Robotica (2004) volume 22, pp. 141–154. © 2004 Cambridge University Press
DOI: 10.1017/S0263574703005434 Printed in the United Kingdom

A modular object-oriented framework for hierarchical multi-resolution robot simulation

Sanghoon Yeo,† Jinwook Kim,‡ Sung Hee Lee,¶ F. C. Park,¶ Wooram Park,¶ Junggon Kim,§ Changbeom Park§ and Intaeck Yeo§

(Received in Final Form: June 9, 2003)

SUMMARY

We describe the design and implementation of RSTATION, an object-oriented, modular robot simulator with hierarchical analysis capabilities. Modularity is achieved via the features of design encapsulation and enables grouping a set of interconnected components into a single component and dividing the robot system into several sets of subordinate modules recursively. By careful construction of the data types and classes, RSTATION allows for hierarchical simulation of the kinematics, and the dynamics at three levels: considering only main links (high-level), using simplified models including dynamic properties of transmission elements (intermediate level), and taking into account the detailed kinematics and dynamics of transmission elements (low-level). Submodules can be set to different resolution during a single simulation. The data types and classes also exploit a recent set of coordinate invariant robot analysis algorithms based on modern screw theory. Central to the low-level dynamic analysis capability is an algorithm for systematically extracting the constraint equations for general gearing systems. The various features of RSTATION are illustrated with a detailed case study of a commercial industrial robot.

KEYWORDS: Robot simulation; Object orientation; Hierarchical simulation.

1. INTRODUCTION

Recently, robot offline programming (OLP) systems have received increasing attention as a means of reducing the cost and effort associated with designing new robot systems. Ideally OLP systems provide an interactive environment for a designer to design, evaluate, and optimize robot systems. They can also serve as a platform for robot motion and task programming, in which a motion program verified by the simulator can be directly downloaded to the robot for immediate execution.

Designing a robot on an OLP system usually begins with a kinematic realization of the conceptual design, in which a mechanism consisting of joints and links that satisfies the desired workspace criteria is constructed. One then endows the links and joints with mass and inertial properties to gauge the dynamic performance of the robot, and to determine, e.g. payload and actuator characteristics. At the final stage, a detailed mechanical realization of the mechanism at the machine element level is constructed; this involves, among other things, modeling the physical properties of transmission elements such as gear trains, springs, and damper units.

The many commercial OLP systems currently available today are more concerned with robot task programming applications rather than design. While these systems provide basic kinematic and dynamic analysis capabilities, none are able to support the detailed physical modeling of transmission elements that constitute a robot system. From the design perspective this is a serious deficiency of current OLP systems; performing a dynamic analysis of a robot taking into consideration only the mass and inertial properties of the links can produce results significantly different from one that accurately models the transmission element dynamics.

On the other hand, general multibody system analysis packages, while supporting detailed dynamic analysis at the machine element level, are not specialized for robot design. As a result these packages lack features such as modular design and hierarchical analysis specific to robot systems. The analysis algorithms also do not exploit the many special structural properties of robots, relying instead on general differential-algebraic constraint solvers that sacrifice a certain degree of computational performance for generality.

This paper describes the design and implementation of RSTATION, an object-oriented, modular, hierarchical robotic simulator for robot design. For our purposes we organize the basic functions of a robotic simulator into that of modeling and analysis. The modeler creates models of the various components constituting the robot, and organizes the topological and functional relationships between the components. Analysis involves tasks such as the kinematic, dynamic, and structural performance evaluation of the robot at varying levels of resolution, as well as trajectory generation and controller design. What sets apart our system from previous approaches is the modularity and hierarchical analysis capability, from the simple kinematic to the detailed machine element level, as well as the exploitation of the many special properties inherent to robots, all in an object-oriented framework. Some industrial robot manufacturers such as Yaskawa have developed analysis software for robot dynamics that is similar to ours. However, the

† Nexon Co., Media Laboratory (South Korea).
‡ KIST, Imaging Media Research Center (South Korea).
¶ Seoul National University, School of Mechanical and Aerospace Engineering (South Korea).
§ Hyundai Heavy Industries Co., Intelligent Mechanical System Research Dep. (South Korea).
E-mail: fcp@snu.ac.kr

software is not open to the public and the concept of multi resolution analysis is a unique feature of our simulator. We now describe in more detail the special features of our robotic simulator.

- The design methodology proposed in this paper allows one to construct whole robot systems by grouping submodules recursively. *Design encapsulation* (i.e. grouping a set of interconnected components into a single component and including/subordinating designed groups recursively) allows the designer to create complex submodules rapidly and efficiently.
- The developed simulator also supports hierarchical analysis capabilities. Unlike the simulation of digital circuits, in which the digital logic, circuit element, and physical device layers can be simulated more or less independently and in an hierarchical fashion, it is difficult to establish such a hierarchy for robot systems with independent layers; while kinematic issues can be separated from dynamic ones, as mentioned earlier, dynamic analysis can be approached from several levels. By a careful construction of the data types and classes, our simulator allows for independent simulation of the kinematics, and the dynamics at three levels: considering only the basic topologies of the main links (high level), adopting a simplified dynamic model including mass and inertias of machine/transmission elements (intermediate level), and taking into full consideration the detailed kinematics and dynamics of all elements comprising the system (low level). Submodules can also be selectively set at different levels to increase the computational efficiency of the overall system simulation.
- The data types and classes are also designed to take advantage of a recent class of coordinate invariant robot analysis algorithms based on modern screw theory.[1–3] Current analysis packages are intended for general multibody system analysis, and apply differential-algebraic equation solving algorithms. These algorithms do not take advantage of the special structure inherent in robot systems, and the specialized algorithms that have recently been developed for their analysis. In this regard one of the contributions of this paper is an algorithm for extracting the constraint equations for complex geared mechanisms, a feature not found in existing robot OLP systems We also make extensive use of a newly defined port data class to augment the traditional graph representation for mechanisms.

The robotic simulator is also designed using the object-oriented programming methodology. Given that the philosophy of object orientation evolves around the idea of modeling and simulating real-world objects by building simple mappings between real-world objects and their software representation,[4] robot OLP systems are one of the most natural candidates for implementation in the object-oriented framework. Zao[5] proposes a set of object-oriented geometric data types for computations involving rigid-body motions, while Kang[6] develops a set of object-oriented data structures and algorithms for the kinematic and dynamic analysis of serial chain mechanisms. Kecskemethy[7,8] also proposes an object-oriented programming framework, intended primarily for general multibody systems, using the client/server model. Closely related to our work is the modular object-oriented software environment for robot simulation developed by Ferretti et al.[9] This system supports modular design using an object-oriented database of physical components, but relies on general differential-algebraic analysis algorithms that do not exploit the advantages of recently developed geometric algorithms. Stewart et al.[10] examine port-based objects in a more general real-time programming setting.

Our object-oriented robot simulator is designed to be consistent with the general procedure for the computer simulation of general engineering systems (see Figure 1). The modeler constructs a computer model of the physical robot system, while the formatter performs the dual functions of extracting the physical parameters required for analysis, and transforming them into a form suitable for the computational analysis engine, or solver. The solver library is designed in a way that is independent of the chosen

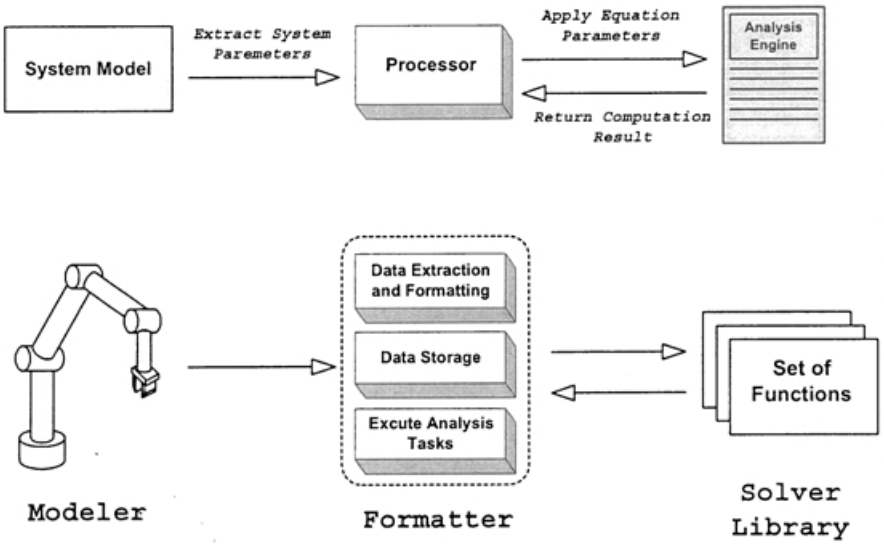

Fig. 1. General framework for a simulation system.

modeler data structures, and in this paper we focus on the development of the modeler and formatter.

2. MODELER DESIGN

In this section we present the functional requirements of the modeler and its design implementation. The modeler's primary function is to construct a computer model of the physical robot system, by classifying the components that constitute the robot and defining relationships between the component classes. The basic functions of the modeler are association and aggregation. Association allows one to connect components together, while aggregation allows one to recognize these connected components as single entities, as either a group of components, or recursively as a group of subgroups. Ultimately the function of association and aggregation allows the user to design components and subassemblies of a robot system separately, and enables the features of design encapsulation.

2.1 Design Encapsulation

The concept of modular design can be understood as encapsulation of interconnected component sets. The notion of encapsulation is one of the fundamental concepts behind object-oriented programming. Abstractly, encapsulation allows one to regard a group of components or subgroups as a single module, with the information about the inner components hidden during the design process. In our modeler, the encapsulation function allows one to define a group of mechanical components or recursively a group of subgroups as a single module. For example, the gearbox shown in Figure 2 consists of many gears and shafts encased in a housing; however, the user may only be interested in the input and output shafts, which are connected to other components. The gearbox can therefore be regarded as a single component regardless of the complexity of the inner structure. The encapsulation feature protects the user from directly accessing and modifying the inner components.

Design encapsulation is also related to the extensibility of the data structures for our model components, i.e. the ability to create new data types by combining pre-existing data types. For example, the spherical joint unit can be represented as a serial connection of three revolute joints rather than creating a completely new data type; by appropriately grouping three revolute joints, a spherical joint can be created.

2.2 Object Design

In our simulator we use graphs to represent the topological relationships between the various components of the robot system. Recall that a graph is composed of mutual connections of edges and vertices. When modeling a robot system as a graph, links are modeled as vertexes and joints are modeled as edges.

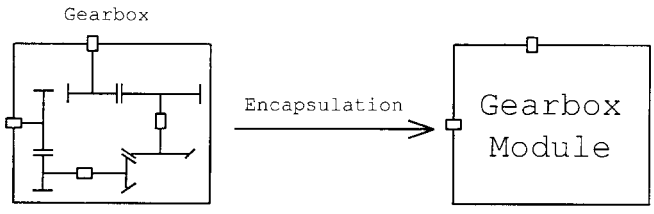

Fig. 2. Example of design encapsulation: gearbox.

To implement the concept of design encapsulation, links and joints can be defined recursively (i.e. a link or a joint can be defined by assembling subordinate links and joints) and therefore a vertex or an edge of a graph structure can represent a group of mechanical elements which has a certain level of encapsulation. For example, a spherical joint is represented as a serial connection of three revolute joints, which can be regarded completely same as a single object.

However, edges and vertexes alone are insufficient to provide complete topological information on the robot system configuration, since there are no elements to indicate the point of attachment between a link and joint. For this purpose we introduce a new type of object, called the port, that represents the position and orientation of the point of attachment between the link and joint. Port denotes a frame attached to its parent link. It consists of an $SE(3)$ element with respect to the reference frame of the parent link, and a $dse(3)$ (i.e. an element of the dual Lie algebra $se^*(3)$; see, e.g. reference [11]) element which indicates the generalized force applied to that frame and pointing to the attached element connector. Port objects also act as a gateway to element type groups in the design encapsulation process.

The classes that constitute the modeler can be classified according to their function as fundamental classes and physical classes. Fundamental classes are abstract classes for implementing aggregation and association relationships of graph structures, while physical classes model actual objects with physical parameters. There is a one-to-one or one-to-many inheritance relationship between fundamental classes and physical classes; a physical class includes the same attributes and operations as its fundamental class. A physical class can therefore be regarded as equivalent to its fundamental class in the overall association and aggregation schema.

Figure 3 represents the object diagram of the designed modeler using the OMT notation as described in Rumbaugh et al.[12] All the classes are aligned systematically by inheritance and association relationships.

2.3 Hierarchical Analysis

The ability to analyze systems in a hierarchical fashion is an indispensable feature of any computer-aided design and analysis system. The simulation of digital logic systems, for example, can be analyzed from the logic level to the transistor level, and even further down to the physical device level when necessary. Similarly, robot analysis usually begins with a conceptual kinematic analysis based on workspace requirements, followed by an initial dynamic analysis for determining the link dimensions and actuator sizes, and eventually to a detailed analysis at the machine element level. In general, layers in a hierarchical model should be independent, meaning the model parameters specified at each level should be sufficient for simulation and analysis of the system at that level.

Unfortunately for robots and other mechanical systems, achieving a hierarchical analysis with independent levels is

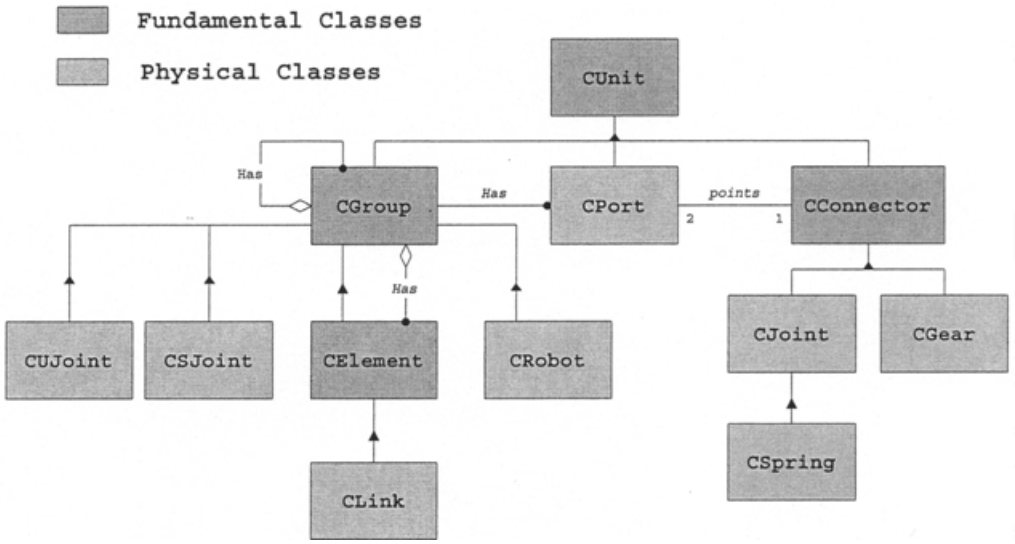

Fig. 3. Object diagram of modeler.

a much more difficult task. Although it is in general possible to separate kinematic analysis issues from dynamic ones for most multibody systems, further subdividing the dynamic analysis into independent levels is difficult due to the much greater internal complexity of the mechanisms compared to digital logic systems.

For example, the three-axis wrist of an industrial robot shown in Figure 4 can be kinematically represented as a serial chain connected by three joints, with the joint axes intersecting at a common point. The actual mechanism, however, consists of four links including base and three joints, together with several auxiliary links and joints

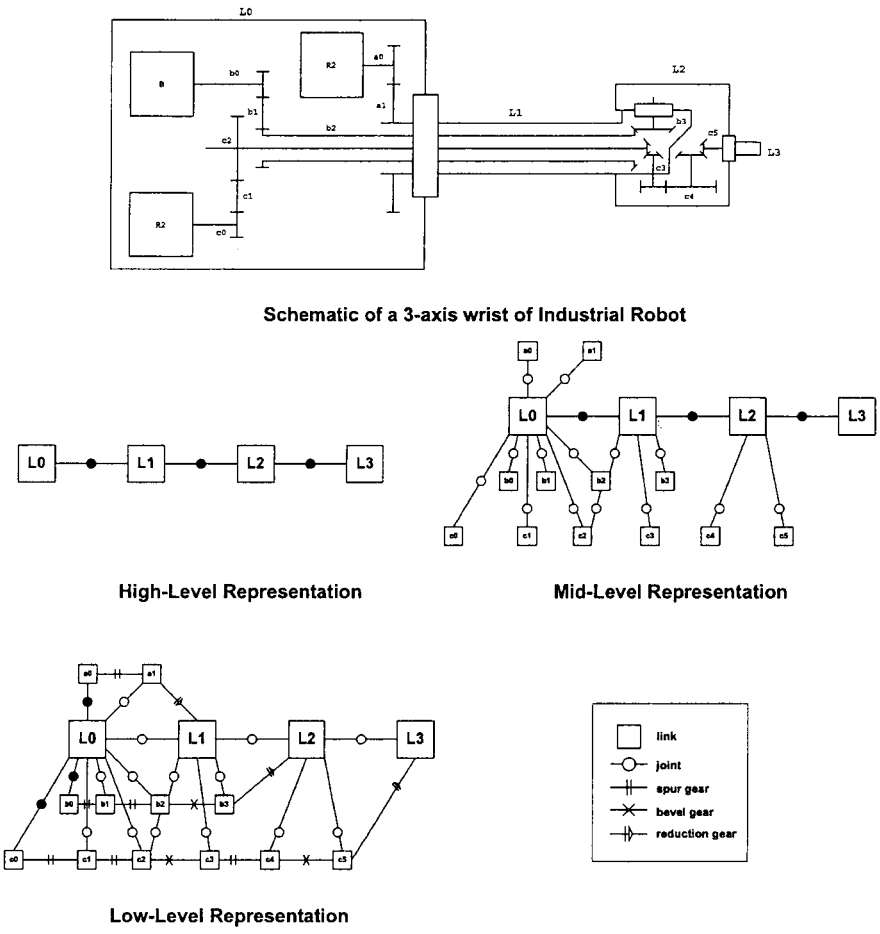

Fig. 4. Multi-level representation of a three-axis robotic wrist mechanism.

arranged in a closed loop structure, with each axis actuated by gear trains.

A simplified dynamic analysis that only considers the masses and inertias of the kinematic links assuming torques are applied directly to the joints, and the lumped approach that adds the masses and inertias of the auxiliary links to the primary kinematic links in some appropriate fashion, may produce results quite different from that produced by a more detailed dynamic analysis that takes into full account the detailed transmission elements such as gear trains that transmit motor torques, and the springs and dampers that generate structural forces.

Notwithstanding these natural limitations, we would like, to the extent possible, to endow hierarchical analysis features in our robot system modeler, in particular because the computation costs can be significantly reduced when the user requires a coarse result or approximation. For example, if the goal is to simulate arm-lifting motions of a humanoid robot, it makes sense to use a simplified leg model since these will not affect the result in a major way. By appropriately designing the data types and classes, we show that it is possible to achieve independent simulation of the kinematics, as well as the dynamics at three levels: considering only the link masses and inertias (high-level), adopting a lumped model approach that augments the link mass and inertial properties with those of the connecting transmission elements (intermediate level), and taking into account the detailed dynamics of all machine and transmission elements, e.g. springs, gears, dampers (low-level).

Figure 4 illustrates this hierarchical analysis feature for the three-axis wrist. In the schematic, the black circles represent actuators, L_1, L_2 and L_3 are the main kinematic links of the mechanism, and a_i, b_i, c_i, $i=1, 2, 3$, represent the individual gear elements in the respective gear train. The mid-level representation depicts the mechanism with the transmission element connections removed, while the low-level connection depicts the complete connection of gears, springs, and dampers. Our case studies will further illustrate the implementation of this hierarchical design feature, including the ability to set different modules of the system to different dynamics levels during a single simulation.

3. FORMATTER DESIGN

The basic function of the formatter is to extract the necessary parameters from the modeler, and to invoke the appropriate functions from the solver library for analysis and simulation of the system.

The data in the formatter can be divided into configuration data local data. The configuration data contains information about the number of units in the model (the number of joints, the number of gears, the number of closed loops, etc.) and the set of pointer arrays covering all the units in the model. The local data contains information about specific parts of the model as required during the analysis stage. The configuration data can be extracted from the modeled robot directly, while the local data can be obtained after data processing.

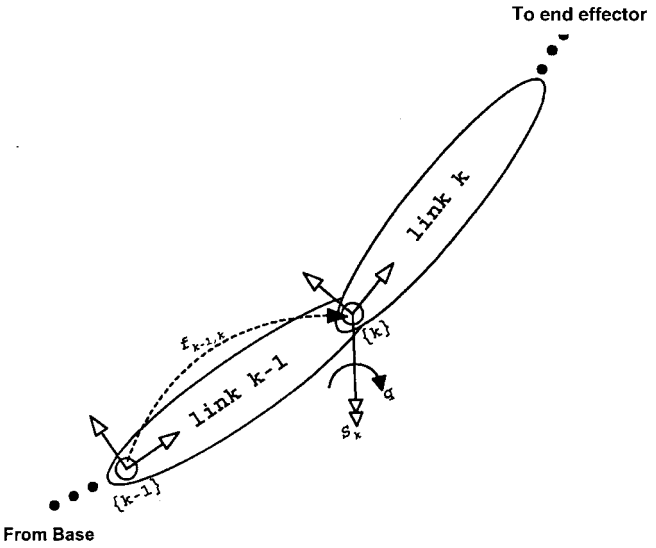

Fig. 5. The forward kinematics of a serial chain mechanism.

3.1 Data Formats for Kinematic Analysis

The forward kinematics of an open chain mechanism can be described by a product of homogeneous transformations. Let the homogeneous transformation of the {k} frame as seen from the {k − 1} frame be $f_{k-1,k}=e^{S_k q_k} M_k$ as shown in Figure 5. Here S_k is the screw parameter for joint {k} expressed in frame {k}, q_k is the joint angle, and M_k is the homogeneous transformation of frame {k} seen from frame {k − 1} when $q_k=0$. The end-effector frame T of a serial chain comprised of n links can be written as follows:

$$T=f_{0,1}f_{1,2}\cdot\cdot\cdot\cdot f_{n-1,n}=M_1e^{S_1q_1}M_2e^{S_2q_2}\cdot\cdot\cdot M_ne^{S_nq_n}$$

The inverse kinematics of an open chain or the kinematics of closed chains can be solved by extracting constraint equation and solving it by iterating forward kinematics. The formatter visits all the elements in the robot and by analyzing them, constitutes constraint equations. The extracted data are stored in the form of a map object, i.e. an array of arrays, and each array includes the specific parameters for each loop. The type of constraint can be divided into joint constraint and gear constraint: the former denotes constraints made by constrained joint sets while the latter denotes constraints made by transmission elements such as gears, belts, and chains (collectively denoted as CGear object).

3.1.1 Joint Constraints. Kinematic and dynamic analysis of chains containing closed loops requires the solution of equations of the form $f_i(q)=M_i$, where f_i: $\mathbb{R}^n \rightarrow SE(3)$ denotes the ith loop closure constraints and $i=1, \ldots, m$. Typically these equations will have a family of solutions whose dimension is given by the degrees of freedom of the kinematic chain under study. Closed form solutions of these kinds of problem are usually not available. One must therefore resort to numerical methods such as the Newton-Raphson method. For this purpose, we present a geometric version of the Newton-Raphson method that takes into account the Lie group structure of $SE(3)$, and results in an

efficient and robust iterative algorithm with guaranteed quadratic convergence in the neighborhood of the solution.

Given a system of nonlinear equations f: $\mathbb{R}^n \rightarrow \mathbb{R}^n$, recall that the Newton-Raphson method of root-finding requires repeating the following step until convergence occurs:

$$x^{i+1} = x^i - \left(\frac{\partial f}{\partial x}\right)^{-1}\Bigg|_{x=x^i} \mathrm{f}(\mathrm{x}^i) \tag{1}$$

Now consider the constraint equations for a six degree of freedom articulated rigid body system of the form f: $\mathbb{R}^6 \rightarrow SE(3)$:

$$f(q) = T_1(q_1)T_2(q_2)\cdots T_6(q_6) \tag{2}$$

$$= M, \tag{3}$$

where T_i is the homogeneous transformation from the ith body frame to the next body, and q_i is a joint variable associated with the ith body.

Let $J_i = f^{-1}\partial f/\partial q_i$ denote the column vector representation of the ith body Jacobian of f and $J = [J_1 \cdots J_6]$. The first order Taylor series expansion of f is given by

$$f(q+\delta q) = f(q)e^{j(q)\delta q} \tag{4}$$

Therefore in order to solve $f(q) = M$ by a Newton-Raphson type iteration, set $q^{i+1} = q^i + \delta q^i$ and solve

$$M = f(q^i + \delta q^i) = f(q^i)e^{J(q^i)\delta q^i} \tag{5}$$

for δq^i. This leads to

$$J(q^i)\delta q^i = \log(f^{-1}(q^i)M). \tag{6}$$

If $J(q^i)$ is invertible, then the iteration is

$$q^{i+1} = q^i + J^{-1}(q^i)(\log(f^{-1}(q^i)M)). \tag{7}$$

While the method does not guarantee the existence of J^{-1} and global convergence, it is very powerful if an initial guess is satisfactory, because it converges quadratically near to solutions. In our case, tracking the time history of the state can be easily done; hence we can assume a fairly nice initial guess from the previous step of the state.

3.1.2 Gear Constraints. A gear loop consists of a gear and surrounding revolute joints. All the joints in the gear loop are constrained by a one degree of freedom geared connection, so the the constraint equation of a gear loop has a scalar form containing joint variables. The constraint equation of a gear loop can be expressed as an inner product between the coefficient vector and joint variable vector:

$$k \cdot q = 0 \quad k, q \in \mathbb{R}^n$$

$$k_0 q_0 + k_1 q_1 + \ldots + k_n q_n = 0$$

To represent the above equation, the data formats for the gear constraint consist of three maps: the map of joint indices, the map of coefficients, and the map of scalar joint variables.

In general, values of k will be ± 1 or $\pm r$ (indicating the gear ratio); the coefficients are ± 1 on one side of the gear loop, and $\pm r$ on the other side. The base link of the gear loop (denoted the carrier as in references [13, 14]) is a link which divides these coefficients. To find the base link and derive the coefficient vector, we must analyze the configuration of the gear loop, for which several assumptions are needed to generalize and categorize the types of gear connection.

- **Assumption 1:** The gear loop consists of two collinear axes, and the base link is a link in which two collinear axes meet.

Supposing Assumption 1 is true, we can find the base link by the following procedure:

- Calculate the homogeneous transformations to all the joints in the gear loop seen from the left/right frame of the gear.
- Check all the joint axes in the gear loop; if the axis is collinear with the left gear frame, its coefficient is ± 1, while the right gear frame is $\pm r$.
- The link at which two different axes meet is the base link.

Once the base link is found, we must determine the signs for each coefficient. The signs vary with the configuration of the gear loop, and the following three checkpoints should be considered:

- **Checkpoint 1:** Check the left/right directions of each joint in the gear loop. If the left/right direction is aligned in the direction from the base to the gear, the sign is positive.
- **Checkpoint 2:** Check the directions of the collinear joint axes in the gear loop.
- **Checkpoint 3:** Check the angular directions of the two axes in the gear loop with respect to the contact point.

The angular direction of the axis with respect to the contact point determines the sign of the coefficients on one side. In considering the angular directions we need another assumption:

- **Assumption 2:** If the two axes of the gear are aligned in the same angular direction with respect to the contact point, the signs of the r coefficients are multiplied by -1.

This assumption is based on the fact that the global velocities of the geared pair at the contact point are equal.

After verifying these checkpoints we can complete the constraint equations of the gear loop. For example, the graph representation and corresponding coefficient vectors for a well-known geared mechanism, the Bendix wrist mechanism are illustrated in Figure 6. (Checkpoints 1, 2 are omitted in the figure.)

3.2 Data Formats for Dynamic Analysis

The Newton-Euler dynamics of an n degree of freedom serial chain mechanism can be expressed recursively in coordinate-invariant form as follows:[1]

- **Initialization**

$$V_0, \dot{V}_0, F_{n+1}$$

- **Forward recursion: for k = 1 to n**

$$f_{k-1,k} = M_k e^{S_k q_k} \tag{8}$$

$$V_k = Ad_{f_{k-1,k}^{-1}} \dot{V}_{k-1} + S_k \ddot{q}_k \tag{9}$$

$$\dot{V}_k = Ad_{f_{k-1,k}^{-1}} \dot{V}_{k-1} + S_k \ddot{q}_k + ad_{V_k} S_k \dot{q}_k \tag{10}$$

- **Backward recursion: for k = n to 1**

$$F_k = Ad^*_{f_{k,k+1}^{-1}} F_{k+1} + J_k \dot{V}_k - ad^*_{V_k} J_k V_k \tag{11}$$

$$\tau_k = S_k^T F_k \tag{12}$$

where V_k is the six-dimensional generalized velocity of link {k} expressed in frame {k}, F_k is the six-dimensional generalized force exerted by link {k − 1} on link {k} expressed in frame {k}, J_k is the 6×6 generalized inertia matrix of link {k}, and T_k is the torque at joint {k}. To evaluate the dynamics, information about generalized inertia matrices, generalized forces applied to each link, and joint torques are required.

Compared to open chain systems, inverse dynamics solution for closed chain systems may not be unique without an appropriate choice of actuators. For this reason we consider closed chain systems assuming that only the generalized coordinates are actuated so as to guarantee an unique solution. For this purpose, consider a *reduced system* which is an open chain system but equivalent to the closed chain system. Figure 7 shows that an arbitrary closed loop system with only the generalized coordinates actuated (in Figure 7(a) denoted by shaded circles) can be modified to a reduced system (Figure 7(b)) by cutting a constraint loop and applying actuation to all the joints. As a result, the reduced system can behave exactly the same as the original system.

Let q_u and q_v denote the generalized and dependent coordinate joint values of the original system, respectively. Similarly let τ_{ou} and τ_{ov} denote the related joint torques of

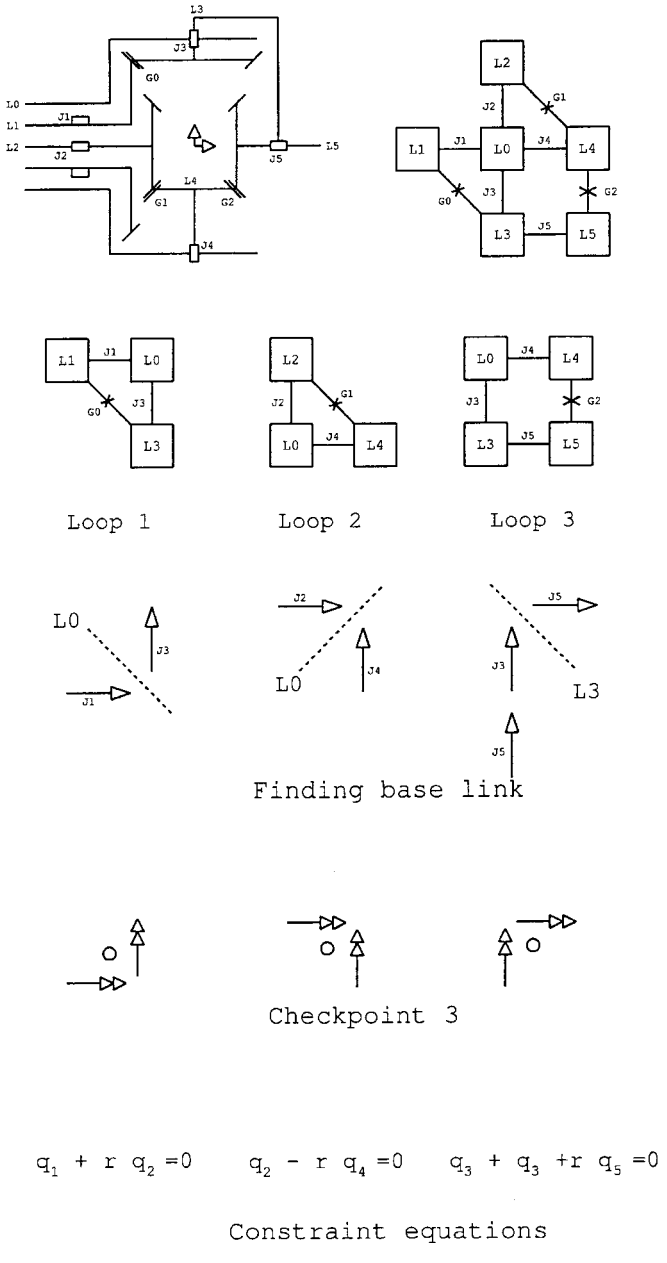

Fig. 6. Example of constructing gear constraints: the Bendix mechanism.

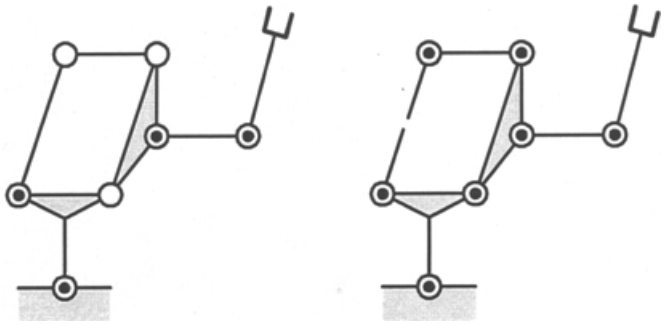

Fig. 7. Construction of the reduced system.

the original system and τ_{ru} and τ_{rv} denote the corresponding joint torques of the reduced system, respectively. Then linear relationship between $\dot{q}_u$ and $\dot{q}_v$ can be described as

$$\dot{q}_u = \Phi \dot{q}_u. \tag{13}$$

By the d'Alembert's principle, work done by the reduced system should now be equal to work done by the original system; $\delta W = \langle \tau_{ou}, \delta q_u \rangle = \langle \tau_{ru} + \Phi \tau_{rv}, \delta q_u \rangle$.

Finally we have the inverse dynamics solution for the closed chain systems as

$$\tau_{ou} = \tau_{ru} + \Phi^T \tau_{rv}. \tag{14}$$

Note that τ_{ru} and τ_{rv} can be easily computed because these are joint torques of the reduced system and the reduced system is open chain.

In general, the number of solutions can vary under the location of actuators. Let τ_a denote actuated joint torques and $\dot{q}_a = \Psi^T \dot{q}_u$, then

$$\tau_u = \Psi \tau_\alpha. \tag{15}$$

Hence the existence of τ_a completely depends on the rank of Ψ. If the rank of Ψ is bigger than a number of rows – a redundantly actuated system – then τ_a can naturally be indeterminate. In this case, we consider e.g. the minimum norm torque solution $\tau_a = \Psi^\dagger \tau_u$, where the small 'p' denotes a Pseudo inverse.

3.3 Functional Design of Formatter

The formatting procedure begins with the function *Initialize*(). *Initialize*() arranges the model as a tree structure by a depth-first-search (DFS) algorithm and extracts the configuration data. Once the initializing process is completed, the functions *BuildLM*(), *BuildKM*(), *BuildGM*() and *BuildDD*() are invoked at the base of the configuration data. The functions build the set of local data for kinematic and dynamic analyses. The functions *BuildLM*() and *BuildKM*() manage the formatting procedure for joint constraint analysis. The function *BuildGM*() manages the formatting procedure for gear constraint analysis, and builds the coefficient vectors of each gear loop. *BuildDD*() manages the formatting procedure for dynamic analysis. The functional diagram of the overall formatting procedure is illustrated at Figure 8.

4. CASE STUDY

Figure 9 presents the HR120 industrial robot manufactured by Hyundai Heavy Industries. The HR120 consists of one closed loop and several gear trains that actuate each joint. Constructing the complete internal model of the HR120 at once is clearly a difficult task; a modular approach to model construction is desirable.

The complete model of the HR120 can be divided into two design modules, the base and wrist, each of which can be designed independently. Moreover, each design module can be represented at three levels of hierarchy. Figure 10 and Figure 11 show the graphical representations of the wrist and the base structures. Using the design encapsulation function, the two parts can be simplified as two modules which have two respective gateway ports. By connecting these ports the full model of the HR120 can be modeled conveniently (see Figure 12). The user can test a new wrist model by connecting the newly designed wrist module to the original base part module and switching the design level of each module in the analysis procedure as appropriate.

Figure 13 shows the two design modules which are modeled independently, while Figure 14 shows the complete model of the HR120 obtained by connecting the two design modules.

Figure 15 shows the workspace of the HR120 obtained by analyzing the kinematics of the HR120 model represented at high level. Generally, analyzing the workspace is a repetitive procedure of evaluating the inverse kinematics with respect to the end-effector adjusted to a specific position and orientation. Analyzing workspace for a whole six-dimensional task object space requires a heavy computation, and, though several algorithms for reducing computation cost were developed such as Haug's algorithm,[15] it remains as a major bottleneck of robot kinematic design.

In our case, we use a refined version for Haug's algorithm[16] and reduce the complexity, taking a high level representation of the hierarchical structure. As shown in Figure 16, the kinematic simulation result by high level is slightly distorted compared with the low level model because of gear interference.

Figure 17 shows the dynamics simulation results of RSTATION for a set of given motion profiles, together with the results obtained from ADAMS, a commercial multibody systems simulation program. A low level model is selected for RStation and an equivalent model with the same kinematic topology is chosen for ADAMS.

In addition to these performancial capacities, we have to pay attention to the modeling capability of our simulator; ADAMS cannot express gear mechanisms that have more than four elements in the loop (e.g. *G64*, *G63* in Figure 10); so it adds some dummy joints in the gear loop.

Figure 16 shows the simulation results for the tip joint (*R1*) torque, performed at three levels with the motion profiles given above. The low level graph shows some discrepancy from the others because of distortion effects caused by interference between geared machine elements that is not considered at the other two levels. But as we mentioned in workspace analysis, this discrepancy can be allowed if the operator wants just a rough result. Figures 18 and 19 (enlarged) show the simulation results for the base joint (*S*) torque, again performed at three levels for the motion profiles given above. We can see that the low level

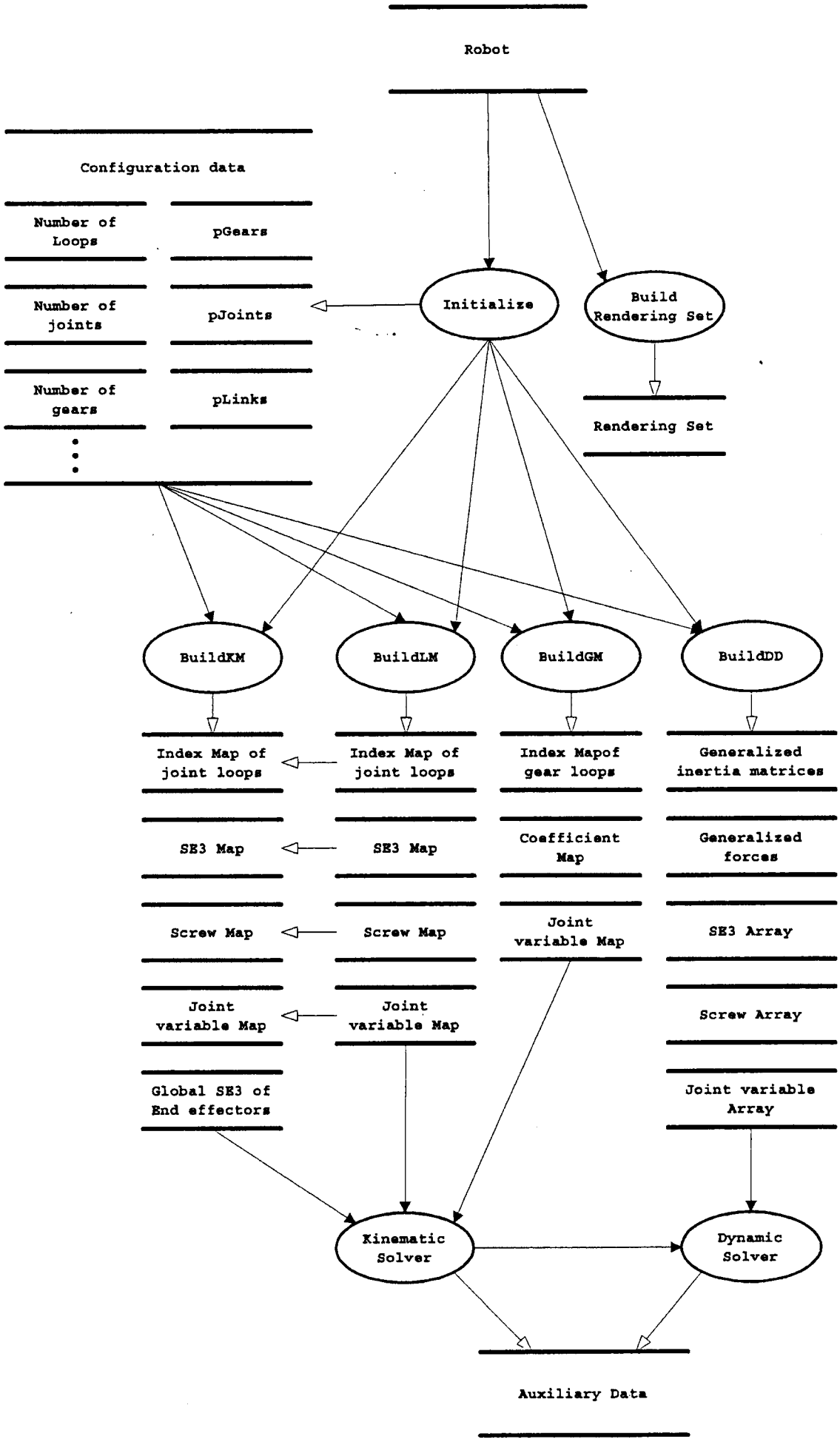

Fig. 8. Functional diagram of formatter.

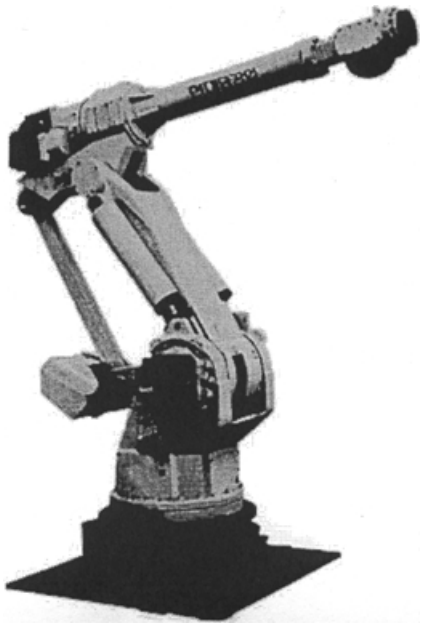

Fig. 9. Hyundai HR120 industrial robot.

result is remarkably different from the results of the other two levels because of the L/IN gyroscopic effects of the rotating machine elements and the forces generated by the spring and damper units. The difference between the intermediate and high level result is relatively small, because, for this part of the assembly, the masses and inertias of the machine elements are much smaller than those of the links. When the relative weight of these

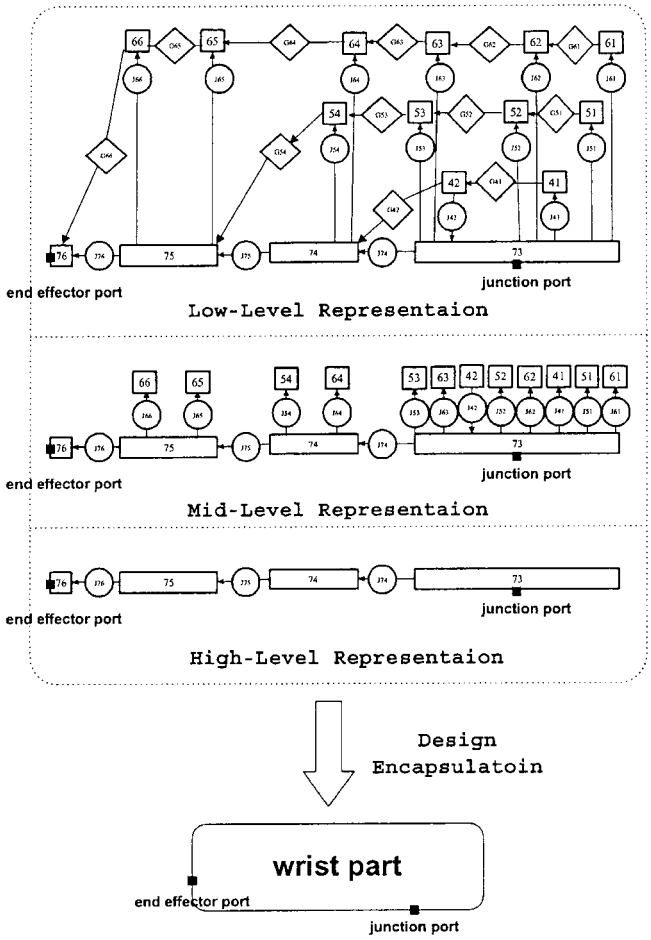

Fig. 10. Multi-level graph representation of wrist module of the HR120.

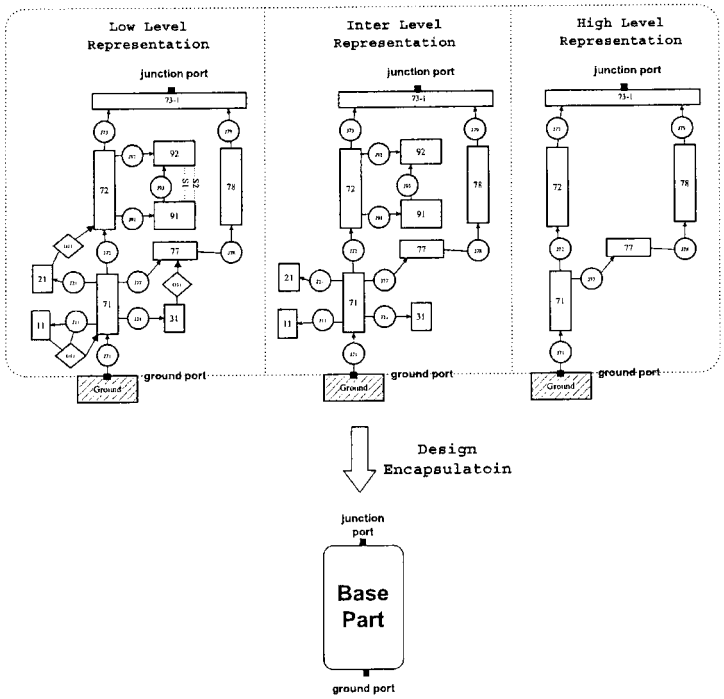

Fig. 11. Multi-level graph representation of base module of the HR120.

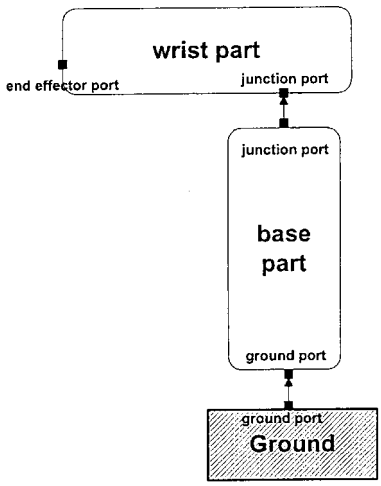

Fig. 12. The full model of the HR120.

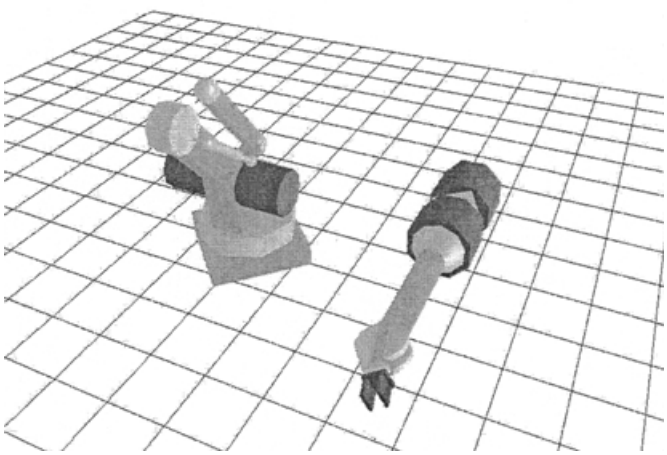

Fig. 13. Two design modules for the HR120 model.

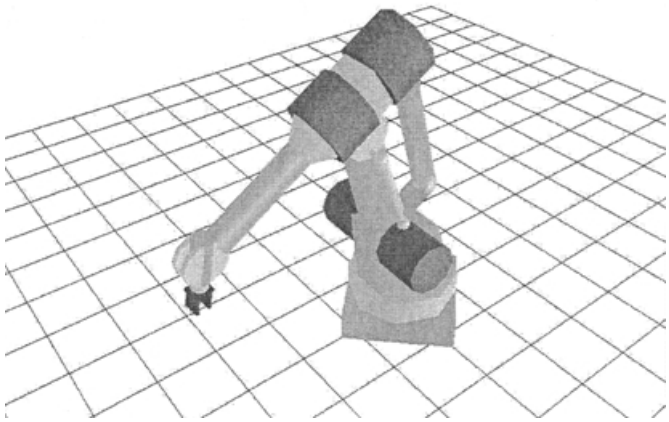

Fig. 14. HR120 model.

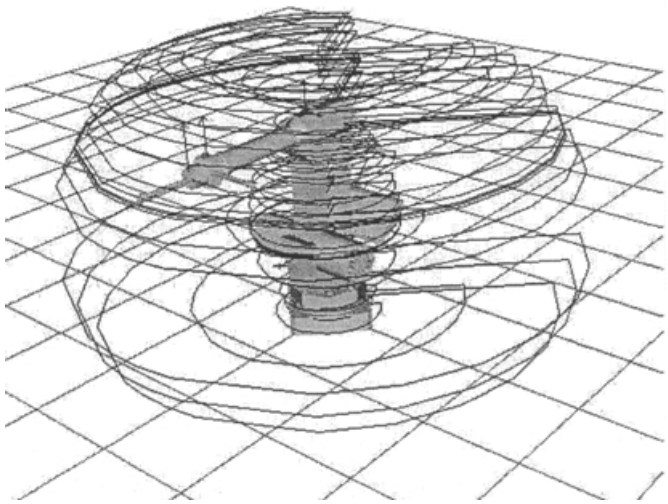

Fig. 15. Workspace of the HR120.

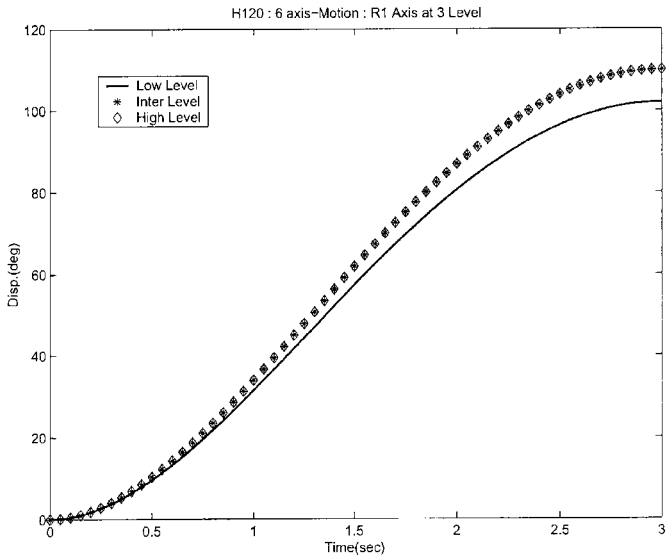

Fig. 16. Multi-level simulation results of the HR120 kinematics: R1 joint.

properties of the machine elements increases, we can expect to see an increase in these differences. Table I shows the computation time for each level; we can verify the trade-off between the computation time and the precision of analysis. In this result, interestedly, the vibrating effects are detected only in the low level simulation. It shows that the vibration effects of a robot system are mainly generated from the transmission elements, and we infer that the low level representation must be regarded for analysis vibrational properties. Similar with the workspace analysis, the developed simulator can regulate the trade-off between computational efficiency and accuracy; the size of actuator can be determined by simulating many tasks at the intermediate level and accurate work paths can also be generated by simulating at the low level representation. In the future, advanced control methodologies such as motion optimization or time-optimal path generation can be the simplest high level representation will be used effectively.

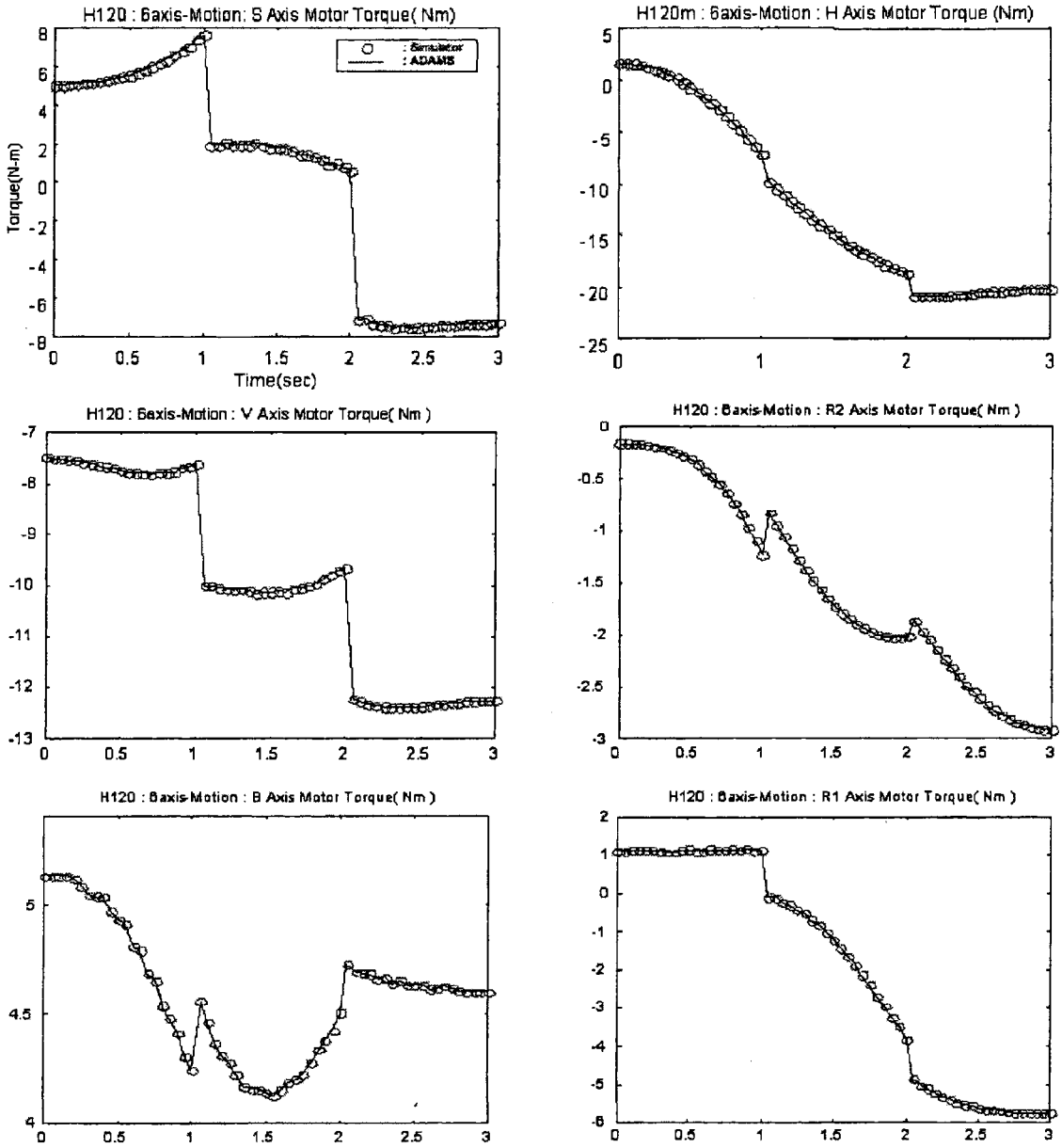

Fig. 17. Simulation results of the HR120 dynamics in comparison with ADAMS.

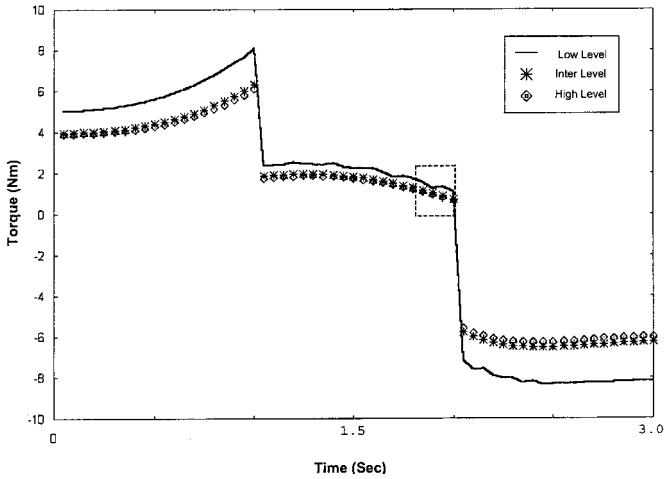

Fig. 18. Multi-level simulation results of the HR120 dynamics: S joint.

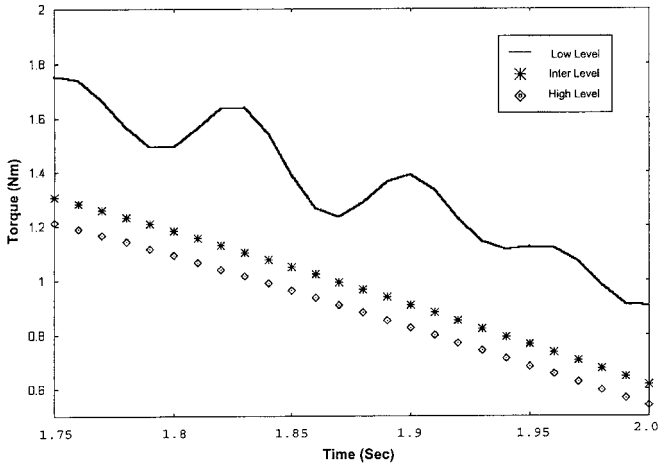

Fig. 19. Multi-level simulation results of the HR120 dynamics: S joint (enlarged).

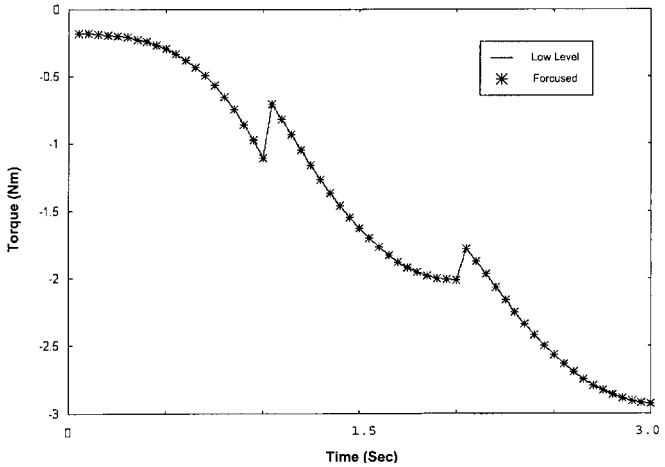

Fig. 20. Simulation results of the focused-dynamics in comparison with the low-level dynamics of the HR120: R2 joint.

Table I. Comparison of the computation times for multi-level dynamic simulation.

	Computation time
High level	0.05
Inter level	0.88
Low level	1.54

Table II. Comparison of the computation times for dynamic simulation of the R2 joint.

	Computation time
Focused	0.44
Full	1.54

The benefits of modular design and hierarchical analysis capability can be largely emphasized when these two functions are combined. We can vary the resolution of model and focus on the module in which we are interested by tuning it to a high level and others to a low level. Figure 20 shows the focused dynamic simulation results for the first wrist joint(R2). Because the analysis is focused on the R2 joint in the wrist group, it make sense to simplify the base group by declaring it as a high-level entity for computational efficiency. Table II confirms the computational advantages of the hierarchical design and analysis feature.

5. CONCLUSIONS

In this paper we have proposed an object-oriented framework for a robot simulator. The developed simulator is module-based, hierarchical, and supports various modeling and analysis functions that provide the robot developer with an efficient and flexible modeling and design environment. Features such as modular design and hierarchical analysis can be realized effectively by using an object-oriented methodology. The computational advantages of recently developed geometrical algorithms specialized for robotic manipulation were exploited in the design of the data types and classes. The object-oriented environment is also ideally suited for realizing one of the main contributions of this work, namely, a hierarchical analysis capability of the robot's dynamics at three independent levels. The introduction of ports as a data class also makes possible a number of additional features, including arbitrarily setting various submodules of the system to different dynamics levels during the same simulation. We also develop an algorithm for extracting the constraint equations for gear systems.

The user interface, graphic interface and plotting function are currently under development, and further analysis functions such as design parameter optimization, workspace and manipulability analysis, collision detection and vibration analysis will be pursued in the near future. Compatibility with other commercial tools, CAD software to import the shape information, and control software (Matlab, Simulink) to provide the virtual plant for control simulation also remain tasks for further development. A

demonstration of the current version of RSTATION can be seen at http://robotics.snu.ac.kr.

ACKNOWLEDGEMENTS

This research was supported in part by the CAD/CAM NRL, the BK21 Program in Mechanical Engineering, and the College of Engineering Research Foundation at Seoul National University.

References

1. F.C. Park, J.E. Bobrow and S.R. Ploen, "A Lie group formulation of robot dynamics", *Int. J. Robotics Research* **14**(6), 609–618 (1995).
2. F.C. Park, J. Choi and S.R. Ploen, "Symbolic formulation of closed chain dynamics in independent coordinates", *Mechanism and Machine Theory* **34**(5), 731–751 (1999).
3. S.R. Ploen and F.C. Park, "Coordinate-invariant algorithms for robot dynamics", *IEEE Trans. Robotics*, **15**(6), 1130–1135 (1999).
4. S. Khoshafian and R. Abnous, *Object Orientation* (Wiley, second edition, 1995).
5. J.K Zao, "Finite-Precision Representation and Data Abstraction for Three-Dimensional Euclidian Transformation", Ph.D. Thesis (Harvard University, Cambridge, Mass., 1995).
6. I.G. Kang and F.C. Park, "Data structures and algorithms for robotics simulation", Ms Thesis (Seoul National University, 1998).
7. A. Kecskemethy and M. Hiller, "An object-oriented tool-set for the computer modeling of vehicle dynamics", *Proc. 26th Int. Symp. Automotive Technology and Automation*, Aachen, Germany (13–17 September 1993), pp. 174–179.
8. A.Kecskemethy, "Sparse-matrix generation of Jacobians for the object-oriented modeling of multibody dynamics", *Proc. NATO Advanced Study Institute on Computer Aided Analysis of Rigid and Flexible Mechanical Systems*, Troia, Portugal (1993), pp. 71–90.
9. G. Ferretti, S. Filippi, C. Maffezzoni, G. Magnani and P. Rocco, "Modular dynamic virtual-reality modeling of robotic systems", *IEEE Robotics and Automation Magazine*, 13–23 (December, 1999).
10. D.B. Stewart, R.A. Volpe and P.K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects", *IEEE Trans. Software Engineering* **23**(12), 759–776 (1997).
11. R.M. Murray, Z. Li and S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation* (Boca Raton, CRC Press, 1993).
12. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design* (Englewood Cliffs, Prentice Hall, 1991).
13. L.W. Tsai, *Robot Analysis* (John Wiley & Sons, 1999).
14. L.W. Tsai, Y.C. Huang and D.W. Duh, "Geared Robot Manipulators with a Joint Unit: Topological Synthesis and Its Application", *Int. J. Robotics Research* **19**(2), 183–194 (2000).
15. E.J. Haug, C.M. Luh, F.A. Adkins and J.Y. Wang, "Numerical Algorithms for Mapping Boundaries of Manipulator Workspaces", *J. of Mechanzcal Design* **118**(2), 228–234 (1996).
16. J.-L. Back, C.-C. Iurascu and F.C. Park, "Finding the Maximally Inscribed Rectangle in a Robot's Workspace", *KSME International Journal* **15**(8), 1119–1131 (2001).