Comparison 11 – DYMOLA

Classical mechanical approach

Automatic-symbolical and numerical inversion

DYMOLA (Dynamic Modelling Laboratory) is an object-oriented simulation environment for the modeling, simulation and visualisation of continuous processes. Besides the classical textual model definition, Dymola provides an editor for graphical model editing together with comfortable possibilities to reuse objects by means of (graphical) libraries. Model details can be given by ODE's and DAE's in the Dymola's object-oriented modelling language; for simulation either Dymola's simulator Dymosim can be used, or other commercial simulators (e.g. ACSL, SIMULINK).

Model Description (Task a): One of the main characteristics of Dymola is the law-oriented model description allowing the formulation of DAE models. This description can be manipulated symbolically depending on certain options. In this solution the textual mode of Dymola is used, defining the equations of motion in DAE form in a Dymola class, instanced once:

```
model class components
  parameter L1=0.4, L2=0.3, L3=0.3, ...
  constant m1=8, m2=6, m3A=2.5, ...
  local M(3,3), b(3), q(3), dq(3), ...
  cut torq1(T1) torq2(T2) torq3(T3) ...
  M=[th1+2*th2*cos(q(2))+th3, ...
  b=[T1+th2*(2*dq(1)*dq(2)+dq(2)**2)* ...
  q=[q1;q2;q3]; dq=[dq1;dq2;dq3]
  dq=der(q)
  M*der(dq)=b
end
```

Dymola is able to transform these equations to explicit form by symbolically inverting the mass matrix, resulting in an explicit system. Another method is provided by a numerical inversion of the mass matrix by means of an iterative algorithm, resulting in a pseudo-explicit system. Depending on options, Dymola translates the system into both forms (as a third form, a pure implicit description is possible).

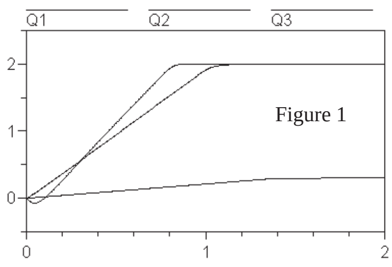
Point to Point Control (Task b): As three motors and three controls are required, two classes are defined: one describing the equations to specify a motor, one defining a control. Then the three instances of each class are connected with the instance of the components class, e.g.:

```
model class drive
  ...
  cut torque (T)
end
model scara
  submodel(drive) d1 d2 d3
  submodel(components) robot
  connect d1:torque at robot:torque1
  connect d2:torque at robot:torque2
  ...
end
```

The bounds for the voltages and the currents are considered by **if**-statements:

```
U=if abs(P*(qdach-q)-D*qd-Umax)0 then
  Umaxreg else P*(qdach-q)-D*qd-Umax
```

For simulation Dymosim is used. Dymola translates each **if** and **when** statement into a state event. Therefore the bounds for the current are formulated with state events in each target simulator. Dymosim handles state events by means of built-in features of the integration algorithms DASSL (used here) and LSODER. Unfortunately, Dymosim does not use the DASSL algorithm for direct integration of implicit equations (third method).



Of course numerical inversion of the implicit system takes more time than the integration of the explicit system; the relation is shown in the table below.

Figure 1 shows the graphs of the three joint positions.

Model description	Norm. CPU-time
Task b) explicit – symbolic inversion	1 (1.32s at P120)
Task b) pseudo-explicit – numerical inversion	1.69
Task c) explicit – symbolic inversion	1.43
Task c) pseudo-explicit – numerical inversion	2.27

Obstacle avoidance (Task c): For the collision avoidance a new class that observes and controls each state variable is implemented. The two instances (one for each state) check the distance between the obstacle and the tool-tip of the robot, and apply either the emergency maximum voltages and set the target positions for the state-variables to the current position, or reset the target positions to the original values bounding the voltages to their regular-mode-interval. Below a part of this description is shown, which is translated into a state event in Dymosim:

```
Xcrit=if abs(Xtip-Xobs)Xobs+Dcrit) then
true else false
Umax=if Xcrit and (q3-Hobs) then
Umaxmax else Umaxreg
q1dach=if Xcrit and (q3-Hobs) then
      q1 else q1target
```

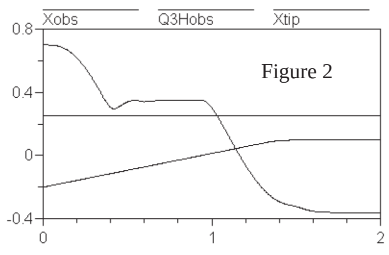


Figure 2 illustrates the behaviour of the tool-tip in this situation: it does not cross the critical region as long as the end-effector has not cleared the obstacle's height. This implementation of obstacle avoidance increases simulation time by a factor 1.43 (see table).

E. Forsthuber, Techologie-Zentrum Steyr, A-4400 Steyr, email: forsthuber@titania.tuwien.ac.at