Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

RobotiCad: an Educational Tool for Robotics

Riccardo Falconi * **Claudio Melchiorri** *

* *DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna Italy (e-mail: {rfalconi, cmelchiorri}@deis.unibo.it)*

Abstract: RobotiCad is a user-friendly Matlab/Simulink toolbox for the modeling and simulation of robotic manipulators. With RobotiCad, starting from Denavit-Hartenberg parameters, it is possible to create the kinematic and dynamic models of any serial mechanical structure, together with its 3D graphical model. When a robot is created, it can be exported in a dedicated file that can be loaded in a Simulink scheme and easily interfaced with other block-sets. A robot, then, can be simulated and, eventually, an *AVI* file of the simulation can be obtained. Moreover, a rich collection of Matlab functions properly developed in order to study industrial robots is included in RobotiCad, e.g. functions for trajectory generation, manipulability analysis, control, and so on.

Keywords: Robotic Education, Robot Simulation, Graphical User Interface, Matlab

1. INTRODUCTION

Because of the complexity of robotic systems, several simulation tools devoted to robotics have been developed and proposed for solving problems ranging from control, trajectory planning, design, programming and so on. Some of these tools are oriented to professional/industrial applications, while others are more specifically suitable for educational purposes. Many of them have been implemented for well defined problems and for defined classes of robots only, such as e.g. RoboOp (Gourdeau [2006]), RoboWorks (Newtonium© [2005]), or Easy-Rob (Anton [2005]). Some of them are stand-alone packages, while others have been created as open source tools, see e.g. GraspIt (Miller and Allen [2004]), or RoboMosp (Jaramillo-Boter et al. [2007]). Among the open source packages, several Matlab toolboxes have been developed, such as SimMechanics (Babuska [2005]), or the Robotics Toolbox (Corke [1996]), to our knowledge the first tool of this type.
The Robotics Toolbox, basically, provides a set of Matlab functions and Simulink blocks for the simulation of the direct and inverse kinematics and of the dynamic model of user-defined robots. Although it can be probably considered as the most popular robotics toolbox, it reveals some limits:

- the Corke's Simulink blocks are related to the current Matlab workspace;
- only a type of trajectory is implemented, that represents the only manner to move a robot;
- a graphical user interface (GUI) is not available, and robots are represented as a collection of segments;
- the inverse kinematic function does not take into account singularity points;
- a robot is simulated without any other object in its environment;
- a robot cannot interact with external objects (e.g. grasped objects).

In particular, the last point is critical if the simulation of a robot within an industrial environment is required. These limitations were the main motivations for the development of RobotiCad, the Matlab/Simulink toolbox described in this paper.

Robotics Toolbox	**RobotiCad**
Text line interface	User friendly 3D dedicated interface
Create dedicate object classes	Compatible with Corke's toolbox
No workspace object	Several different workspace objects
Workspace simulation without objects	Workspace objects included in Simulink model
Fifth order polynomial trajectories	Twelve different types of trajectories; dedicated script tool

Table 1. Comparison between Robotics Toolbox and RobotiCad.

The main differences between the Robotics Toolbox and RobotiCad are summarized in Table 1. It's worth to notice that some of the Corke's Matlab functions have been implemented and expanded in RobotiCad. Among them, all the functions allowing the definition of the robot and of the link object classes. This requirement was fundamental as we decided to keep software compatibility between models created in Robotics Toolbox and RobotiCad.

This paper is organized as follows. In Section 2 the main features of RobotiCad are introduced (the fundamental Matlab functions, the graphic user interface, the algorithms for the study of kinematics and dynamics, the RobotiCad Simulink Blockset library). An example of control implemented using RobotiCad features for an 8-DOF robot welding two perpendicular intersecting cylinders is discussed in Section 3. Section 4 concludes with final comments and plans for future activity.

2. ROBOTICAD

As shown in Fig. 1, the RobotiCad environment is composed of three fundamental modules:

- RobotiCad Matlab Functions.
- RobotiCad GUI.
- RobotiCad Blockset.

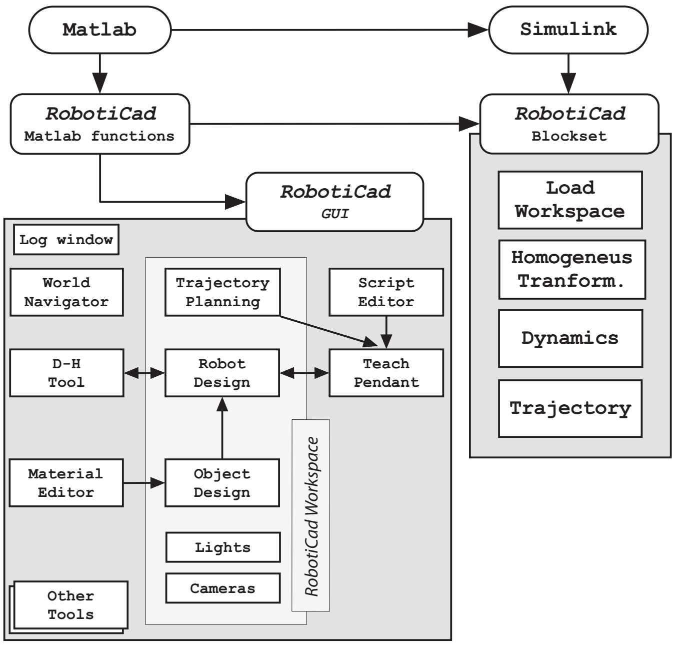


Fig. 1. RobotiCad functional blocks.

2.1 *RobotiCad Matlab Functions*

This module can be considered the core of RobotiCad. Many functions used by the RobotiCad GUI (see Sect. 2.2) and by the RobotiCad Simulink Blockset (see Sect. 2.3) are implemented as *m*-functions and can be used at the Matlab prompt. This module allows the user to handle homogeneous transformation and rotation matrices, the creation of robot objects, and the study of its forward and inverse kinematics and dynamics. Moreover, this module allows to study singularity configurations and to handle robot's Jacobian matrices. Although many of these functions are improvements of Corke's Robotics Toolbox functions, the real innovation introduced in this module is the possibility of generating workspace and joint-space trajectories.

2.2 *RobotiCad Graphic User Interface*

The GUI (Graphic User Interface) is probably the most important module of RobotiCad. By using it, the user can create a workspace, e.g. an industrial working cell, with many robots and objects to interact with. By typing "`>> RobotiCad`" at the Matlab prompt, the toolbox is started. The environment, shown in Fig. 2, is composed of three main windows, that are:

Log Window (a): this window contains a single text field that is automatically updated every time the user executes an action on the workspace or on an object in the scene. By means of the Log Window, an history of the executed actions in the current workspace is then available to the user.

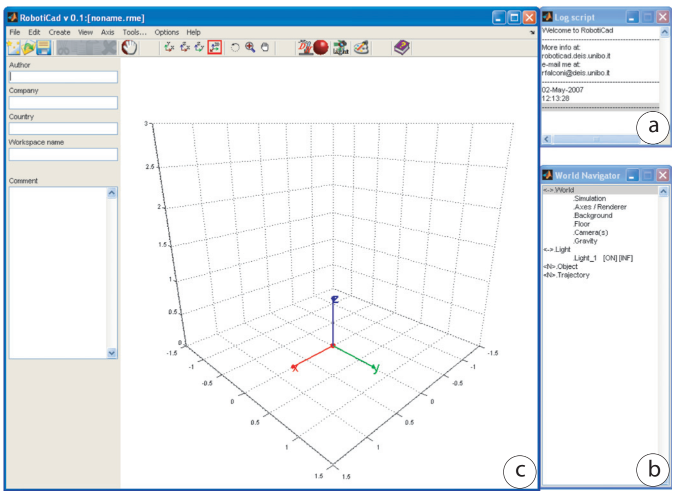
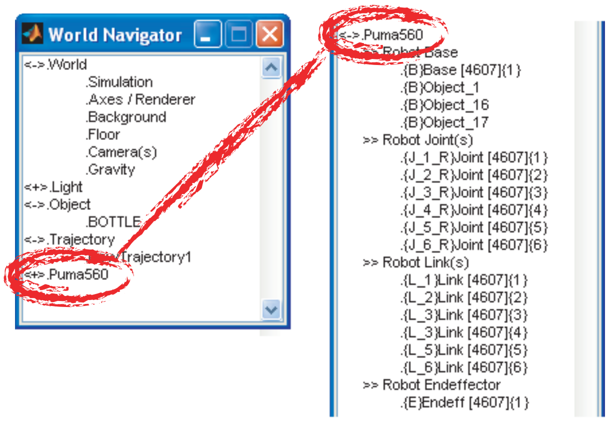


Fig. 2. RobotiCad main windows: a) Log Window. b) World Tree. c) RobotiCad Workspace.

World-Tree Window (b): the World Navigator Window presents the hierarchical structure of the objects in the workspace, automatically updated when objects are created or modified. As example, in Fig. 3 the structure of a Puma 560 robot is shown.

Workspace Window (c): robots, workspace trajectories and other objects can be created in this window. The user can also insert lights and cameras to improve the representation of the scene. An absolute reference frame $\mathcal{F}_0 = O_0x_0y_0z_0$ is present in the center of the new workspace.



World Navigator

```
<->World
        .Simulation
        .Axes / Renderer
        .Background
        .Floor
        .Camera(s)
        .Gravity
<+>.Light
<->.Object
        .BOTTLE
<->.Trajectory
        .Trajectory1
<+>.Puma560
```

```
<->.Puma560
   >> Robot Base
        .{B}Base [4607]{1}
        .{B}Object_1
        .{B}Object_16
        .{B}Object_17
   >> Robot Joint(s)
        .{J_1_R}Joint [4607]{1}
        .{J_2_R}Joint [4607]{2}
        .{J_3_R}Joint [4607]{3}
        .{J_4_R}Joint [4607]{4}
        .{J_5_R}Joint [4607]{5}
        .{J_6_R}Joint [4607]{6}
   >> Robot Link(s)
        .{L_1}Link [4607]{1}
        .{L_2}Link [4607]{2}
        .{L_3}Link [4607]{3}
        .{L_3}Link [4607]{4}
        .{L_5}Link [4607]{5}
        .{L_6}Link [4607]{6}
   >> Robot Endeffector
        .{E}Endeff [4607]{1}
```

Fig. 3. Example of World Navigator Window: Unimation Puma 560 and its subtrees.

In the *RobotiCad Workspace Window* the user can create its workspace with objects, robots, 3D workspace trajectories, lights and cameras.

In order to create a new robot, the user can use the Denavit-Hartenberg (DH) parameters or load a Corke-robot model if saved in a Matlab file. After defining the DH parameters, the user can fix the position and the orientation of the base of the robot base w.r.t. $\mathcal{F}_0$. Once all the parameters have been defined, the robot can be exported in the RobotiCad workspace. As a new robot is created, it is characterized by four categories of objects: *base*, *joint*, *link*, and *endeffector*. Each category may contain several objects, and their dynamic properties (i.e. mass and inertia matrix) can then be specified. As default, each category contains an object with default properties. By using the *World-Tree Window*, a workspace object can be moved to a robot's category and viceversa. In Fig. 4 a crane manipulator moving a box along a workspace trajectory is shown. Once the robot is created, all its properties can be analyzed (i.e.: Jacobian matrix, manipulability measures, force and velocity ellipsoids). Moreover, it can be programmed by using a virtual teach pendant (for joint-space trajectories), by using the *Script Editor* tool, or by assigning to the manipulator a workspace trajectory previously created. The Trajectory Planning module allows to

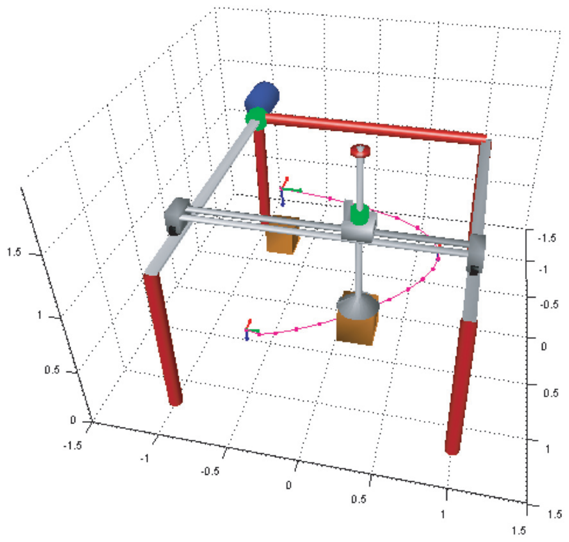


Fig. 4. A Crane moving a box along a 3D workspace trajectory.

define both Cartesian and Joint-Space trajectories. Many of the trajectories available for joint space motions can be found in Melchiorri [2000]. Cartesian trajectories can be specified in three ways:

- by using one of the predefined trajectories implemented in RobotiCad, such as circle and straight line;
- by the definition of a collection of via-points with specified position and orientation;
- by loading a collection of via-points from a text file **.txt* containing the trajectory description.

As example, in Fig. 5 some workspace trajectories are shown.

For planning Cartesian trajectories in RobotiCad, several interpolation methods have been implemented (the default is based on cubic splines). Moreover, RobotiCad allows the user to combine together trajectories from workspace and joint space in order to achieve complex behaviors. To conclude, each workspace trajectory is associated with a semi-transparent bounding box (that suggests to the user the space covered by it) and a fixed frame positioned in the center of the trajectory. In this manner, scaling, rotation, and translation operations can be applied to the whole trajectory. Independently from the method chosen to program the robots, the obtained trajectories can be saved in a file that can be loaded in the RobotiCad

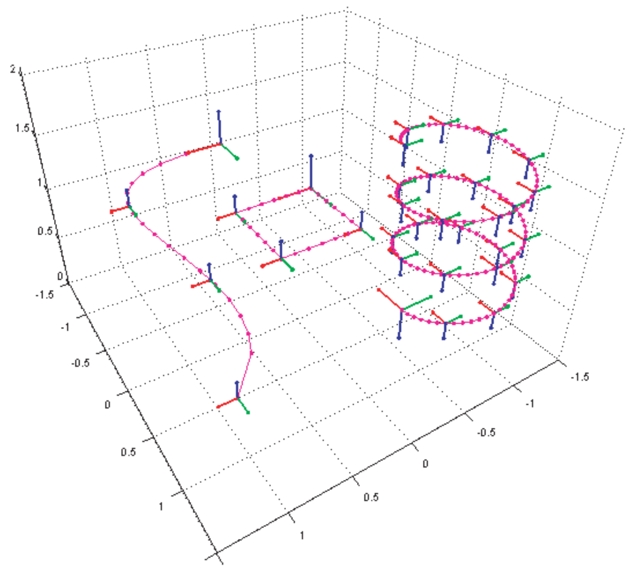


Fig. 5. Examples of workspace trajectories.

GUI and in a Simulink model using a dedicated block (see Section 2.3). During the execution of a movement, RobotiCad gives also the opportunity of controlling if the moving robot collides with one or more objects in the workspace.

Kinematic and Dynamic Algorithms The position and orientation in space of each object is provided by a reference frame rigidly connected with the object itself. Since the DH parameters are used to describe robot configurations, the position and orientation of each object in space results from a proper multiplication of homogeneous matrices. For the i-th joint (and related objects), the matrix providing position and orientation in a specified configuration is:

$$\mathbf{B}\ \mathbf{A}_1^2(q_1)\ \ldots\ \mathbf{A}_{i-1}^i(q_i) = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

where $\mathbf{B}$ is the homogeneous matrix used to define the position and the orientation of the base of the manipulator w.r.t. $\mathcal{F}_0$, $\mathbf{R}$ is a rotation matrix and $\mathbf{p}$ is a vector.

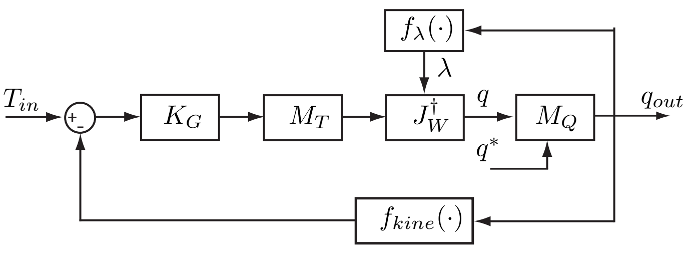


Fig. 6. Scheme of the inverse kinematic algorithm.

The *inverse kinematic algorithm* is based on a modified calculus of the pseudo-inverse of the Jacobian matrix (see Kelmar and Khosla [1988]). The general scheme to compute the robot's joint configuration starting from an homogeneous matrix or from a sextuple of elements like [X_{pos}, Y_{pos}, Z_{pos}, *Roll*, *Pitch*, *Yaw*] is shown in Fig. 6, where:

- T_{in} is the homogeneous matrix for the desired end-effector position and orientation, and q_{out} is corresponding joints configuration;
- K_G is a matrix gain that can be set by the user, whose value is critical for the convergence of the algorithm; the *Inverse Kinematic block* (see Section 2.3) allows to set the value of K_G (the default value is 1) during simulation;
- $J_W^\dagger$ is the weighted pseudo-inverse of the Jacobian matrix computed as
$$J_W^\dagger = J^T \left(JJ^T - \lambda\right)^{-1},$$
where J is the Jacobian matrix of the manipulator in the current configuration and λ is a weight matrix defined in Kelmar and Khosla [1988];
- M_T and M_Q are two filters customable by the user in order to mask some positions or directions in the workspace (M_T) or to lock some joints to a defined value (M_Q). In particular, the first filter can be used to improve algorithm performances for under-actuated robots (i.e.: for a planar robot, we can mask $[Z_{pos},\ Pitch,\ Yaw]$); the second filter can be used to select only proper subsets of the joints for the inverse kinematic function, leaving the possibility for the remaining joints to be programmed directly in joint space.

The dynamic algorithm used in RobotiCad is based on the standard Newton-Euler method and consists of a recursive algorithm to define the torques applied to the endeffector. For more details, see Sciavicco and Siciliano [1996].

2.3 RobotiCad Blockset

The second main part of the RobotiCad toolbox is a Simulink blockset collection, called RobotiCad Blocks. In Fig. 7 a snapshot of this library is shown.

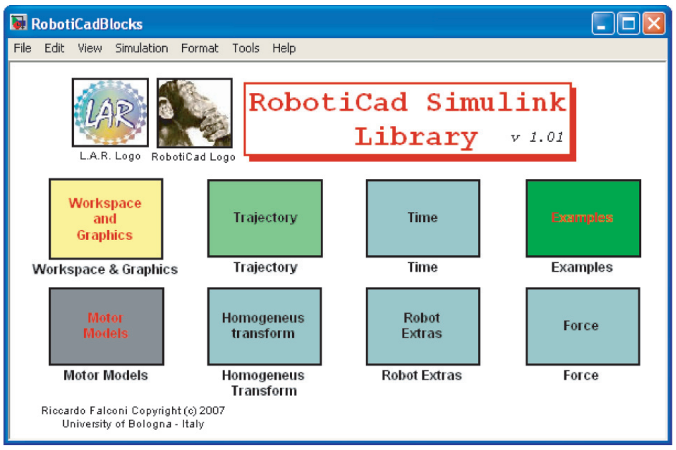
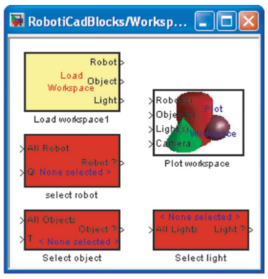


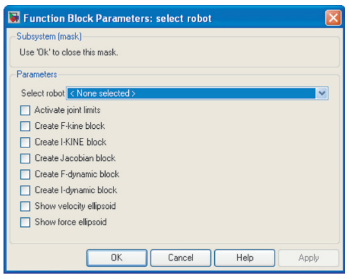
Fig. 7. RobotiCad Library for Simulink.

These eight fundamental blocks are summarized in the following list. Their properties are explained in the next Section with a case study.

Workspace and Graphics This main block, shown in Fig. 8(a), contains all the blocks that allow the user to interface the model created in the RobotiCad GUI with a Simulink environment. In particular, the *Select*



(a) Workspace and Graphics


Function Block Parameters: select robot

Subsystem (mask)

Use 'Ok' to close this mask.

Parameters

Select robot < None selected >

- ☐ Activate joint limits
- ☐ Create F-kine block
- ☐ Create I-KINE block
- ☐ Create Jacobian block
- ☐ Create F-dynamic block
- ☐ Create I-dynamic block
- ☐ Show velocity ellipsoid
- ☐ Show force ellipsoid

OK Cancel Help Apply

(b) *Select Robot* mask

Robot (see Fig. 8(b)) block allows to choose the robot to work with and, by a simple tick in a checkbox, can automatically create all the blocks that implement the fundamental functions, such as forward and inverse kinematics and dynamics, and so on.

Trajectory Contains all the blocks dedicated to the trajectory generation. The user can choose between many kinds of trajectories, or load the **.txt* files created by the RobotiCad GUI containing workspace trajectories. By means of these blocks, the user to combine many different trajectories to create complex motion profiles.

Motor Models Some basic motor models, such as DC motors and so on, are collected here. The user can add new models.

Homogeneous Transformation Contains all the blocks to handle homogeneous transformation.

Robot Extras Once the user selects a robot, the blocks collected here allow the computation of the manipulability values, of the Jacobian matrix, of singularity conditions.

Force Contains two blocks to define a force vector and to load the gravity vector.

Time Contains two blocks that can be programmed as switches and that can be used to combine trajectories.

Examples As indicated by the block name, this block contains some examples of robotic systems.

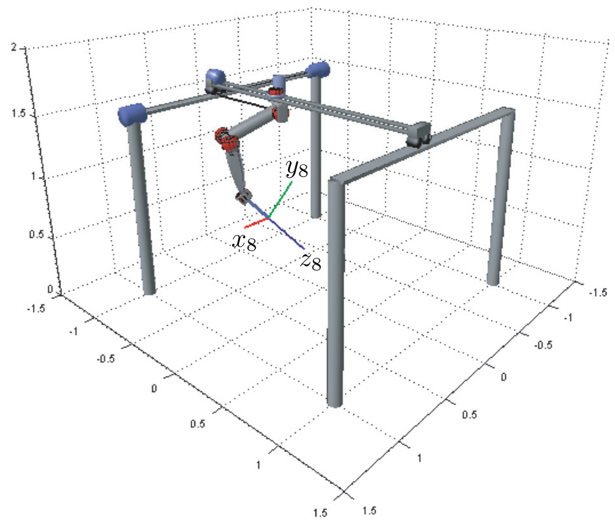


Fig. 8. The case study robot.

3. CASE STUDY

In this Section, in order to show how to create and simulate an industrial robot cell within RobotiCad, a case study is reported. In particular, an 8 DOF robot composed by a 2 DOF planar arm moving on the XY plane and a Unimation Puma 560 (see Fig. 8) is considered. The DH parameters are reported in Table 2. In the simulation, this

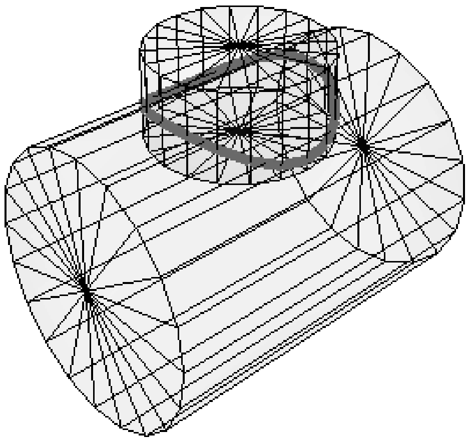


Fig. 9. Two orthogonal cylinders.

8 DOF robot is used to weld two orthogonal and intersecting cylinders. The robot's tool trajectory is specified in the workspace, and the inverse kinematic algorithm is used to obtain the corresponding joints configurations. The workspace trajectory is described analytically and a subset of via-points is used and interpolated for the robot programming. Of course, for this kind of task, it is not necessary to use a manipulator as complex as this one. The dynamic parameters are not reported for the sake of brevity and because they are well known in the literature (see Armstrong et al. [1986]). The position of the robot

Table 2. DH Welding Robot Parameters.

	α_i	a	θ_i	d	R/P
$Joint_1$	$\frac{\pi}{2}$	0	0	d_1	1
$Joint_2$	$-\frac{\pi}{2}$	0	$\frac{\pi}{2}$	d_2	1
$Joint_3$	$\frac{\pi}{2}$	0	θ_3	0	0
$Joint_4$	0	0.43	θ_4	0	0
$Joint_5$	$-\frac{\pi}{2}$	0.02	θ_5	0.15	0
$Joint_6$	$\frac{\pi}{2}$	0	θ_6	0.43	0
$Joint_7$	$-\frac{\pi}{2}$	0	θ_7	0	0
$Joint_8$	0	0	θ_8	0	0

base in $\mathcal{F}_0$ is $[-1,\ -1,\ 1.5]^T$, while its *Tool* has an offset $d_t = 0.5$ along the z_8 endeffector axis.

3.1 Trajectory Definition

The trajectory followed by the robot's tool is shown in Fig. 9. To obtain the requested path, one can start from considering the locus obtained by intersecting two orthogonal cylinders, with radii $r_1 = 0.4$ and $r_2 = 0.75$ respectively. This locus is described by:

$$p(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} r_1 \cos(t) \\ r_1 \sin(t) \\ \sqrt{r_2^2 - r_1^2 \sin^2(t)} \end{bmatrix},$$

where $t \in [0 \dots 2\pi]$.

To define the Frenet frame associated to each via-point, $p(t)$ is derived:

$$\dot{p}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} -r_1 \sin(t) \\ r_1 \cos(t) \\ -\frac{r_1^2 \sin(t)\cdot\cos(t)}{\sqrt{r_2^2 - r_1^2 \sin^2(t)}} \end{bmatrix},$$

$$\ddot{p}(t) = \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ \ddot{z}(t) \end{bmatrix} = \begin{bmatrix} -r_1 \cos(t) \\ -r_1 \sin(t) \\ \frac{r_2^2(\sin^2(t) - \cos^2(t)) - r_1^2 \sin^4(t)}{\sin(t)\cos(t)} \end{bmatrix},$$

By normalizing $p(t)$, $\dot{p}(t)$, $\ddot{p}(t)$, one obtains $\bar{p}(t) = \frac{p(t)}{|p(t)|}$, $\dot{\bar{p}}(t) = \frac{\dot{p}(t)}{|\dot{p}(t)|}$, $\ddot{\bar{p}}(t) = \frac{\ddot{p}(t)}{|\ddot{p}(t)|}$ that represent the unit vectors of the Frenet frame parametrized in $t \in [0 \dots 2\pi]$.

Now, by considering the 8 DOF robot, one can see that the robot's *approach* vector and z vector of the Frenet frame differ of π radians. To get the via point that describe the trajectory, one can get the Frenet frame for $t = i \cdot \frac{2\pi}{N}$, with $i = 0 \dots N$, and apply a rotation about z of an angle $\theta = \pi$. In our case, the value $N = 20$ has been chosen to get a low position error using cubic spline interpolation (see Table 3). The resulting workspace trajectory is shown in Fig. 10.

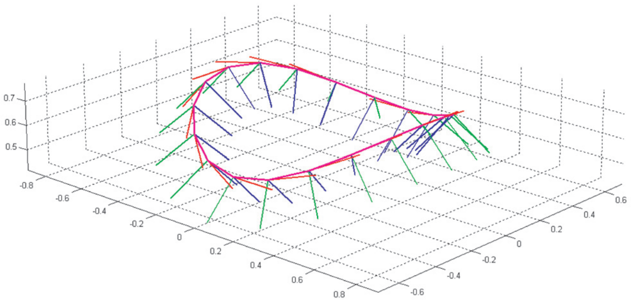


Fig. 10. Resulting trajectory with 20 via-points.

Table 3. Number of trajectory via points versus approximation error.

Number of via points	Error %
3	2.4039
5	0.0882
20	$7.4488 \cdot 10^{-5}$
100	$5.7429 \cdot 10^{-13}$

3.2 Robot Control Scheme

The control scheme used in this example is the well known *PD with gravity compensation*. The overall scheme is reported in Fig. 11. In Fig. 12 the analogous scheme created with the RobotiCad Simulink Library is shown. Note that in this scheme, a *Random Number* generator has been introduced to simulate the fact that the gravity term computed by the control algorithm in general does not match the real value. Moreover, note that the reference signal is created with two different blocks: the first one loads the **.txt* file containing the 20 via points used for the trajectory interpolation, the second one generates a joint space trajectory. In fact, the *Inverse Kinematic Block*

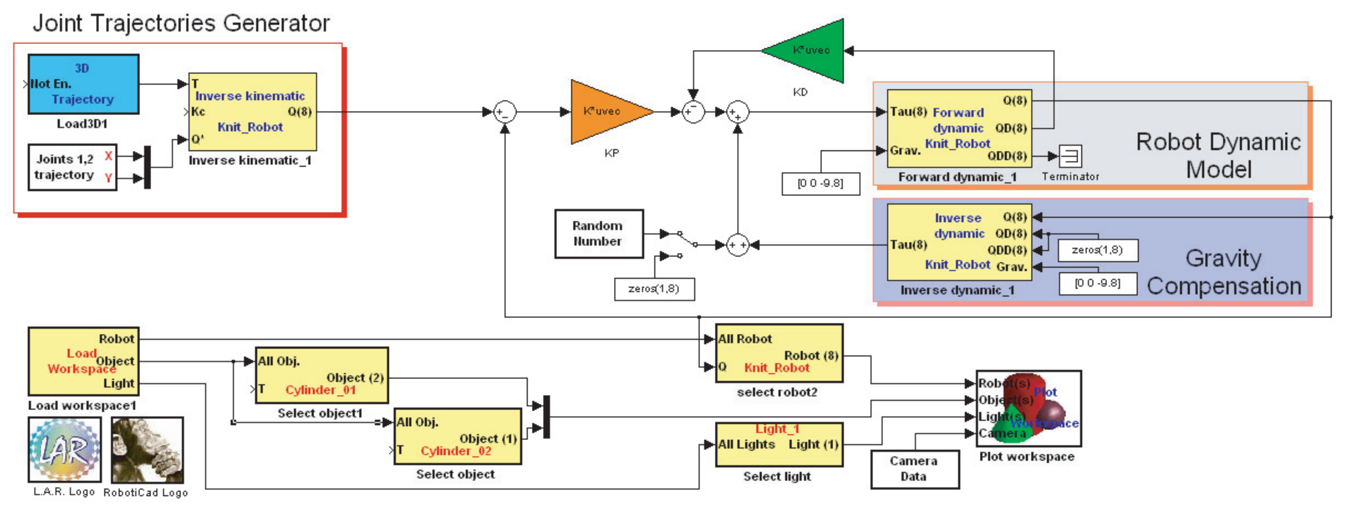


Fig. 12. PD with gravity compensation control scheme implemented with RobotiCad Simulink Library.

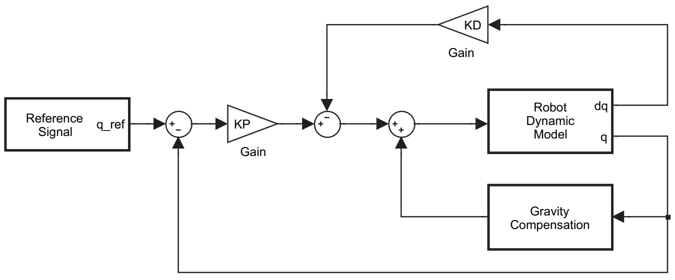


Fig. 11. PD with gravity compensation control scheme.

is used to get joints trajectory reference for joints 3 . . . 8, while the first two joints are moved by a circular trajectory specified directly in joint space. This allows to move the Cartesian part of the structure on a circle, while the tool of the robot is always maintained orthogonal to the intersection of the two cylinders.

4. CONCLUSIONS AND FUTURE WORK

In this paper RobotiCad, a new robotic toolbox for Matlab/Simulink, has been presented. It is based on a rich collection of Matlab functions, and can be used as a tool for educational purposes or industry robot prototyping and control.

The user-friendly interface allows to model a robot by specifying its Denavit-Hartenberg matrix and its physical properties, and to add objects to the environment by creating or loading them from a library. Moreover, the manipulator can be programmed in different ways and simulated by means of new Simulink blocks. An *AVI* file can be created for each simulation session.

The RobotiCad toolbox is available on the web at:

`http://www.roboticad.deis.unibo.it/`

Here, several examples and a library of robots (such as SCARA, Unimation Puma 560, Stanford Arm and so on) can be found.

Currently, new functions are being added to RobotiCad, such as new Simulink blocks and the possibility to include parallel manipulators and composed robots.

REFERENCES

S. Anton. Easy-Rob, 3D Robot Simulation Tool. *www.easy-bot.com*, 2005.

B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the puma 560 arm. *IEEE International Conference on Robotics and Automation.*, 3:63–73, 1986.

R. J. Babuska. Matlab Design Environment for Robotic Manipulators. *IFAC 2005*, 2005.

Peter I. Corke. A robotics toolbox for Matlab©. *IEEE Robotics and Automation Magazine*, 3:24–32, 1996.

R. Gourdeau. Roboop, a Robotics Object Oriented Package in c++. *http://www.cours.polymtl.ca/roboop/*, 2006.

A. Jaramillo-Boter, A. Matta-Gomez, J. F. Correa-Caicedo, and W. Perea-Castro. Robomosp. *IEEE Robotics and Automation Magazine*, 6:63–73, 2007.

L. Kelmar and P. K. Khosla. Automatic generation of kinematics for a reconfigurable modular manipulator system. *IFAC 2005*, 1988.

C. Melchiorri. *Traiettorie per Azionamenti Elettrici.* Esculapio Ed, Bologna, 2000.

Andrew T. Miller and Peter K. Allen. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Rob. and Autom. Magazine*, 2004.

Newtonium©. Roboworks, a Tool for Real Time Interactive 3D Modelling and Animation with Distributed Simulation. *http://www.newtonium.com/*, 2005.

L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators.* McGraw-Hill, New York, 1996.