ROBOMOSP

ROBOtics Modeling and Simulation Platform

Andrés Jaramillo Botero, Antonio Matta Gómez, Juan Fernando Correa Caicedo, Wilber Perea Castro

The ability to represent the behavior of complex systems through accurate models has become an indispensable tool in their design, evaluation, and operation. At the same time, the process of optimizing and fine-tuning mathematical models has been increasingly aided by the advent of high-performance computational environments. Analytical models are used to characterize and optimize their performance via closed-form solutions, numerical solutions, or, in some cases, solutions inferred through observation. Of course, the ability of a particular mathematical construct to reflect reality is dependent on what expression of reality one wants to model with it; nonetheless, computational experiments permit a higher degree of parametrization of physical phenomena without the expense or dangers of a flawed outcome.

Robotics manipulators are highly complex systems; consequently, the development of computational platforms that allow for their precise modeling, and close to real-life simulation of their behavior, constitute a fundamental tool for robot designers, users, and students of the field. This reason has inspired the creation of numerous graphical software environments, from nonrobot-manufacturers, such as: RoboWorks (1], Robot Assist (2), Easy-ROB3D (3), WorkspaceS (4), and also from the manufacturers themselves.

However most of these packages are expensive, highly proprietary, and usually limited in the amount of available features (i.e., scarce aid for engineering analysis or for user-defined functionalities).

This article describes the design and development of the modeling and simulation environment for robotic manipulators named *ROBOMOSP* (ROBOtics MOdeling and Simulation Platform), which addresses important limitations of existing software for this purpose, under a highly parametric interface ideal for academics and research. ROBOMOSP is a multiplatform three-dimensional (3-D) computer-aided-design (CAD) system (cross-compatible between the Linux, MacOSX and MS Windows operating systems) implemented with open-source [5] tools. This environment allows users to model individual physical components or compound elements, full robotic manipulators with user-defined kinematics configurations, discrete and continuous spatial trajectories, multisystem workspaces composed of several robots and other static or dynamic elements [e.g., computer numerical control (CNC) machining tools], and complex motion tasks involving the interaction of all components within a specified workspace. Furthermore, ROBOMOSP includes embedded engineering analysis tools for dynamic and kinematic simulation and control, a high-level language for ofline programming, a built-in application interface (API) for added user functionality, and a simulation engine capable of remote operation.

The "General Overview" section describes the main aspects of the methodology employed for the development of ROBOMOSP and the overall software components of the platform. "Software Design and Implementation" focuses on the high-level software architecture, the user interface, and implementation specifics. "Modeling" describes the software modules that deal with the construction, processing, and rendering of solids and articulated multibodies, the trajectory planning constructs, and the workspace or "world" environment. In dealing with import/export functions for user- designed components, the design team of ROBOMOSP established a very light and expressive functional grammar that permits rapid interpretation with very little storage requirements and increased portability; file formats for user-designed input/output (I/O) are covered in "File Formats (Input/Output)." In order to allow the user to interact with ROBOMOSP at the design and simulation levels, a special API denoted as *IOCI* (*Input Output Command Interface*) was embedded to provide a rich stack of commands that enhances user control of the environment; this is the topic of "Interacting with the Platform Via the IOCI." In addition to the aforementioned API, ROBOMOSP incorporates a high-level, task-oriented robot language compiler, derived from a subset of the International Robot Language (IRL) [6], for offline programming of robotic manipulators and a visual programming interface for online programming ("Offline and Online Programming"). "Simulation and Control" describes the implementation of embedded solutions for both the kinematics and dynamics problems in their forward and inverse regimes. ROBOMOSP's soundness for modeling and simulating realistic systems is demonstrated in "Validation of ROBOMOSP's Modeling and Simulation Capabilities" through several test cases. Last but not least, "Final Observations" includes final comments about the developed platform, and "Future Work" explores upcoming additions under current development.

General Overview

The ROBOMOSP system is composed of the following hierarchical subsystems: a graphical user interface (GUI), a 3-D modeling subsystem, a 3-D simulation subsystem, a robotics subsystem, and the IOCI API used as a layer between ROBOMOSP and external applications.

Because the ROBOMOSP construction process has had an evolutionary approach [7], the spiral model based on components [8] has been used as the development methodology. This methodology allows the design and implementation of software systems, whose functionalities are continuously being improved or added, by following a number of cycles, previously defined, which are attached to the main development spiral. To guarantee a proper and consistent growth, a series of connectivity and organization rules must be observed [9]; these are detailed in the system's software architecture.

Software Design and Implementation

Design

All of the ROBOMOSP subsystems are made of functional blocks that represent some of the high-level operations and utilities provisioned. A functional block is formed by a group of basic data entities called *components* [10]. Figure 1 shows how these functional blocks are connected and how the data flow through them.

There is also a hierarchical subordination among all of the ROBOMOSP subsystems. Because of this, the software architecture has a stratified order: it is divided into layers (onion-like) according to the dependency present among the components that belong to each functional block.

The central layer is in charge of basic operations (e.g., memory allocation, structures initialization), and all the components belonging to the external layers can access this layer directly. The data processing layer (second from central) stores all the components that belong to the robotics subsystem (engine); these components perform the mathematical calculations required for solving all the robotics-related problems before sending these to the 3-D simulation subsystem. The application layer (third from central) provides the 3-D modeling services and a communication channel with the robotics subsystem. The GUI layer (outer shell) gathers all the components that allow a graphical interaction between the end user and ROBOMOSP's internal subsystems.

Graphical User Interface

The GUI (see Figure 2) was built to simplify interactive modeling and simulation of all the elements belonging to the system's functional blocks. It is integrated by five main windows.

1) **Object navigator window:** It allows access to the data stored in both the 3-D modeling and the 3-D simulation subsystem's components (materials, CSGs [Constructive Solid Geometry], objects, robots, trajectories, and worlds) via common operations such as open, save, save as, delete, reload, edit, or memory checks over such data. The components displayed in this window are immediately drawn in the 3-D rendering display window.
2) **Properties palette window:** In this window, the properties of the elements displayed in the object navigator window are shown. The end user can modify such properties through this window.
3) **Display window:** This window is used to visualize and manipulate in three dimensions—using an ample array of controls from the provided toolbar—any element present in the object navigator window.
4) **Messages window:** Provides real-time messages of the system's status (including error logs and successful executions).
5) **Console window:** The commands belonging to the IOCI API are executed through this window, directly or through script files. External applications can interact with ROBOMOSP's internal subsystems, both locally or remotely, using sockets connected to the IOCI.

Implementation

Open source tools and libraries that comply with the GNU public license [11] were used to implement the ROBOMOSP system. For performance and portability purposes, most of the low-level processing code was written in ANCI C, while all the GUI was implemented using the Tcl scripting language and its graphical library Tk [12]. The 3-D visualization window implementation lays over the Togl TCL/TK widget; in this window all of the systems' 3-D solids are represented using the OpenGL library. The meshach library [14] was used for linear algebra calculations, and the NetGen [15] library was used for automatic generation of polygonal meshes from a CSG tree (described in the following section).

Modeling

As its name implies, the 3-D modeling subsystem performs all the platforms' 3-D modeling. The tasks supported are: 3-D solids design and parametrization (position, orientation, and rendering surface characteristics); creation and manipulation of materials; coloring, ambient, and local light specification; 3-D solids visualization schemes (wireframe, filled, flat shaded, and smooth); and isometric and perspective rendering views ports.

The 3-D solids construction is accomplished following a bottom-up order that depends on the complexity of what is being built. Consequently, the construction process starts with the design of CSG solids (geometric and material parameters are defined) and continues with the integration of CSG compound objects that are later assembled into full 3-D robotic manipulators or other workspace objects (mechanical systems).

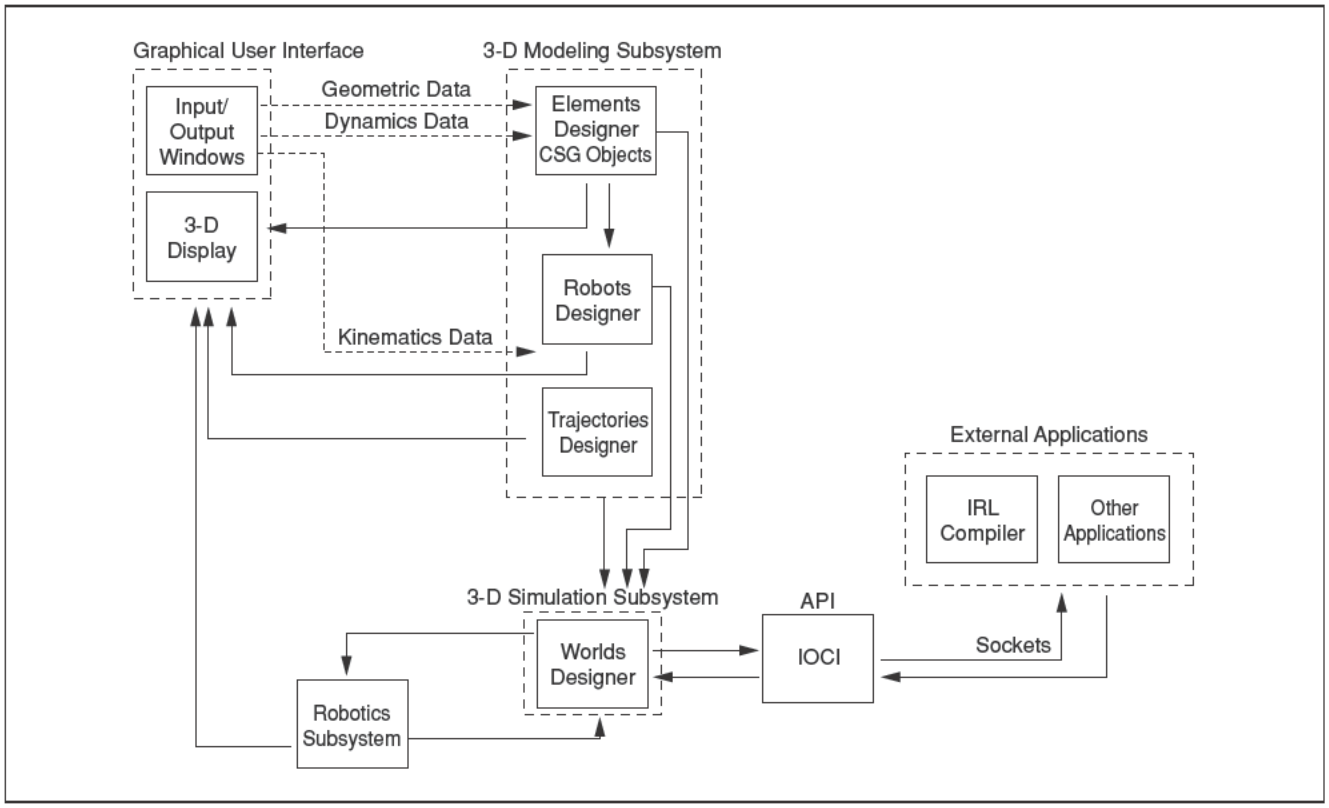

Figure 1. *ROBOMOSP's functional blocks.*

Cartesian trajectories are also modularly designed or assembled before being incorporated, along with other objects, to the 3-D simulation workspace, denoted simply by "world" (see Figure 2). The user can modify objects in any level and at any time; the system performs automatic updates on all subsequent levels.

Geometric Modeling of 3-D Rigid Solids

ROBOMOSP uses a hybrid modeling environment that combines both the CSG technique to define geometrical properties and the 3-D solids mesh representation to visualize 3-D rigid solids in real time.

The CSG technique allows the modeling of complex objects using the boolean operations: union, intersection, and difference. These boolean operations are applied to a set of basic primitives that are the branches of an *n*-ary tree [16]. This technique simplifies the interactive modeling of complex solids, profiting from the advantages that a GUI can offer, and allows for the representation of an almost unlimited number of 3-D objects [17]. Furthermore, the portability and simplicity of parametric objects defined with CSG permit a simple and effective file format for systems defined under the ROBOMOSP platform. On the other hand, real-time spatial transformation of compound CSG-rendered objects are computationally expensive even when their corresponding trees are fully normalized and optimized, making polygonal meshes a better means for rendering under such conditions.

A polygonal mesh defines a 3-D solid surface with a set of triangular polygons, B-rep representation, and it is used to render the solid represented with a CSG tree. This mesh is obtained using the mesh generation functions of the NetGen library [15].

For localizing the rigid solids in Cartesian space, the ROBOMOSP system uses a structure called *Homogeneous Graphics Library Matrix Transformation* (*HGLMT*), which stores both the position and orientation matrices referenced to a local or global inertial frame for each object.

Modeling Rigid Body Assemblies with Mass Properties

Once the geometrical-graphical representation of the elements that constitute a rigid solid is obtained, the next step involves calculating its mass properties (properties required for solving both the forward and inverse robot dynamics problems; see "Dynamics Simulation Control"). The object component in ROBOMOSP allows the assembly of parts into compound objects and provides the means for computing the corresponding mass properties with respect to the object's center of gravity.

The mass properties of each graphical element are calculated using the polygonal mesh generated by the CSG component and the user-defined material density property. This representation of vertices and faces approximates the 3-D solid with a polyhedron of uniform density. The calculation of the mass properties (i.e., mass, center of mass, and tensor of inertia) is achieved from the mesh representation using the method proposed by [19]. The tensor of inertia components (moments and products of inertia) are calculated with respect to the center of mass of the object, measured from the solid's CSG coordinate frame origin.

Each one of the CSG graphical elements is a node in an *n*-ary tree that represents and relates the solid's location within the inertial reference frame. In turn, each node has its mass parameters reference to the local coordinate system. The entire object is represented by its mass parameters and the *n*-ary tree that defines its parts and location.

A compound objects' mass is defined from the aggregate masses that make up the assembled object and from its corresponding tensor of inertia, computed via the parallel axes theorem.

Special treatment is given to overlapping CSG solids to avoid redundant volumes in the calculations.

The advantage of calculating an object's mass parameters from a polyhedron representation comes from the speed of computations. The tradeoff is a tolerable penalty in terms of accuracy for the estimation of solid volumes; this in turn affects the calculation of mass properties, which are certainly dependent on the geometrical complexity of a particular object (see "Validation of ROBOMOSP's Modeling and Simulation Capabilities" for a numerical example). There are two main sources of error for single primitive objects (objects being a disjoint union of uniform density, with a boundary representation): numerical errors from the algorithm itself and approximation errors inherent to the geometric model (since object surfaces are approximated from simple polyhedrons). The mass properties for all the elements of an object are computed in a straightforward manner by traversing the *n*-ary tree that defines it. This allows a user to interactively monitor and update the mass properties of a complex assembly. For compound, nonsymmetric objects, an additional source of error comes from the summation of individual volumetric errors and from the computation of the parallel axis theorem.

Robot Modeling

This module uses the objects previously modeled and stored in the parts database to define a robot's base, its links, and joints from its corresponding kinematics configuration.

The base objects reference to the world coordinate reference frame (inertial reference). The kinematic configuration of a robot is defined in terms of its Denavit-Hartenberg (DH) parameters [20] or the modified DH convention [21], the type of

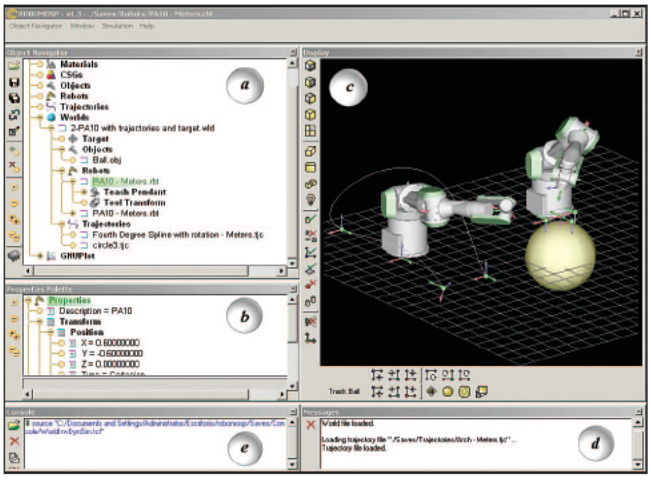

Figure 2. *ROBOMOSP graphical user interface.*

joints (rotational or prismatic), their range of motion (maximum and minimum values), and a visibility option (which allows for multiple joint definitions relative to a single-frame origin). Each joint has an associated coordinate frame attached to it.

Trajectory Modeling

The trajectories construction module is designed to allow modeling of Cartesian paths with parametric constraints. The time, velocity, and motion-type variables, with which a trajectory is executed, are defined by the end user. This module handles the definition of trajectories in both Cartesian and joint spaces. In joint space, ROBOMOSP allows the specification of PTP trajectories, with initial and final control points, as well as the incorporation of path constraints. Since joint trajectories depend on the particular robot kinematics, these may only be defined and visualized in the interface once the "robot-trajectory" association is made explicitly by the user.

For modeling continuous Cartesian trajectories in ROBOMOSP, several analytical functions and polynomial interpolation methods are available; all of which can be used in any combination. Analytical functions geometrically constrain the robot's motion to straight lines, circumference arches, and circumferences. Within these, the user is permitted to define parametric kinematic motion variables, including: constant velocity, constant acceleration, and curves with trapezoidal velocities, among others. Polynomial interpolation trajectories are fitted to the critical control path points (meeting desired position, time, and the velocity conditions) for a particular task specification. Parametric curves implemented via Bezier curves and third- (clamped or natural) and fourth-order splines [23] join all control points defined by the user in a continuous and smooth way. Bezier parametric curves have been modified to allow the specification of motion control parameters [22] such as time, initial velocities, and final velocities. In every case, the user is allowed to define both position and orientation, a useful feature for establishing orientation changes in a tool control path (TCP).

ROBOMOSP supports the definition of complex tasks that require the construction of trajectories made up of different primitive paths in any combination of analytical and interpolated functions. This is achieved via the definition of *segments* in a trajectory's definition. Each segment of a complex trajectory can be built independently; this allows for modularity and high reutilization of stored trajectories.

When a complex trajectory is being built, consecutive segments may be spatially sparse, or they may coincide in their positions but not in their orientation values, or their translational or angular velocities may present spatial discontinuities. To avoid these problems, ROBOMOSP automatically calculates a smooth transition between consecutive segments. The end user has the possibility to change these parameters in order to adjust them to the physical restrictions imposed by the particular robotic manipulator hardware under use (e.g., individual joint velocity, acceleration, or torque limits).

The user-designed spatial trajectories can be visualized in three dimensions on the display window or plotted versus time using the embedded GNUPlot [24] toolbox (see Figure 3).

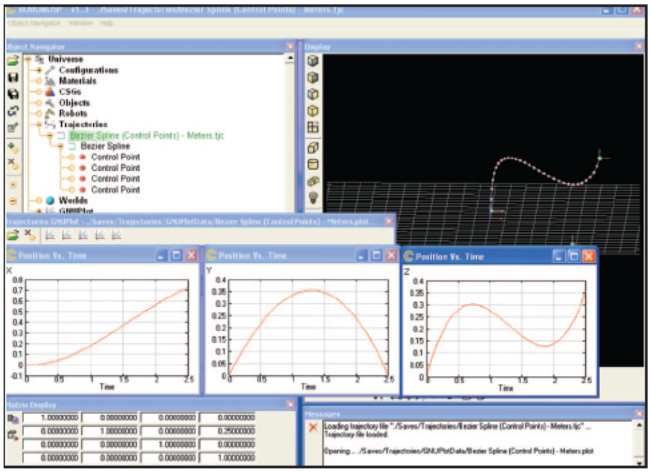

Figure 3. *Visualization of a trajectory under construction and its two-dimensional Cartesian projection plots (GNUplot).*

Modeling Complex Worlds

ROBOMOSP allows the design and modeling of complex simulation workspaces denoted within the environment as *worlds*. In each simulation world, the end user defines all the objects, robots, and trajectories that make up a specific workspace before performing any analysis. ROBOMOSP uses every element belonging to a simulation world in its parametric form in order to establish the corresponding spatial relations among them and to speed up the calculations performed during actual simulation time by using a hashed list that indexes all the elements of a simulation world.

The parametric robot has an associated structure denoted as *teach pendant*, which is created by indexing a robot's joint set (and querying the *robotics subsystem*) in order to allow the end user to manually alter the joint/Cartesian pose for a given manipulator directly from the GUI.

Once a world is completely characterized, the user can perform complex kinematic and dynamic simulations on any active component of the world through the use of the robotic subsystem (see "Simulation and Control").

File Formats (Input/Output)

The ROBOMOSP platform manages its own file format, which is defined by a nonrecursive grammar. This grammar defines both the set of tokens and the syntax rules to be used by the ROBOMOSP lexical and syntactic analyzer, respectively.

One of the advantages of using this grammar is that all of the elements belonging to the ROBOMOSP system (a CSG solid, a robot, a trajectory, a task, or a world) can be defined in a plain ASCII text file, which can be shared among users or used via remote operations with little communication overhead. Elements defined in this language can be loaded into the graphics simulation process during run time, allowing dynamic world configurations to exist.

The same grammar can be used by external XML parsers to support I/O in other graphical file formats (DXF, IGES).

The following section describes the main grammatical constructs for data I/O (file I/O) for the world and robot modules in ROBOMOSP using Backus-Naur form (BNF).

World and Robot File Format Definitions

A world file describes the elements and their spatial relation in a 3-D simulation world. As mentioned previously, a world is made up of a list of objects (ObjectsList), a list of robots (RobotsList), a list of trajectories (TrajectoriesList), and a list of tasks (TasksList). The main purpose of these sets of rules is to describe a world reference frame (HGLMT) and to specify the text file that contains the definition of a list element (an object, a robot, or a trajectory).

On the other hand, a robot is defined as a set of 3-D objects (ObjectsLst) that make up its base and links and an *n*-ary tree (LinksNTree) that specifies the hierarchical relationship among the links. For any given link, its associated sons or brothers (LinkSons, LinkBrothers), its type (LinkType), and its corresponding 3-D object (LinkObject) are also specified. The following code fragment expresses, in BNF notation, the robot file format definition.

```
Robot            →   Description HGLMT '(' ')'
                     ObjectsLst
                     LinksNTree EOF
LinksNTree       →   '{' '[' '(' ListOfReals
                     ')'   LinkType   '('
                     LinkObject   ')'   ']'
                     LinkSons '}'
LinkSons         →   LinksNTree LinkBrothers
                 →   ϵ
LinkBrothers     →   LinkSons
                 →   ϵ
LinkType         →   'Revolute'
                 →   'Prismatic'
LinkObject       →   FileName(Object)
                     Description HGLMT
                 →   ϵ
ObjectsLst       →   FileName(Object)
                     Description HGLMTRstOb-
                     jectsLst
                 →   ϵ
RstObjectsLst    →   ',' ObjectsLst
                 →   ϵ
File Extension   →   .rbt
```

Interacting with the Platform Via the IOCI

The IOCI is an API that allows direct interaction with ROBOMOSP's internal elements, namely: robots, objects, trajectories, tasks, and worlds from an external application. ROMOBOSP tasks can be defined and used from a remote connection via sockets. For IOCI primitives running locally, a Tcl command is executed through an interface event or through the console window. For remote connections, a similar process takes place, except that the IOCI command arrives via a socket channel definition. A few of the embedded IOCI commands in ROBOMOSP are:

- **World** *Id Option args*
  Description: It performs the operation defined by the option value in a simulation world defined by *Id*.
  Options:
  – **World** *Id* **LsPObjects**: It lists all the parametric objects present in the simulation world identified by *Id*.
  – **World** *Id* **LsPRobots**: It lists all the parametric robots present in the simulation world identified by *Id*.
  – **World** *Id* **LoadPTrajectory** *FileName*: It loads the parametric trajectory defined in *FileName* into the simulation world identified by *Id*.
  – **World** *Id* **WorldInvKineSim** *PRobotId PTjcId*: It performs an inverse kinematic simulation between the parametric robot identified by *PRobotId* and the parametric trajectory identified by *PTjcId*.
  – **World** *Id* **WorldInvDynSim** *PRobotId PTjcId*: It performs an inverse dynamic simulation between the parametric robot identified by *PRobotId* and the parametric trajectory identified by *PTjcId*.
  – **World** *Id* **WorldfKineSim** *PRobotId PTjcIdj*: It performs a forward kinematics simulation between the parametric robot identified by *PRobotId* and the parametric joint trajectory identified by *PTjcIdj*.
  – **World** *Id* **WorldfDynSim** *PRobotId PTjcId*: It performs a forward dynamics simulation between the parametric robot identified by *PRobotId* and the forces identified by *PTforce*.
- **PRobot** *Id Option args*
  Description: It perfoms the operation defined by the Option value in a parametric robot defined by *Id*
  Options:
  – **PRobot** *Id* **AddLinks** *Values*: It allows to add the data given by *Values* to the links belonging to the parametric robot identified by *Id*.
  – **PRobot** *Id* **vGetDynamic**: It returns the dynamic parameters of all the 3-D objects associated to the parametric robot identified by *Id*.
  – **PRobot** *Id* **mGetDHp**: It returns the D-H matrix parameters that belongs to the parametric robot identified by *Id*.
  – **PRobot** *Id* **mGetToolHMT**: It returns homegenous transformation matrix parameters from the tool transform that belongs to the parametric robot identified by *Id*.
  – **PRobot** *Id* **SetLinks** *Values*: It allows to set the values given by *Values* to the links belonging to the parametric robot identified by *Id*.

OffLine and Online Programming

ROBOMOSP incorporates a high-level programming language compiler [25] that helps users model and execute complex tasks for the robots or other objects present in a simulation world. These actions are described using a high-level task-oriented language, the Industrial Robot Language (IRL, specified in the standard DIN 66312 [6]). The IRL compiler implemented is included as an embedded module in

ROBOMOSP, allowing users to execute programs written in this language and to simulate the associated response in objects bound to the user program within a ROBOMOSP world (Figure 4). It can also run in stand-alone mode with a socket-enabled IOCI connection for remote simulations.

IRL has characteristics that correspond to a conventional high-level structured language, and it also adds particular functions that simplify the specification of tasks using robotic manipulators. All four motion descriptors and data types of the original IRL specification are supported. The programmer can control the interaction of the robot with other devices in the world through the definition and management of signals (interprocess communication takes place over socket channels within ROBOMOSP). The compiler generates an intermediate level code (a three-address format code, appropriate for translation into robot controller languages) that is then postprocessed to produce specific target languages. ANSI C is supported as a final language within the environment (to control the flow in a user program for interacting with the IOCI interface and allowing, in turn, direct interaction with the robotic subsystem and with a simulation world). For each proprietary or open robot language, a particular translation file (postprocessing) is required. Currently, implemented postprocessors include definitions for Mitsubishi's PA-10 imperative programming libraries [18] in ANSI C and MELFA [26].

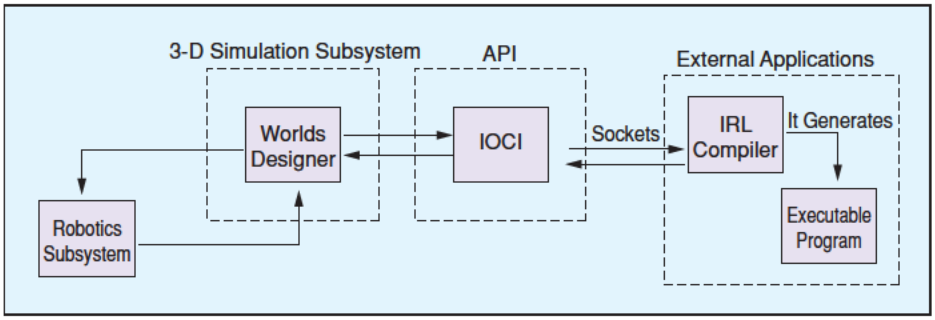

Figure 4. *Integration of the IRL compiler into ROBOMOSP.*

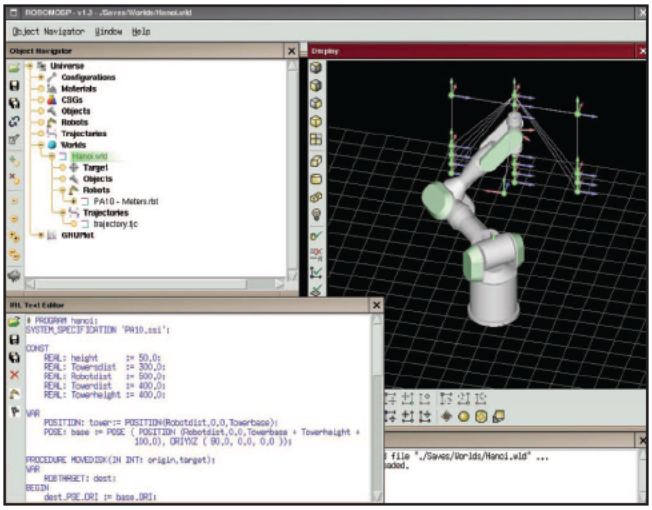

Figure 5. *IRL generated trajectory for the Towers of Hanoi problem running in ROBOMOSP.*

The embedded interface for IRL in ROBOMOSP is shown in Figure 4.

Figure 5 shows the trajectory produced by a program written in IRL to generate a solution to the problem of the Towers of Hanoi (corresponding IRL code fragment shown in the same figure).

Another alternative task programming scheme provided by ROBOMOSP is based on linear programming. Using a command interpreter, the control flow and motion parameters for a particular task are defined within a visual interface and with the association of predefined objects in a ROBOMOSP simulation world. The implemented visual (iconic) commands have a direct correspondence with those supported by the IOCI interface.

Simulation and Control

The robotic subsystem's main component is denoted as *LLRL* (*Low Level Robotics Library*); it contains numeric and algebraic implementations to solve the forward and inverse kinematics and dynamics problems as well as other fundamental mathematical constructs and methods required for simulation and control of robot manipulators. Solutions are provided in LLRL for open chains with single and multiple degrees of freedom (DOF) per joint, closed chains, and hyperbranched systems (no current support is provided for flexible joints). The latter two are currently being connected to the 3-D simulation engine.

Kinematics Simulation and Control

The forward kinematics solution is used to simulate spatial motions of the system modeled in time from its physical parameters and the joint state. The implementation is based on a sequential multiplication of the homogeneous transformations that describe the spatial relation between consecutive joints, $A_{1,0}..A_{n,n-1}$, to find the spatial location of the end effector with respect to the inertial reference frame, $T_{n,0}$.

$$A_{1,0}A_{2,1}\ldots A_{n,n-1} = T_{n,0} \qquad (1)$$

On the other hand, the solution to the inverse kinematics problem is conventionally used for control purposes (and, of course, to determine joint states in simulated motions). Using the system's physical parameters and the end-effector state (position and orientation), the corresponding joints state is determined. The generic numerical solution implemented in ROBOMOSP uses the manipulator's base referred Jacobian operator. As an alternative, the user is allowed to program his own solution—closed form (if one exists) or numerical—through the IOCI API to replace the numerical one provided.

ROBOMOSP also provides for alternate spatial transformations notations including Euler angles and quaternions.

This provides added physical insight into the nature of spatial transformations as well as didactic functionality.

The joint or Cartesian trajectories and other time-dependent variables can be visualized on the platform using embedded GNUplot graphics (see Figure 6).

Dynamics Simulation and Control

The solution to the equations of motion (EOM) for robot manipulators is used 1) to determine the joint forces (torques) that intervene in a particular spatial motion, referred to as the *inverse dynamics problem*, and 2) to find the spatial motion induced by a particular set of joint forces (torques), referred to as the *forward dynamics problem*. The default implemented solution in ROBOMOSP, for both forward and inverse solutions (serial chains, closed chains, and hyperbranched systems), is based on the spatial notation Newton-Euler solutions in [27]. The primitive solution for all mentioned systems is the serial chain, which is presented next.

Newton-Euler for Serial Chain Robotic Manipulators: Inverse Dynamics Solution

Assuming a base to end effector recursive propagation of kinematic parameters (refer to Figure 7), the spatial velocities (V) [rotational and translational components stacked in a single six-dimensional (6-D) quantity] for the $i = 1..n$ bodies of the system are expressed as

$$V_i = \hat{P}^T_{i-1,i} V_{i-1} + H_i \dot{Q}_i \tag{2}$$

$$\hat{P}_{i,i-1} = \begin{bmatrix} U & \tilde{p}_{i,i-1} \\ 0 & U \end{bmatrix}, \tag{3}$$

where U corresponds to the identity matrix $\epsilon^{3\times3}$, $\tilde{p}_{i,i-1}$ to the skew symmetric representation of a distance vector (equivalent to the vector cross product), H_i denotes the projection matrix for the DOF, and $\dot{Q}$ corresponds to the vector of joint velocities. Differentiating (2) with respect to time and setting as the initial condition of $\dot{V}$ the gravitational acceleration 6-D vector results in the expression for the spatial accelerations ($\dot{V}$):

$$\dot{V}_i = \hat{P}^T_{i-1,i} \dot{V}_{i-1} + H_i \ddot{Q}_i + \dot{\hat{P}}^T_{i,i-1} V_{i-1} + \dot{H}_i \dot{Q}_i. \tag{4}$$

A reverse recursive propagation (downward through the serial chain) from $i = n..1$ of the spatial forces (F_i) for each body of the system completes the EOM for the multibody:

$$F_i = \hat{P}_{i,i+1} F_{i+1} + I_i \dot{V}_i + \left[\dot{I}_i + I_i \dot{\tilde{S}}^T_{Oi,cm} \right] V_i, \tag{5}$$

where, the 6-D tensor of inertia for a body i, $I_i \in \Re^{6\times6}$ is formed from the scalar mass and the 3-D tensor of inertia with respect to the point of interest within the rigid body (for additional detail see [27] and [28]). Appropriate boundary conditions for the robotic manipulator being modeled are defined at the base (e.g., fixed, moving, or floating base) and at the end-effector body n (e.g., load conditions). To obtain the effective forces, the spatial forces are projected onto the DOF axes for the corresponding joints.

$$\mathcal{F}_i = H^T_i F_i. \tag{6}$$

Using the results in (2), (4), and (6), the user can simulate a torque control scheme for the robot being modeled. Results for gravitational torques (load), inertial torques, friction torques, and nonlinear torques (velocity dependent components, from [6]) are also obtained directly from ROBOMOSP solution to the above equations.

Solution to the Forward Dynamics Problem

On the other hand, to determine the motion induced by a particular set of forces (torques) on a particular multibody, ROBOMOSP uses the results described in the previous section to establish the acceleration-dependent force equation. F_i in (5) is expressed using high-level spatial notation (no indices) as

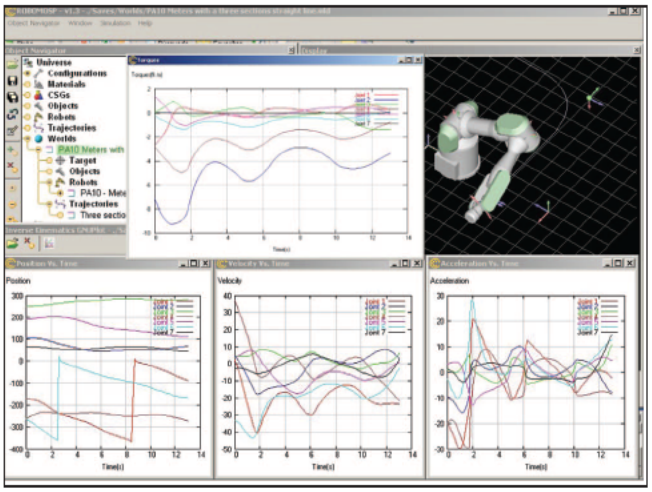

Figure 6. *Kinematic and dynamic joint profiles for a particular Cartesian motion with a 7-DOF Mitsubishi PA-10 robotic manipulator.*

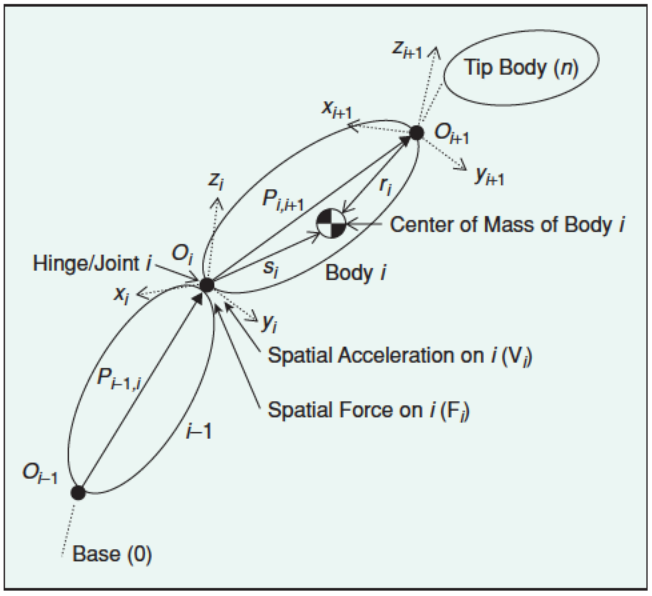

Figure 7. *Serially articulated system of rigid bodies.*

$$\mathcal{F} = H^T P^{-1} I (P^{-1})^T H \ddot{Q} + H^T P^{-1} [I((P^{-1})^T \dot{H} + (\dot{P^{-1}})^T H) + (\dot{I} - \dot{S}I)(P^{-1})^T H] \dot{Q}, \quad (7)$$

and in simplified state space form as,

$$\mathcal{F} = \mathcal{M}(Q)\ddot{Q} + \mathcal{C}(Q, \dot{Q}), \quad (8)$$

where the nonlineal velocity dependent terms in the EOM (i.e., coriolis and centrifugal accelerations and gyroscopic forces) are condensed in $\mathcal{C}$, $\mathcal{F}$ corresponds to the effective forces, and $\mathcal{M}$ corresponds to the joint-space inertia (JSI) matrix of the multibody. The JSI matrix of the multibody is computed from its mass distribution and kinematics parameters (see [29] for additional details). The JSI matrix relates the forces with the accelerations that take place under a particular motion of the system. The net acceleration-dependent effective forces for the system are then given by

$$\tau(Q, \dot{Q}) = \mathcal{F} - \mathcal{C}(Q, \dot{Q}) = \mathcal{M}(Q)\ddot{Q}. \quad (9)$$

From (8) it becomes clear that the nonlineal velocity-dependent term $\mathcal{C}$ is found from the application of the EOM developed in the previous section with $\ddot{Q} = 0$. It then follows that the forward dynamics problem solution is expressed by

$$\ddot{Q} = \mathcal{M}(Q)^{-1}\tau(Q, \dot{Q}). \quad (10)$$

A straightforward solution for the above equation involves using (9) to find the JSI matrix. For each each column k_i $i = 1 \ldots n$ of the JSI matrix $\mathcal{M}$ the vector of joint accelerations, $\ddot{Q}_i$, corresponds to $\ddot{Q}_i = \{0 : \forall_{i \neq k}; 1 : \forall_{i=k}\}$, where the column k_i of $\mathcal{M}$ is calculated from the $O(n)$ Newton-Euler formulation, described previously, to find the forces (torques) for a given joint trajectory. This solution has a serial computational complexity of $O(n^2)$. An alternate solution to the forward dynamics problem of $O(n)$ serial computational complexity has also been implemented in ROBOMOSP from [27]. In spite of the lower order of complexity, the latter is useful in systems with a large number of DOF (>6) due to its higher linear coefficient for its corresponding serial computational complexity. Nonetheless, this implementation will become increasingly important in future versions of ROBOMOSP with multithreading and real concurrency support, given its time lower bound computational complexity $O(log_2 n)$ with $O(n)$ processors under strict parallelism.

A variable rate Runge-Kutta integration scheme is used to determine the joint position state at each time step of the dynamics.

Remote/Distributed Control Via Sockets

The current trend in software development is to produce applications that can be connected to other applications using the TCP/IP protocol. This is due primarily to the massive use of the Internet. With Tcl, a TCP/IP connection can easily be created with the socket command.

The ROBOMOSP plataform makes use of this socket command using a window called *sockets windows*. Through this window, ROBOMOSP establishes a communication channel with an external application that could be running either on the local machine or remotely (e.g.,

Table 1. Correlation coefficients of physical and simulated results (PA-10 circular interpolated trajectory).

r_x	r_y	r_z	r_{θ_1}	r_{θ_2}	r_{θ_3}	r_{θ_4}	r_{θ_5}	r_{θ_6}	r_{θ_7}
0.9987	0.9986	1	0.9970	0.9986	0.8405	0.9987	0.9328	0.9985	0.9982

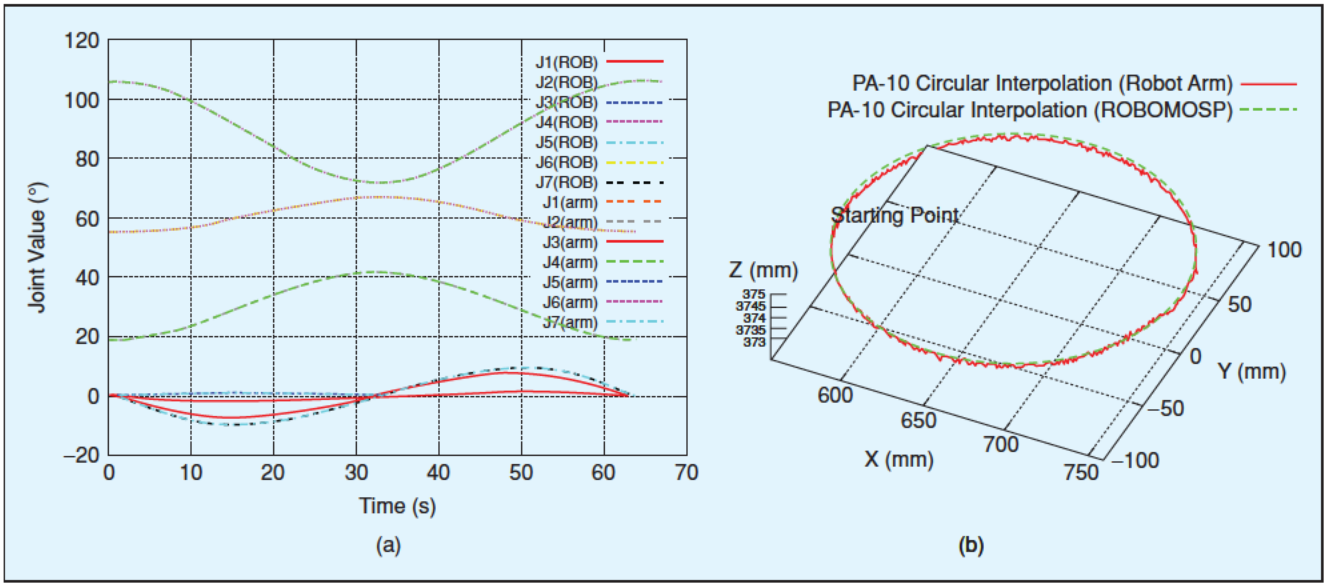

Figure 8. *Cartesian/joint circular interpolated trajectory on a PA-10 robot manipulator (actual and simulated).*

from a physical robotic manipulator controller). ROBOMOSP supports Tcl scripts that contain both Tcl and IOCI commands, thereby allowing simulation processes to be executed and controlled remotely.

Validation of ROBOMOSP's Modeling and Simulation Capabilities

Perhaps the most important aspect of any modeling and simulation software tool is how well its results correlate to the actual physical behavior of the corresponding simulated systems. Four test cases are included here, to demonstrate and validate some of ROBOMOSP's principal capabilities:

1) *Kinematic execution of a circular interpolated Cartesian trajectory using a PA-10 robot manipulator.* An actual PA-10 arm was used to perform a circular interpolated motion of radius 100 mm with origin at $x = 658$ mm, $y = 0$ mm, and $z = 734.6$ mm using three coplanar control points at $x = 558; 758; 658$ mm, $y = 0; 0; -100$ mm, and z constant at 734.6 mm, respectively. The circular trajectory was executed in counterclockwise motion with respect to the axis perpendicular to the plane of the circle at a constant tangential velocity of 10 mm/s . The same trajectory, with the same parameters, was programmed and executed in ROBOMOSP for the modeled PA-10. The results, for both the Cartesian and joint trajectories, are presented in Table 1. From Figure 8 the deviation of the actual robot from the desired trajectory in depicted. The deviation in the vertical axis (note the z-axis range) is due to the effect of gravity and the fluctuating underdamped/overdamped responses of the control scheme used for kinematic positioning of the PA-10 arm. The maximum deviation from the trajectory set point in the vectical axis is 0.8 mm, with a standard deviation of 0.1465 (within the PA-10 specifications).
2) *Calculation of the mass parameters for an asymmetric, heterogeneous density, compound object*: A compound object made of a steel rectangular solid (dimensions 20 cm × 20 cm × 5 cm, $\rho = 7,850$ kg/m^3) and a cylindrical aluminum rod (length = 20 cm, radius = 10 cm, $\rho = 2,700$ kg/m^3) placed on top of it displaced 5 cm onto the side of the axial center (along the y-axis of the reference frame) was assembled, and the corresponding mass properties were calculated using Pro/Engineer [32] and ROBOMOSP. The results are tabulated in Table 2.
3) *Inverse dynamics for a Puma 560 robot manipulator using ROBOMOSP and the Robotic Toolbox in MATLAB*: This test was performed using the Robotics Toolbox for MATLAB [33] and the mass parameters for the Puma 560 robot as given by [34] (due to unavailable structural design parameters for this particular arm). The torques were computed for a {0, 0, 0, 0, 0, 0} pose using DH parameters in [30] (identical coordinate frames assignment), with null joint velocities and accelerations, no external forces acting on the end-effector, and a gravity force along the waist axle of the manipulator. For both tools, ROBOMOSP and MATLAB's toolbox, the resulting torques were {0, 37.48366, 0.248928, 0, 0, 0} N·m and {0, 37.4837, 0.2489, 0, 0, 0} N·m, respectively.
4) *Remote execution of a user defined task*: The test performed consisted of the remote execution of a user task on a predefined world (via Tcl script written with the defined BNF grammar) using available parametric objects to solve and query for the joint state of a PA-10 robot arm for a Cartesian target (see Figure 9) given at end-effector position [0.557, 0, 0.374] m and (roll, pitch, yaw) orientation $[\pi, 0, \pi]$. The relevance of this result, in this particular context, lies in the ability to provide a virtual experimental platform that conveys results otherwise physically unavailable to the end user. For example, a student can remotely test solutions through the Internet in ROBOMOSP without requiring an actual (costly) robotic system being present or without risking damages to a remote physical facility. Other applications for this utility include remote testing of user-defined parametric objects, tasks, and methods, including virtual robot cooperative tasks synchronized via sockets.

Final Observations

This article has described the main modules that make up ROBOMOSP, a robotics modeling and simulation platform

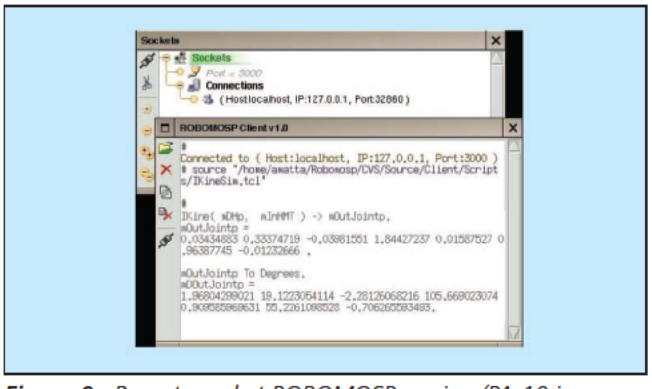

Figure 9. *Remote socket ROBOMOSP session (PA-10 inverse kinematics solution at pose* [0.557*m*, 0*m*, 0.374*m*, π, 0, π]*).*

Table 2. Mass properties of a compound object (Pro-E versus ROBOMOSP).

	Mass (kg)	x_c(m)	x_y(m)	x_z(m)	I_{xx}	I_{yy}	I_{zz}	I_{xy}	I_{xz}	I_{yz}
Pro/Engineer	19.9411	0	0.0106	0.0515	0.1329	0.1245	0.1183	0	−0.0208	0
ROBOMOSP	19.6439	0	0.0102	0.0505	0.1257	0.1177	0.1169	0	−0.02	0
Relative difference	1.49%	0	3.77%	1.94%	5.42%	5.46%	1.18%	0	3.85%	0

developed by the Robotics and Automation Group (GAR) of the Pontificia Javeriana University, in Cali, Colombia. This software platform adds novel characteristics/functions that are not found in other commercial and noncommercial robot modeling and simulation packages available today (see partial comparison in Table 3), including: solution to the multibody dynamics problem using automatic calculation of the mass properties of robot multibodies, offline programming using a standard language (IRL), an API interface to allow experimentation with new algorithms written by the end users, and support for remote/distributed use of the platform via socket communications. ROBOMOSP is ideal for training robotic operators, as a research aid, and for studying the mathematical and physical foundations of robotics manipulators, thanks to its ability to permit the expression of models that closely simulate the behavior of real systems within a feature-packed user-friendly interface. Last but not least, the fact that ROBOMOSP is cross-compatible with Linux, MacOSX, and MS Windows, makes it the ideal environment for cooperation among robot researchers around the world.

Future Work

Current work around ROBOMOSP is oriented towards adding new features, such as: a graphical teaching module to be used in a simulation world (joystick training); incorporation of a collision detection subsystem based on the objects' mesh representation; import/export data files filters; adding graphical libraries of commercial robots and robot components; stereoscopic animation; and developing additional postprocessors (for the embedded IRL programming compiler) for commercial controllers such as KRL [31], MRL [35], SRL [36], and RAPID [37]).

Acknowledgments

The authors would like to thank O. Ramos, G. Meneses, M.C. Pabón, N. Rojas-Libreros, and others that have contributed code in the development of ROBOMOSP.

Keywords

Robotics manipulator modeling and simulation, manipulator kinematics, manipulator dynamics, robot trajectory planning, offline programming of robotic manipulators.

References

[1] RoboWorks Software, Newtonium Web site [Online]. Available: http://www.newtonium.com/

[2] RoboAssist, "Robot modeling and control package," New River Kinematics Inc. [Online]. Available: http://www.kinematics.com/products/educational/robotassist/index.html

[3] Easy-ROB3D Software, Easy-ROB Web site [Online]. Available: http://www.easy-rob.com/

[4] Workspace Software, Flow Software Technologies Web site [Online]. Available: http://www.workspace5.com.

[5] Open Source Web site[Online]. Available: http:// www.-opensource.org/

[6] Deutches Institut for Normung, "IRL, Industrial Robot Language," Beuth-Verlag, Germany, DIN 66312, 1992.

[7] A. Matta and W. Perea, "RobLab: Herramienta gráfica para el modelamiento de robots," M.A. Thesis, Carrera de Ingeniería de Sistemas y Computación, Pontificia Univ. Javeriana, Cali, Colombia, 2003.

[8] R.S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*. New York: McGraw-Hill, 2002.

Table 3. Features comparison for several commercial and noncommercial robot graphic simulation platforms.

Feature	RONOMOSP	Ropsim	Simderella	Workspace	Easy-rob	RobotAssist
Multiplatform support	Yes	No	No	No	No	No
3-D solids rendering	Yes	Yes	No	Yes	Yes	Yes
CAD import/export utilities	No	Yes	No	Yes	Yes	Yes
Forward kinematics	Yes	Yes	Yes	Yes	Yes	Yes
Inverse kinematics	Yes	No	No	Yes	Yes	No
Automatic mass properties calculation	Yes	No	No	No	No	No
Inverse dynamics	Yes	No	No	No	Yes	No
Forward dynamics	Yes	No	No	No	No	No
Analytical trajectories	Yes	No	No	Yes	No	No
Interpolated (splines) trajectories	Yes	Yes	No	Yes	Yes	Yes
PTP trajectories	Yes	Yes	Yes	Yes	Yes	Yes
Collision detection	No	Yes	No	Yes	No	No
Complex worlds modeling	Yes	Yes	No	Yes	Yes	Yes
Tasks modeling	Yes	No	No	Yes	No	No
User defined variable watch	Yes	No	No	No	Yes	Yes
Graphic visualization of results	Yes	Yes	Yes	Yes	Yes	Yes
Offline programming	Yes	Yes	No	Yes	Yes	Yes
Online programming	Yes	Yes	Yes	Yes	Yes	Yes
Remote operation via sockets	Yes	No	Yes	No	Yes	Yes
API support (user defined methods)	Yes	No	No	Yes	No	No

Only platforms that allow graphical modeling are considered; some features could not be directly verified from available demos.

(9) B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin spiral model: A case study ' *IEEE Comput. Mag.*, pp. 3.July 1998.

(10) J.G. Schneider and O. Nierstrasz, "Components, scripts, and glue:• in *Software Arc/1iteá11res-Adva11ces a11d Applications*, L. Barroca, J Hall, and P Hall, Eds. New York: Springer-Verlag, 1999.

(11) GNU Public License, Descripción de Ja Licencia (Online). Available: http://wwwgnu.org/copyleft/gpl.html.

(12) J K Ousterhout, *Tcl atUI the* Th *Too/kit*. Reading. PA: Addison-Wesley, 1993.

(13) J. Neider, T. Davis, and M. Woo, *OpenG/ Programmi11g Guide-111e Offrcia/ Cuide to Leami11g OpenGI*. Reading, PA: Addison-Wesley, 1993.

(14) D.E. Stewart and Z. Leyk, "Meschach library version *t.2:*• School of Matbematical Sciences, Australian National Univ, Australia, 1994.

(15) J. Sbolber, "NETGEN-An advancing front 20/30-mesh generator based on abstract rules:• *Comp11t. Visual. Sd.*, 1997.

(16) J. Goldfeather and H. Fuchs, "Near real-time CSG rendering using tree normalization and geometric pruning:• *IEEE Comput. Grapl1ics Applicat.*, vol. 9, no. 3, pp. 20-28, May 1989.

[17) H Mayr, *VirtualA11tornation & IVÍromnents*. New York: Man:el Dekker, 2002

(18) *PA- 10 Portable General Purpose f1lte/Jigent Arm*, Mitsubishi Heavy Industries, ltd.5-1, Chiyoda-ku, Tokyo, Japan.

(19) B. Mirtich, "Fast and accurate computation of polybedral mass properties:' *J Grapl1ics Too/s*, vol. 1, no. 2, 1996.

(20) J. Denavit and R.S. Hartenberg, "A kinematic notion for Jower-pair mechanisms based on matrices," *Trans. ASME, J Appl Mech.*, vol. 22, no. 2, pp. 215-221, June 1955.

(21) J Craig, *Introd11aio11 to Robotics: Mecl1a11ics atUI Co11trol*, 2nd ed Reading, PA: Addison-Wesley, 1989.

(22) A. Jaramillo-Botero, J.F. Correa-Caicedo, and lJ. Osorio-Parra, "Trajectory planning in ROBOMOSP '(in Spanish), Robotics and Automation Group {GAR), Univ. Javeriana, Cali, Colombia, Tech. Rep. GAR-TR-10-2004, 2004.

(23) B. Richard, L. Faires, and J. Douglas, "Análisis Numérico:• Internacional Thomson Editores, México, 1998.

(24) T. Williams, C. Kelley, J. Campbell, D. Kotz, and R. Lang. "GnuPlot: An interactive plotting program:• Dartrnoutb Univ, Cambridge. MA, 1998.

(25) O. Ramos, G. Meneses, M.C. Pabón, and A. Jaramillo-Botero, "IRL {lnternational Robot Language) for ROBOMOSP," (in Spanish), Robotics and Automation Group (GAR), Univ. Javeriana, Cali, Colombia, Tech. Rep. GAR-TR-12-2004, 2004.

(26) *MELFA, Mitsubis/1i Robots Programmi11g La11g11age*, Mitsubishi RVM-1 Robot, Mitsubishi.

(27) A. Jaramillo-Botero and A. Crespo, "A unified formulation for massively parallel rigid multibody dynamics of *O(L og2N)* computational complexiry •*J Para/Jet Distrib. Comput.*, vol. 62, no. 6, June 2002.

(28) R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *Int. J Robotics Researc/1*, vol. 2, no. 1, pp. 1 30, 1983.

(29) A. Jaramillo Botero, "Design criteria for simplified dynamics complexity serial robotic manipulators," *Epiciclos*, vol. 1, no. 2, 2002.

(30) K.S. Fu, R.C. González, and C.S.G. Lee, *Robotics Control, Se11si11g, Vision, and lrltelligence*. New York: McGraw-Hill, 1987.

(31) *K11ka Robotics Lang11age*, Kuka Roboter GmbH Web site (Online). Available: htttp://www.kuka.com/en

(32) Parametric Technologies Corporation Web site (Online). Available: http://\vww.ptc.com/appserver/mkt/products/home. jsp?k=403

(33) P. Corke, "A robotics toolbox for MATLAB:' *IEEE Robot. A11tamat. Mag.*, vol. 3, no. 1, pp. 232, Sept. 1996.

(34) B. Armstrong, O. Kbatib, and J. Burdick, "The explicit dynamic modelo and inertial paramaeters of the Puma 560 arm:• in *Proc. IEEE Int. Conf Robotics a11d Automation*, 1986, vol. 1, pp. 510-518.

(35) H. Nishiyama, H. Obwada, and F. Mizoguchi, "A multiagent robot language for communication and concurrency control ' in *lrlt. Conf Mu/tiagent Systems (ICMAS)*, 1998, pp. 206-213.

(36) C. Blume and W. Jakob, "Design of the structured robot language {SRL):' in *Adva11ced Software i11 Rabotics*, A. Danthiene and M. Geradin. Amsterdam, Elsevier, 1984. pp. 127-143.

(37) RAPID, ABB Robotics Language (Online). Available: http://\vww.abb. com/robotics