Comparison 11 – MATLAB/SIMULINK
Hybrid Modelling Approach – Model Level

MATLAB is a widely used software tool based on numerical vector and matrix manipulation, SIMULINK is MATLAB's extension for graphical modelling and numerical simulation of dynamic systems.

Model Description (Task a): The model was implemented in two ways, using MATLAB 5.2, SIMULINK 1.3. First SIMULINK's `Algebraic Constraint` block was used and the implicit equation $b(q,\dot{q}) - M(q_2)\cdot\ddot{q} = 0$ directly implemented:

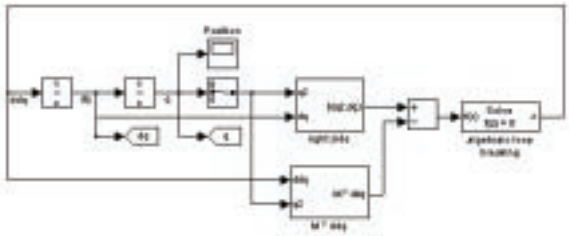

Fig.1: Implicit Equation, Algebraic Loop Breaking

For every integration step SIMULINK's `Algebraic Constraint` block *searches* for a solution of the implicit equation. This procedure is comfortable and does also work in the presence of a `Hit Crossing` block (which was needed for task c)! For the second solution the systems mass matrix M was inverted symbolically outside MATLAB and the explicit equation $\ddot{q} = M(q_2)^{-1}\cdot b(q,\dot{q})$ implemented in SIMULINK.

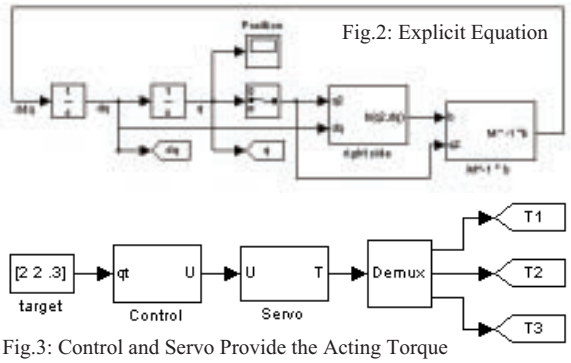

Fig.2: Explicit Equation

Fig.3: Control and Servo Provide the Acting Torque

Point to Point Control (Task b): For point to point movement both solutions use the same controller. A target vector is the input for Submodel `Control` which contains the PD-Controller (`q` and `dq` are provided by `Goto` blocks). The output `U` (the applied voltage) is fed into submodel `Servo` which models the servo drives of the three axes. Finally the resulting torque `T` is provided for the calculation of the right hand side `b`. For implementation of the boundaries of the voltages and currents, SIMULINK offers a very comfortable way: Voltage `U` is bounded by a `Saturation` block inside the submodel `Control` and the resulting armature current is limited by the corres- ponding `Integrator` block itself (inside submodel `Servo`). Figure 4 shows the graph of the joint positions for the demanded movement. Of course execution of the explicit model takes more time than for the implicit one. The processing times were measured from MATLAB using the commands `tic` and `toc` (average of four runs):

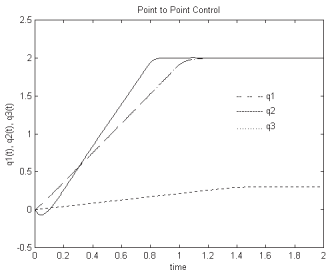

Fig.4: Joint Positions

Model description	Norm. CPU-time
Explicit - inverted matrix	1 (4.57s at P150)
Implicit - algebraic loop breaking	3.28

Obstacle Avoidance (Task c): For collision avoidance submodel `Control` was extended. The distance between the obstacle and the tool tip is permanently checked. If it gets smaller than the critical distance (event `too near`) the target positions for the state-variables are changed to the current position and the emergency maximum voltages are allowed. The robot arms 1 and 2 slow down and return to the position where the danger has been detected. Just after the tool tip of the robot has reached an admissible height (event `clear`) the original target position is reactivated and the arms 1 and 2 start to move again.

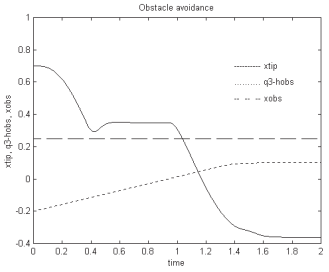

Fig. 5: Obstacle Avoidance

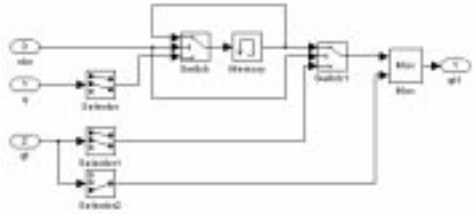

Fig. 6: Changed Target Position in Case of Obstacle

Figure 6 shows how the target position is changed in case of the event `too near`. As soon as the signal `obs` equals one, the current position `q` is stored in a `memory` block. This position is used as target for axes 1 and 2 until the occurrence of event `clear`: `obs` turns `zero`, target `qt` is accepted again and the voltages are limited to `Ureg`.

J. Scheikl, M. Lingl, SIMTECH / ARGESIM, TU Vienna, Wiedner Hauptstr. 8-10, A-1040 Vienna, email: joxg@osiris.tuwien.ac.at