this ODE suite is available, not including Euler solver and Heun solver, and not offering features for event detection and limited integration. So in any case the limitations, event detection, and event actions must be programmed 'manually' - with `IF-THEN-ELSE`-constructs – similar to the implementation in EXCEL. So basic MATLAB is not the best tool for the tasks of *ARGESIM Benchmark C11 'SCARA Robot'*.

A spreadsheet tool as EXCEL is definitely not a simulator – modelling features for ODEs, processes, events, etc. are missing. But spreadsheet programs are an excellent experiment environment with statistical analysis, optimisation, what-if analysis, data handling, etc. Of course, macros and external programming could be used, but to some extent the standard features allow to implement explicit ODE solvers as recursive formulas.

Basic implementations are faced with the problem of equidistant small step sizes, which are necessary in case of technical dynamic systems; here the round-off errors cause problems, and the number of rows in the spreadsheet may increase drastically.

It is to be noted, that also variable step size control could be implemented: in case of using solvers with different order (as here with Euler and Heun) the difference of the solvers in the integration step estimates the error, so that in case of a too big error the step size can be decreased – and increased in case of very small error. Especially the second case - step size increase - could prevent from the EXCEL-genuine round-off error.

But a general disadvantage is the lack of accuracy in the EXCEL standard configuration – possible but laborious to improve. On the other hand, a spreadsheet tool is a very suitable tool for education, so that this C11 benchmark study is mainly intended for educational use. On the other side, the use of advanced EXCEL features as macros, programmed modules, and EXCEL add-ons would allow much more comfortable implementation and also more accuracy.

References

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

[2] MathWorks Support Team *Is there a fixed-step Ordinary Differential Equation (ODE) solver in MATLAB 8.0 (R2012b)*; https://de.mathworks.com/matlabcentral/answers/98293; Oct. 2012

5 Appendix

EXCEL is no simulator, it does not provide ODE solvers or other simulation tools, and it does not have a structure for implementing dynamic models. On the other hand, EXCEL allows calculation and documentation of any kind and in any structure. So also ODE solvers - being state updates - can be implemented by recursive formula in consecutive rows (and cells).

The implementation developed in this benchmark study provides different worksheets for tasks and ODE solvers, but with same structure, see Figure 8. The upper left region of all worksheets (columns `A,B,..M`) is reserved for definition and documentation of the system: model sketch, summary of equations, definition of parameters (named cells), etc. Furthermore, right above the simulation parameters can be put in and changed: initial and target position, and step size for the ODE solver. At bottom, time diagrams are provided.

The calculation area starts with column P. Figure 9 sketches the first five rows of the recursive implementation of the Euler solver in columns `P, Q, ..,AB`. There, the first row denotes time and states, the sec-ond sets the initial values, and the following rows calcu-late recursively updates for time $t_{i+1} = t_i + h$ and states $x_{i+1} = x_i + h \cdot f(x_i)$ due to (6) Euler integration - details see Section 2.1. Depending on step size and on distance to target, a usually big number of rows have to be used for the full time course.

For calculating the derivative functions, the following columns `AC,...,AN` provide auxiliary and control variables. The Heun solver must calculate a second evaluation of the derivative functions, so further columns from column AR on are foreseen (details in Section 2.3, sketch in Figure 10). The cell content window in Figure 9 and Figure 10 show the formula for calculation the control voltage: from a relatively simple formula in Figure 9 for *Task b - Point-to-Point Movement* to a more complex one for *Task c - Collision Avoidance* in Figure 10.

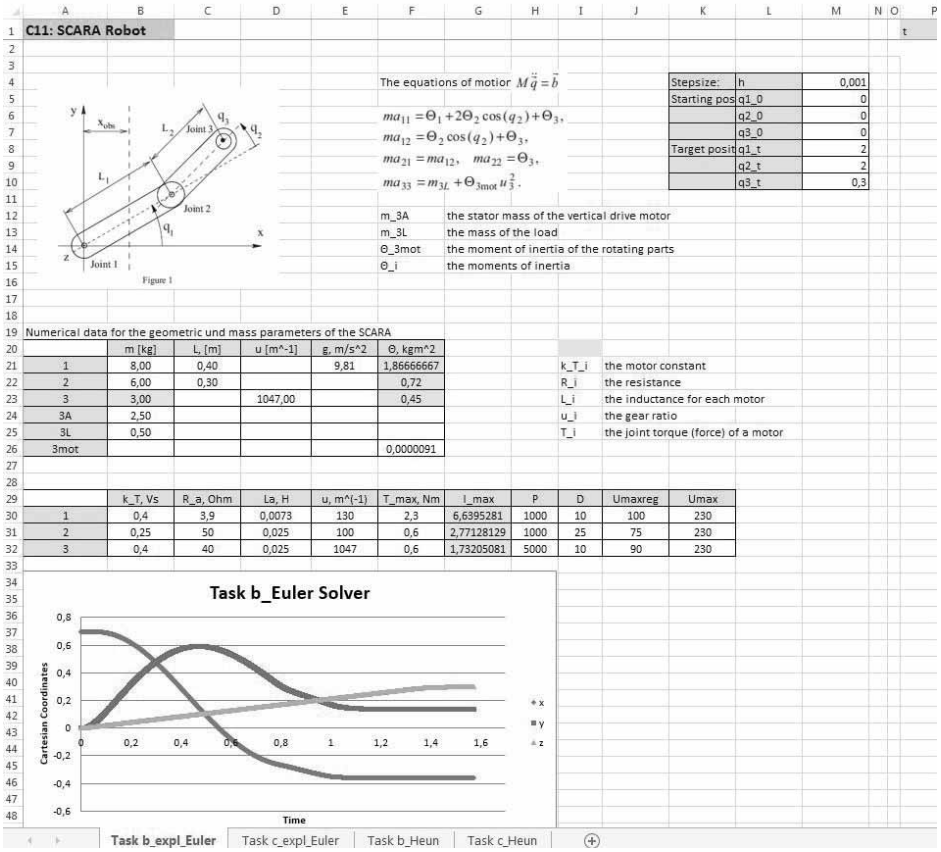
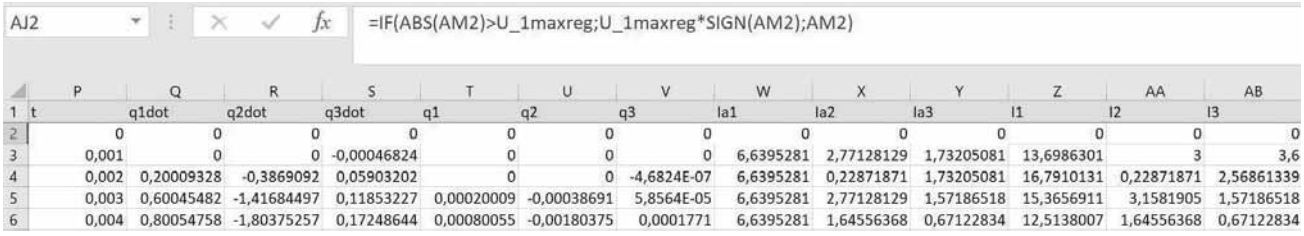
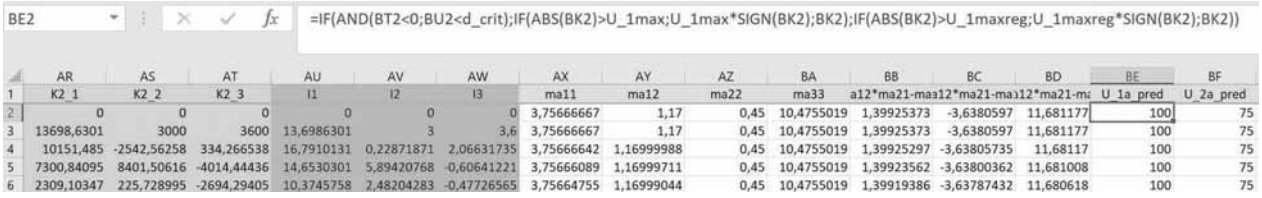


Figure 8: EXCEL implementation – definition and documentation area.



t	q1dot	q2dot	q3dot	q1	q2	q3	Ia1	Ia2	Ia3	I1	I2	I3
0	0	0	0	0	0	0	0	0	0	0	0	0
0,001	0	0	-0,00046824	0	0	0	6,6395281	2,77128129	1,73205081	13,6986301	3	3,6
0,002	0,20009328	-0,3869092	0,05903202	0	0	-4,6824E-07	6,6395281	0,22871871	1,73205081	16,7910131	0,22871871	2,56861339
0,003	0,60045482	-1,41684497	0,11853227	0,00020009	-0,00038691	5,8564E-05	6,6395281	2,77128129	1,57186518	15,3656911	3,1581905	1,57186518
0,004	0,80054758	-1,80375257	0,17248644	0,00080055	-0,00180375	0,0001771	6,6395281	1,64556368	0,67122834	12,5138007	1,64556368	0,67122834

Figure 9: EXCEL implementation – calculations for *Task b - Point-to-Point Movement*.



K2_1	K2_2	K2_3	I1	I2	I3	ma11	ma12	ma22	ma33	a12*ma21-ma	12*ma21-ma	12*ma21-m	U_1a_pred	U_2a_pred
0	0	0	0	0	0	3,75666667	1,17	0,45	10,4755019	1,39925373	-3,6380597	11,681177	100	75
13698,6301	3000	3600	13,6986301	3	3,6	3,75666667	1,17	0,45	10,4755019	1,39925373	-3,6380597	11,681177	100	75
10151,485	-2542,56258	334,266538	16,7910131	0,22871871	2,06631735	3,75666642	1,16999988	0,45	10,4755019	1,39925297	-3,63805735	11,68117	100	75
7300,84095	8401,50616	-4014,44436	14,6530301	5,89420768	-0,60641221	3,75666089	1,16999711	0,45	10,4755019	1,39923562	-3,63800362	11,681008	100	75
2309,10347	225,728995	-2694,29405	10,3745758	2,48204283	-0,47726565	3,75664755	1,16999044	0,45	10,4755019	1,39919386	-3,63787432	11,680618	100	75

Figure 10: EXCEL implementation – calculations for *Task c - Collision Avoidance*.