Using RTSS with the Scicos package

Matteo Morelli*

October, 2007

1 Introduction

RTSS comes with several demonstration scripts which show how the toolbox can be productively used in studying robotics. Demos on subjects like

- Homogeneous Transformations
- Cartesian and joint-space Trajectories
- Forward and Inverse Kinematics
- Differential Motions and Manipulator Jacobians
- Forward and Inverse Dynamics
- Graphical Animations of Robot Motions

can be run individually by choosing "Robotics" in the Scilab demo menu.

Other demos are also provided with RTSS. They illustrate common topics in robot control and give the user an insight into the use of RTSS for constructing robot kinematic and dynamic models with Scicos.

Users with no previous Scicos experience are advised to read the relevant manuals [1, 2, 3, 4] and experiment with the examples supplied. Experienced Scicos users should find the use of the Robotics blocks quite straightforward.

In the following sections, a short description of all the examples of Scicos diagrams provided with RTSS will be given. Each of these demos can be run by executing the corresponding Scilab simulation script that is located in the `<PATH>/demos/` directory, where `<PATH>` is the installation path of RTSS.

*Interdepartmental Research Center "E. Piaggio", University of Pisa, Italy

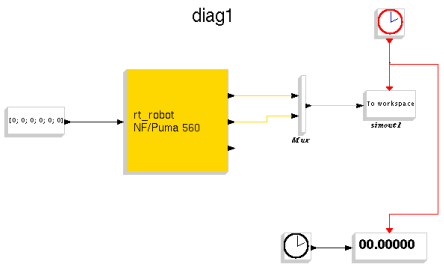

Figure 1: RTSS example `diag1.cos`

2 Dynamic simulation of Puma 560 robot collapsing under gravity

The Scicos model `diag1.cos`, shown in figure 1, is the Scilab counterpart of the Simulink model `demo1.mdl` provided with the Robotics Toolbox for MATLAB– from which RTSS draws inspiration –written by Professor Peter Corke [5].

The `rt_robot` block in this diagram would be familiar to RTSS users. It is similar to the `rt_fdyn()` function and represents the forward dynamics of the robot (in this case, a Puma 560 robot). The parameters of the `rt_robot` block contain the robot object to be simulated and the initial joint angles.

This demo can be run by executing the corresponding Scilab simulation script `rt_demo1.sce`. To do this, just type the following command at the Scilab prompt (please, replace `<PATH>` with the RTSS installation path):

```
exec <PATH>/demos/rt_demo1.sce;
```

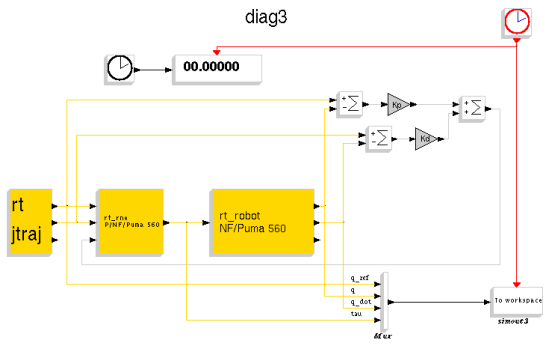

Figure 2: RTSS example `diag3.cos`

Alternatively, to run this example, first create a Puma 560 robot object in the workspace, then start the simulation using Simulate/Run option from the diagram toolbar:

```
exec <PATH>/models/rt_puma560.sce;
scicos("<PATH>/demos/scs/diag1.cos");
```

Please note that following this procedure, the simulation results will have to be processed manually (see the file `<PATH>/demos/rt_demo1.sce` for further details).

Finally, it is important to note that the presence of friction in the dynamic model can prevent the integration from converging. For this reason, the function `rt_nofriction` has been used in the context of the Scicos diagram to return a friction-free robot object.

3 Computed torque control of a Puma 560 robot

The Scicos model `diag3.cos`, shown in figure 2, is the Scilab counterpart of the Simulink model `demo3.mdl` provided with the Robotics Toolbox for MATLAB by Peter Corke. It represents a Puma 560 robot with a computed torque control structure [6].

This diagram introduces the `rt_rne` block which computes the inverse dynamics using the fast recursive Newton-Euler algorithm (see help on `rt_frne()`), and the `rt_jtraj` block which computes a vector quintic polynomial. `rt_jtraj` has parameters which include the initial and final values of the each output element as well as the overall motion time. Initial and final velocity are assumed to be zero.

Note that, in practice the dynamic model of the robot is not exactly known and we can only invert our best estimate of the rigid-body dynamics. In the simulation, this has been modeled by using the `rt_perturb()` function in the context of the Scicos diagram to alter the parameters of the dynamic model used in the `rt_rne` block.

This demo can be run by executing the corresponding Scilab simulation script `rt_demo3.sce`:

```
exec <PATH>/demos/rt_demo3.sce;
```

Alternatively, to run this example, first create a Puma 560 robot object in the workspace, then start the simulation using Simulate/Run option from the diagram toolbar:

```
exec <PATH>/models/rt_puma560.sce;
scicos("<PATH>/demos/scs/diag3.cos");
```

See the file `<PATH>/demos/rt_demo3.sce` for details on how to process the simulation results.

4 A comparison between some inverse kinematics algorithms

The examples that follow allow for a comparison of performance between some inverse kinematics algorithms presented in [7]. Simulation results obtained by using RTSS are validated by those discussed in the book.

Consider a three-link (RRR) planar arm whose link lenghts are $a_1 = a_2 = a_3 = 0.5$m. Let the arm be at the initial posture $\mathbf{q} = \begin{bmatrix} \pi & \pi/2 & \pi/2 \end{bmatrix}^T$rad, corresponding to the end-effector location: $\mathbf{p} = \begin{bmatrix} 0 & 0.5 \end{bmatrix}^T$m, $\phi = 0$ rad. A circular path of radius 0.25 m and center at $(0.25, 0.5)$ m is assigned to the end-effector. Let the motion trajectory be

$$\mathbf{p}_d(t) = \begin{bmatrix} 0.25\,(1 - \cos(\pi t)) \\ 0.25\,(2 + \sin(\pi t)) \end{bmatrix} \quad 0 \le t \le 4$$

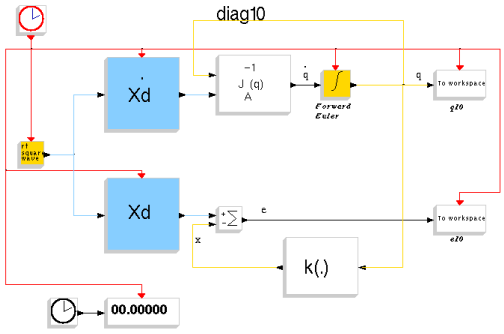

Figure 3: RTSS example `diag10.cos`

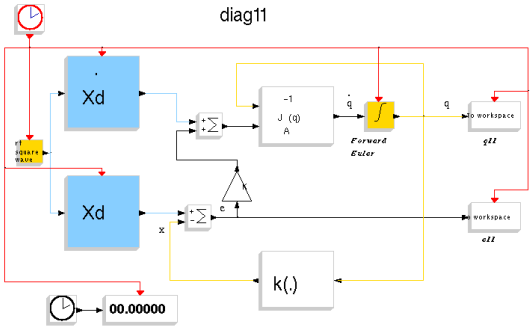

Figure 4: RTSS example `diag11.cos`

As regards end-effector orientation, initially it is required to follow the trajectory

$$\phi_d(t) = \sin\left(\frac{\pi}{24}t\right) \quad 0 \le t \le 4$$

The forward kinematics for this arm is computed by the `rt_fkine` block, while its Jacobian is returned from the `rt_jacob0` block by extracting the three nonnull rows of interest for the operational space. Note that, the `rt_jacob0` block returns the *geometric* Jacobian of the manipulator, whereas the algorithms that will be described in the following subsections utilize the *analytical* Jacobian since they operate on error variables (position and orientation) that are defined in the operational space. In general, these two Jacobians are different, but they coincide in this simple case of study and therefore the use of the `rt_jacob0` block is correct.

The inverse kinematics algorithms were implemented by adopting the Forward Euler numerical integration scheme provided by the `rt_fe` block, with an integration time $\Delta t = 1$ms.

4.1 Open-loop inverse Jacobian algorithm

At first, the inverse kinematics along the given trajectory has been performed with the Scicos model `diag10.cos` shown in figure 3. It integrates the joint velocity vector

$$\dot{\mathbf{q}} = \mathbf{J}_A^{-1}(\mathbf{q})\dot{\mathbf{x}}_d \qquad (1)$$

where $\dot{\mathbf{x}}_d = \begin{bmatrix} \dot{\mathbf{p}}_d^T & \dot{\phi}_d \end{bmatrix}^T$ is the time derivative vector of the assigned end-effector posture $\mathbf{x}_d$. The obtained results show that the norm of the position error along the whole trajectory is bounded; at steady state, after $t = 4$ sec, the error sets to a constant value in view of the typical drift of open-loop schemes. A similar drift can be observed for the orientation error.

This demo can be run by executing the corresponding Scilab simulation script `rt_demo10.sce`. To do this, just type the following command at the Scilab prompt:

```
exec <PATH>/demos/rt_demo10.sce;
```

4.2 Closed-loop inverse Jacobian algorithm

Next, the Scicos block scheme illustrated in figure 4 corresponding to the Jacobian inverse algorithm, that integrates the joint velocity vector

$$\dot{\mathbf{q}} = \mathbf{J}_A^{-1}(\mathbf{q})\left(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}\right) \qquad (2)$$

has been used, with the matrix gain $\mathbf{K} = diag\{500, 500, 100\}$. Thanks to the closed-loop feature of the scheme, as shown by the simulation results, the norm of the position error is radically decreased and converges to zero at steady state; the orientation error, too, is decreased and tends to zero at steady state.

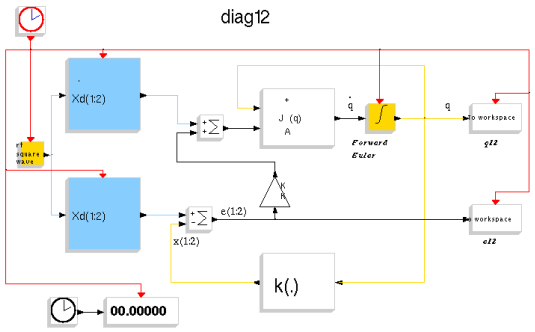

Figure 5: RTSS example `diag12.cos`

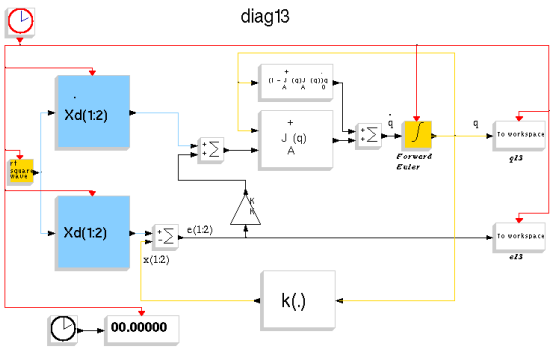

Figure 6: RTSS example `diag13.cos`

This demo can be run by executing the corresponding Scilab simulation script `rt_demo11.sce`:

```
exec <PATH>/demos/rt_demo11.sce;
```

4.3 Jacobian pseudo-inverse algorithm

If the end-effector orientation is not constrained, the operational space becomes two-dimensional and a redundant degree of mobility is then available. In the case of a redundant manipulator, equation (2) can be generalized into

$$\dot{\mathbf{q}} = \mathbf{J}_A^{\dagger} \left(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e} \right) + \left(\mathbf{I} - \mathbf{J}_A^{\dagger} \mathbf{J}_A \right) \dot{\mathbf{q}}_0 \qquad (3)$$

4.3.1 Unconstrained solution

The Scicos diagram in figure 5 implements (3), where the redundancy is not exploited ($\dot{\mathbf{q}}_0 = \mathbf{0}$) and $\mathbf{K} = diag\{500, 500\}$. The simulation results reveal that position tracking remains satisfactory and, of course, the end-effector orientation freely varies along the given trajectory.

This demo can be run by executing the corresponding Scilab simulation script `rt_demo12.sce`:

```
exec <PATH>/demos/rt_demo12.sce;
```

4.3.2 Solution with mechanical joint limit constraints

In order to show the capability of handling the degree of redundancy, a Scicos model based on (3) with

$$\dot{\mathbf{q}}_0 = k_0 \left(\frac{\partial w(\mathbf{q})}{\partial \mathbf{q}} \right)^T \qquad (4)$$

has been used; only one type of constraint has been considered concerning an objective function ($w(\mathbf{q})$ in (4)) to locally maximize. The objective function considered is the *distance from mechanical joint limits*, defined as

$$w(q_1, q_2, q_3) = -\frac{1}{6} \sum_{i=1}^{3} \left(\frac{q_i - \bar{q}_i}{q_{iM} - q_{im}} \right)^2 \qquad (5)$$

where q_{iM} (q_{im}) represents the maximum (minimum) joint range and $\bar{q}_i$ is the middle value of the joint range. Specifically, it is assumed what follows: the first joint does not have limits ($q_{1m} = -2\pi$, $q_{1M} = 2\pi$), the second joint has limit $q_{2m} = -\pi/2$, $q_{2M} = \pi/2$, and the third joint has limit $q_{3m} = -3\pi/2$, $q_{3M} = -\pi/2$. It is not difficult to verify that, in the unconstrained case, the trajectories of joints 2 and 3 violate the respective limits. The gain in (4) has been set to $k_0 = 250$. Simulation results show the effectiveness of the technique with utilization of redundancy, since both joints 2 and 3 tend to invert

their motion– with respect to the unconstrained trajectories obtained as in subsection 4.3.1 –and keep far from the minimum limit for joint 2 and the maximum limit for joint 3, respectively. Such an effort does not appreciably influences the position tracking error, whose norm is bounded anyhow within acceptable values.

This demo can be run by executing the corresponding Scilab simulation script `rt_demo13.sce`:

```
exec <PATH>/demos/rt_demo13.sce;
```

References

[1] S. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and Simulation in Scilab/Scicos*. New York, NY: Springer-Verlag, 2005.

[2] R. Bucher, S. Mannori, and T. Netter, *RTAI-Lab tutorial: Scilab, Comedi, and real-time control*, 2006.

[3] R. Nikoukhah and S. Steer, *SCICOS A Dynamic System Builder and Simulator User's Guide*, 2005.

[4] Scilab Group, "Introduction to Scilab–User's Guide," tech. rep., INRIA Metalau Project/ENPC Cermics, 2000.

[5] P. I. Corke, "A robotics toolbox for MATLAB," *IEEE Robotics and Automation Magazine*, vol. 3, pp. 24–32, Mar. 1996.

[6] P. I. Corke, *Robotics TOOLBOX for MATLAB (Release 7.1)*, Apr. 2002. robot7.1/robot.pdf.

[7] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Advanced Textbooks in Control and Signal Processing, London, UK: Springer-Verlag, 2nd ed., 2000.