Abstract

A modular robotic system consists of standardized joint and link units that can be assembled into various kinematic configurations for different types of tasks. For the control and simulation of such a system, manual derivation of the kinematic and dynamic models, as well as the error model for kinematic calibration, require tremendous effort, because the models constantly change as the robot geometry is altered after module reconfiguration. This paper presents a framework to facilitate the model-generation procedure for the control and simulation of the modular robot system. A graph technique, termed kinematic graphs and realized through assembly incidence matrices (AIM), is introduced to represent the module-assembly sequence and robot geometry. The kinematics and dynamics are formulated based on a local representation of the theory of Lie groups and Lie algebras. The automatic model-generation procedure starts with a given assembly graph of the modular robot. Kinematic, dynamic, and error models of the robot are then established, based on the local representations and iterative graph-traversing algorithms. This approach can be applied to a modular robot with both serial and branch-type geometries, and arbitrary degrees of freedom. Furthermore, the AIM of the robot naturally leads to solving the task-oriented optimal configuration problem in modular robots. There is no need to maintain a huge library of robot models, and the footprint of the overall software system can be reduced.

1. Introduction

A modular reconfigurable robot consists of a collection of individual link and joint components that can be assembled into a number of different robot geometries. Compared to a conventional industrial robot with fixed geometry, such a system can provide flexibility to the user to contend with a wide spectrum of tasks, through proper selection and reconfiguration of a large inventory of functional components. Several prototyping systems have been demonstrated in various research institutions (Cohen et al. 1992; Fukuda and Nakagawa 1988; Schmitz, Khosla, and Kanade 1988; Wurst 1986). Applications of modular systems have been proposed in rapid deployable robot systems for hazardous material handling (Paredis et al. 1995), in space-stationed autonomous systems (Ambrose 1995), and in manufacturing systems (Chen and Yang 1996).

In the control and simulation of a modular reconfigurable robot system, precise kinematic and dynamic models of the robot are necessary. However, classical kinematic and dynamic modeling techniques for robot manipulators are meant for robots with fixed geometries. These models must be manually derived and individually stored in the robot’s controller prior to simulating and controlling the robot. Commercial robot-simulation software usually provides end users with a library of predefined models of existing robots. The models of any new robot not in the library have to be derived exclusively from the given parameters and commands in the package. For a modular robot system built upon standard modular components, the possible robot geometries and degrees of freedom become huge. As shown by Chen (1994), the number of
robot-assembly configurations grows exponentially when the module set becomes large and the module design becomes complicated. To derive all of these models and store them as library functions require not only tremendous effort, but also large amounts of disk-storage space. In such cases, it is impractical and almost impossible to obtain the kinematic and dynamic models of a robot based on the fixed-geometry approach. Hence, there is a need to develop an automatic model-generation technique for modular robot applications.

In this article, we introduce a framework to facilitate the model-generation procedure for the control and simulation of modular robots. The framework consists of three parts: a component database; a representation of modular robot geometry; and geometry-independent modeling techniques for kinematics, dynamics, and calibration. The component database maintains the description and specifications of standard robot components, such as actuators, rigid links, sensors, and end effectors. The robot representation indicates the types and orders of the robot components being connected. The geometry-independent modeling algorithms then generate the proper models, based on the robot description.

A graph-based technique, termed the kinematic graph, is introduced to represent the module-assembly sequence and robot geometry. In this graph, a node represents a connected joint module and an edge represents a connected link module. Modules attached to or detached from the robot can be indicated by adding or removing nodes or edges from the graph. The realization of this graph is through an assembly incidence matrix (AIM) (Chen and Burdick 1993; Chen 1994). A modular robot can be conceived according to a given AIM without knowing the other parameters, such as joint angles and initial positions. Here, we assume the generic structure of a modular robot is branch-type. The serial-type modular robot is a special case of the branch-type structure.

Previous attempts to deal with automatic model generation for modular robots employed Denavit-Hartenburg (D-H) parameterization of the robot (Kelmar and Khosla 1988; Benhabib, Zak, and Lipton 1989). However, the D-H method does not provide a clear distinction between the arranging sequence of the modules in the robot chain and their spatial relationships. Also, it depends on the initial position of the robot: the same robot may have different sets of D-H parameters just because of the different initial or zero positions. When evaluating the task performance of a modular robot with respect to its corresponding geometry, complicated equivalence relationships must be defined on the sets of parameters to identify the uniqueness of the robot geometry (Chen and Burdick 1993).

The formulation of the kinematics and dynamics is based on the theory of Lie groups and Lie algebras. The robot kinematics follows a local representation of the product-of-exponential (POE) formula, in which the joints, regardless of the types, are defined as members of se(3), the Lie algebra of the Euclidean group SE(3). The associated Lie algebraic structure can simplify the construction of the differentials of the forward-kinematic function required for numerical inverse solutions. The POE representation can also avoid the singularity conditions that frequently occur in the kinematic calibration formulated by the D-H method (Chen and Yang 1997). Thus, it provides us with a uniform and well-behaved method for handling the inverse kinematics of both calibrated and uncalibrated robot systems. Since the joint axes are described in the local module (body) coordinate systems, it is convenient for progressive construction of the kinematic model of a modular robot, as it resembles the assembling action of the physical modular robot components. The formulation of the dynamic model is started with a recursive Newton-Euler algorithm (Hollerbach 1980; Rodriguez, Jain, and Kreutz-Delgado 1991). The generalized velocity, acceleration, and forces are expressed in terms of linear operations on se(3) (Murray, Li, and Sastry 1994). Based on the relationship between the recursive formulation and the closed-form Lagrangian formulation for serial-robot dynamics discussed in (Featherstone 1987; Park, Bobrow, and Ploen 1995), we use an accessibility matrix (Deo 1974) to assist in the construction of the closed-form equation of motion for a branch-type modular robot, which we assume is the generic topology of a modular robot. Note that all the proposed modeling techniques can contend with redundant and nonredundant modular robot configurations.

This article is organized as follows. Section 2 introduces the basic features of the hardware and software of a newly conceived modular robotic work cell. Section 3 briefly reviews the definitions of the AIM presentation and the associated accessibility matrix and path matrix. Section 4 concerns the formulation and implementation of geometry-independent kinematic, dynamic, and calibration models for modular robots. In addition to automated model generation, identification of the optimal modular robot assembly geometry for a specific task from the vast candidate database is also important. The AIM representation facilitates the search/optimization process by using the genetic algorithms approach. Section 5 investigates the task-oriented optimal geometry issues in modular reconfigurable robots, and the advantages of using AIM to solve this type of problem. Section 6 summarizes the automated model-generation techniques.

2. System Architecture

Figure 1 shows a conceptual design of a modular work cell that is currently under construction at Nanyang Technological University and Gintic Institute of Manufacturing Technology, Singapore. The objective of this project is to design, simulate, and construct a reconfigurable modular manufacturing work cell that is capable of performing a variety of tasks, such as part assembly, material transfer, and light machining (grinding, polishing, and deburring), through a rapid change
of reusable work-cell components. The work-cell hardware is built around modular reconfigurable robot components, such as actuators, links, sensors, end effectors, and other tools. Each work cell may comprise several modular robots of different geometries to perform different tasks, based on their requirements.

The work-cell software is also designed in reusable- and reconfigurable-object fashion, for ease of maintenance and development. Figure 2 illustrates the overall software architecture of the modular work cell. The user environment provides all of the necessary functions to facilitate the end user in controlling, monitoring, and simulating the work cell. It consists of the following parts:

- Component browser—for viewing and editing the components available in the component database;
- Simulator—for generating a computer-simulation model of a modular robot and the entire work cell; additionally, the simulator may be employed as the core function for future virtual manufacturing capabilities;
- Task-level planner—for determining the optimal geometry of a modular robot for a given task and the overall layout of the work cell for a particular manufacturing process;
- Programming interface—for providing command and control of the system; and
- Controller—for commanding the low-level individual controllers located in the components, and identifying the robot’s geometry from the local component controllers.

The system kernel, which is hidden from the user, provides automated model-generation functions and the configuration-optimization function (a component database is also associated with it):

- Object-oriented component database—manages the specification of all the components, such as the dimensions and weights of the joints and links, maximum kinematic and dynamic performance of the actuators, etc. It can be accessed by the user for browsing and editing purposes.
- Geometry-independent kernel functions—generate kinematic and dynamic models of the robots shared by the simulators and the controller. Using identical models in the simulation and control of the work cell ensures the reliability and integration of the system, and enables physically based simulations through the work-cell controller. The configuration-optimization function can enumerate all possible robot geometries from an inventory of module components in the database, and select the most suitable one for a prescribed task. This information will pass back to the task-level planner to determine the optimal layout and locations of the robots in the work cell.

The information passing from the component database to the modeling functions is through the assembly incidence matrix. Robot geometries (serial, branch, or hybrid) and detailed connection information, such as the connecting orientation and the types of adjacent modules, are all indicated in the matrix. This matrix is then passed to the geometry-independent functions for model generation.

In such a system, the need to maintain a huge library of robot models is eliminated; instead, we retain a small selection of the component-database and kernel functions for automated model generation, reducing the overall footprint of the system software.

3. Modular Robot Representation

3.1. Module Representation

To make the automatic model-generation algorithms work on a variety of module components, we introduce a conceptual set of modules whose features are extracted from those of real implementations. The modular systems developed to date have several common mechanical and structural features: (1) consideration of only 1-DOF revolute and 1-DOF prismatic joints; (2) symmetric link geometries for interchangeability; and (3) multiple connection ports on a link.

3.1.1. Joint Modules

A modular robot joint module is an “active” joint, which allows the generation of a prescribed motion between connected links. Two types of joint modules, the revolute joints (rotary motion) and the prismatic joints (linear or translational motion), are considered. Rotary and linear actuators must reside in the modules to produce the required motions and maintain the modularity of the system. Multi-DOF motions can be synthesized with several 1-DOF joints. Joint modules are attached to link modules through standardized connecting interfaces for mechanical, power, and control connections.
Fig. 2. Software architecture for the work cell.

3.1.2. Link Modules

The place on a link module where the joint is connected is called a connecting port. Without loss of generality, we assume that a link module is capable of multiple joint connections, and the link module has symmetrical geometry. Such a design allows modules to be attached in various orientations, and the robot geometry to be altered via simple reassembling. The modular robot components currently under construction at our university are shown in Figure 3. This design follows the building-block principle whereby modules can be stacked together in various orientations through connecting points on all six faces of the cubes.

3.2. Assembly Incidence Matrix

DEFINITION 1. (Graph) A graph \( G = (V, E) \) consists of a vertex set, \( V = \{v_0, \ldots, v_n\} \), and an edge set, \( E = \{e_0, \ldots, e_m\} \), such that every edge in \( E \) is associated with a pair of vertices, i.e., \( e_i = (v_j, v_k) \).

In mechanism design theory, a kinematic chain of links and joints is often represented by a graph, termed a kinematic graph (Dobrjanskyj and Freudenstein 1967), in which vertices represent the links, and edges represent the joints. Using this graph representation, we are able to categorize the underlying structure (or geometry) of a linkage mechanism and apply the result from the graph theory to enumerate and classify linkage mechanisms. A robot manipulator is also a kinematic chain, thus admitting a kinematic graph representation. For example, an 8-module 7-DOF branch-type modular robot and its kinematic graphs are shown in Figures 4 and 5, respectively. It is also known that a graph can be represented numerically as a vertex-edge incidence matrix, in which the entries contain only 0s and 1s (Deo 1974). Entry \((i, j)\) is equal to 1 if edge \(e_j\) is incident on vertex \(v_i\); otherwise, it is equal to 0. This incidence relationship defines the connectivity of the link and joint modules. Because link modules may have multiple connecting points, we can assign labels to the connecting points to identify the module connections. To further identify those
connections in the incidence matrix, we can replace those entries of 1 by the labels of the connected ports being identified on the link modules, and keep those entries of 0 unchanged. This modified matrix, termed an assembly incidence matrix, provides us with the necessary connection information of the modules and also the basic geometry of the modular robot.

**Definition 2. (Assembly Incidence Matrix)** Let $\mathcal{G}$ be a kinematic graph of a modular robot and $\mathcal{M}(\mathcal{G})$ be its incidence matrix. Let $\text{port}$ be the set of labels assigned to the connecting ports on the link modules. The assembly incidence matrix of the robot $A(\mathcal{G})$ is formed by substituting the 1s in $\mathcal{M}(\mathcal{G})$ with labels in $\text{port}$ on respective modules. One extra column and row are augmented to $A(\mathcal{G})$ to show the types of link and joint modules.

Note that the representation and assignment of the labels are nonunique. The labels of the connecting ports may be numerical values (Chen 1994) or may be derived from the module coordinates (Chen and Yang 1996). In this case, the module-component database should use consistent bookkeeping for this information. The AIM of the modular robot (8 link
modules and 7 joint modules) shown in Figure 4 is a $9 \times 8$ matrix:

$$
A(G) = \begin{bmatrix}
1 & 3 & 5 & 0 & 0 & 0 & 0 & B \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & C1 \\
0 & 1 & 0 & 6 & 0 & 0 & 0 & C1 \\
0 & 0 & 2 & 0 & 6 & 0 & 0 & C1 \\
0 & 0 & 0 & 5 & 0 & 2 & 0 & C2 \\
0 & 0 & 0 & 0 & 5 & 0 & 3 & C2 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & C2 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & C2 \\
P & R & R & R & R & P & P & 0
\end{bmatrix}.
$$

Note that there are three types of link modules in the robot: the base ($B$), the large cubic module ($C1$), and the small cubic module ($C2$). Cubic modules have six connecting interfaces labeled 1–6; i.e., port = {1, ⋯, 6}, which follows the labeling scheme on dice. The revolute joints and prismatic joints are denoted by $R$ and $P$, respectively.

### 3.3. Accessibility and Path Matrices

Two matrices, namely, the accessibility matrix and the path matrix, derived from a given AIM, are defined in this section to provide the accessibility information from the base module to every pendant module in a branch-type modular robot. The accessibility information enables us to formulate the kinematics and dynamics of a general branch-type robot in a uniform way.

#### 3.3.1. The Module Traversing Order

The links and joints of a serial-type robot can follow a natural order from the base to the tip. A branch-type robot has more than one tip, and no loops. Therefore, the order of the links of a branch-type robot depends on the graph-traversing algorithms (Cormen, Leiserson, and Rivest 1990). Let $\tilde{G} = (V, E)$ represent the kinematic graph of a branch-type modular robot with $n + 1$ link modules, where $V = \{v_0, v_1, \ldots, v_n\}$ represents the set of modules. The fixed-base module is denoted by $v_0$, and is always the starting point for the traversing algorithm. The rest of the modules are labeled by their traversing orders, $i$. The traversing orders of the links in the robot of Figure 4 are indicated by the numbers on the vertices of the graph of Figure 5. This order is obtained by the depth-first-search algorithm. Note that the farther the module is away from the base, the larger its traversing order.

#### 3.3.2. Directed Graphs

A branch-type robot with $n + 1$ modules has $n$ joints. Let $E = \{e_1, \ldots, e_n\}$ represent the set of joints, where joint $e_j$ is designated as the connector preceding link module $v_j$. With a given traversing order, the robot graph $\tilde{G}$ can be converted to a directed graph (or digraph) $\tilde{G}$, which is an outward tree for a branch-type manipulator in the following manner. Let $e_j = (v_i, v_j)$ be an edge of the graph $\tilde{G}$, and $i < j$. An arrow is drawn from $v_i$ to $v_j$ as edge $e_j$ leaves vertex $v_i$ and enters vertex $v_j$. Suffice to say, link $v_i$ precedes link $v_j$. An example of the directed graph is shown in Figure 5. From an outward tree with $n + 1$ vertices, an $(n + 1 \times n + 1)$ accessibility matrix can be defined to show the accessibility among the vertices.

**Definition 3. (Accessibility Matrix)** The accessibility matrix of a directed kinematic graph $\tilde{G}$ of a modular robot with $n + 1$ modules (vertices) is an $(n + 1) \times (n + 1)$ matrix, $\mathcal{R}(\tilde{G}) = [r_{ij}]$, $(i, j = 0, \ldots, n)$, such that $r_{ij} = 1$ if there is a directed path of length 1 or more from $v_i$ to $v_j$; and $r_{ij} = 0$ otherwise.

The accessibility matrix can be derived from the AIM once the traversing order on the link modules is determined. For example, the accessibility matrix of $\tilde{G}$ in Figure 5 is

$$
\mathcal{R}(\tilde{G}) = \begin{bmatrix}
v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
v_0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
v_1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
v_2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
v_3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
v_4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
v_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
v_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
v_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.
$$

From $\mathcal{R}(\tilde{G})$, we can obtain the shortest route from the base to the pendant link. This route is called a path. The pendant links are the rows of $\mathcal{R}(\tilde{G})$ with all 0s. The number of paths in a branching robot is equal to the number of pendant links. Let link $v_i$ be a pendant link. All link modules on the path from the base to $v_i$ are shown in the nonzero entries of column $i$ of $(\mathcal{R}(\tilde{G}) + I_{(n+1)\times(n+1)})^T$. Collecting all the paths, we obtain the path matrix, as defined below.

**Definition 4. (Path Matrix)** The path matrix $\mathcal{P}(\tilde{G})$ of a directed kinematic graph $\tilde{G}$ of a branch-type robot with $n + 1$
link modules (vertices) and \( m \) paths is an \( m \times (n+1) \) matrix, 
\[
\mathcal{P}(\tilde{g}) = [p_{ij}] (i = 1, 2, \ldots, m; \ j = 0, 1, \ldots, n) \text{ such that } p_{ij} = 1 \text{ if path } i \text{ contains vertex } j, \text{ and } p_{ij} = 0 \text{ otherwise.}
\]

For instance, the robot of Figure 4 contains three branches (paths). The three paths can be represented as a \( 3 \times 8 \) matrix:

\[
\mathcal{P}(\tilde{g}) = \begin{pmatrix}
    v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
    1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
    1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
    1 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{pmatrix}.
\]

Row 1 represents the branch of the robot containing link modules \( v_0, v_1, v_2, \) and \( v_3 \); row 2 represents the branch of \( v_0 \) and \( v_4 \); and row 3 represents the branch of \( v_0, v_5, v_6, \) and \( v_7 \). It can be seen that the rows of \( \mathcal{P}(\tilde{g}) \) are identical to columns 3, 4, and 7 of \( (\mathcal{P}(\tilde{g}) + I_{(n+1) \times (n+1)}) \), respectively.

4. Geometry-Independent Models

4.1. Forward Kinematics

The forward kinematics of a general branch-type modular robot starts with a given AIM and a dyad kinematic model that relates the motion of two connected modules under a joint displacement. A dyad is a pair of connected links in a kinematic chain. Using dyad kinematics recursively with a prescribed graph-traversing order assigned to the robot modules, we may obtain the forward transformation of every branch with respect to the base frame, having a prescribed set of joint displacements. Note that a branch-type robot is one without any closed-loop geometry. The kinematics of a closed-loop-type robot mechanism requires additional constraints, and is not considered in this article.

4.1.1. Dyad Kinematics

Let \( v_i \) and \( v_j \) be two adjacent links connected by a joint \( e_j \), as shown in Figure 6. Denote joint \( e_j \) and link \( v_j \) as link-assembly \( j \), and the module-coordinate frame on link \( v_i \) as frame \( i \). The relative position (the orientation) of the dyad, \( v_i \) and \( v_j \), with respect to frame \( i \) with a joint angle \( q_j \), can be described by a \( 4 \times 4 \) homogeneous matrix,

\[
T_{ij}(q_j) = T_{ij}(0) e^{\hat{s}_j q_j},
\]

where \( \hat{s}_j \in se(3) \) is the twist of joint \( e_j \) expressed in frame \( j \), \( T_{ij}(q_{ij}) \) and \( T_{ij}(0) \in SE(3) \). \( T_{ij}(0) \) is the initial pose of frame \( j \) relative to frame \( i \). Note that in the following context, the pose of a coordinate frame is referred to as the \( 4 \times 4 \) homogeneous matrix of the orientation and position of a coordinate frame:

\[
T_{ij}(0) = \begin{bmatrix}
R_{ij}(0) & d_{ij}(0) \\
0 & 1
\end{bmatrix},
\]

where \( R_{ij}(0) \in SO(3) \) and \( d_{ij}(0) \in \mathbb{R}^3 \) are the initial orientation and position of link-frame \( j \), relative to frame \( i \), respectively. The twist \( \hat{s}_j \) of link-assembly \( j \) is the skew-symmetric matrix representation of the six-vector line coordinate of the joint axis, \( s_j = (q_j, p_j) \); \( p_j, q_j \in \mathbb{R}^3 \). \( p_j = (p_{jx}, p_{jy}, p_{jz}) \) is the unit-directional vector of the joint axis relative to frame \( j \), and \( q_j = (q_{jx}, q_{jy}, q_{jz}) = p_j \times r_j \), where \( r_j \) is the position vector of a point along the joint axis relative to frame \( j \). For revolute joints, \( s_j = (0, p_j) \); and for prismatic joints, \( s_j = (q_j, 0) \).

4.1.2. Recursive Forward Kinematics

Based on eq. (4), we propose a recursive algorithm for a general branch-type modular robot, termed TreeRobotKinematics. This algorithm can derive the forward transformations of the base link to all pendant links, based on graph-traversing algorithms. The procedure is illustrated in Figure 7. Implementation details can be found in an earlier work (Chen and Yang 1996). The algorithm takes three inputs: the AIM of the robot \( A(G) \), the base-link location \( T_0 \), and a set of joint angles \( \{q\} \). The forward-kinematics calculation follows the breadth-first-search (BFS) traversing algorithm to travel on the connected robot modules.
4.1.3. Path-by-Path Forward Kinematics

A tree-type robot consists of several paths that give the shortest routes from the base to the respective pendant links. Each path can be considered as a serially connected submanipulator, so that the forward transformation can be derived as a conventional industrial manipulator. The sequence of the connected modules in a path is indicated in a row of the path matrix \( P \). Let \( a = \{a_0, a_1, a_2, \ldots, a_n\} \) represent the links of path \( k \). The base is \( a_0 \equiv 0 \), and the number of links in the path \( k \) is defined to be \(|a| = n + 1\). For instance, path 1 of the robot in Figure 4 is \( a = \{0, 1, 2, 3\} \). The forward kinematics from the base to the pendant link \( a_n \) of path \( k \) is given by

\[
T_{a_0a_n} = T_{a_0a_1}(q_{a_1})T_{a_1a_2}(q_{a_2}) \cdots T_{a_{n-1}a_n}(q_{a_n}) = \prod_{i=1}^{n}(T_{a_{i-1}a_i}(0)e^{s_{a_i}q_{a_i}}).
\]

For a branch-type modular robot with several paths, the forward kinematics is

\[
T(q_1, q_2, \ldots, q_n) = \begin{bmatrix}
T_{a_0a_n} \\
T_{b_0b_n}
\end{bmatrix} = \begin{bmatrix}
\prod_{i=1}^{n}(T_{a_{i-1}a_i}(0)e^{s_{a_i}q_{a_i}}) \\
\prod_{i=1}^{m}(T_{b_{i-1}b_i}(0)e^{s_{b_i}q_{b_i}})
\end{bmatrix},
\]

where \( T(q_1, q_2, \ldots, q_n) \) represents the vector of \( 4 \times 4 \) homogeneous matrices of the poses of all the pendant end effectors. Since many paths in the branch-type robot share many common modules, there will be repetitive calculations using the model of eq. (7). In actual implementation, we prefer the recursive approach, which introduces no repetitive calculations.

4.2. Inverse Kinematics

The purpose of an inverse-kinematics algorithm is to determine the joint angles that cause the end effector of a manipulator to reach a desired pose. Current robot inverse-kinematics algorithms can be categorized into two types: closed form and numerical. Closed-form-type inverse kinematics requires a complete parameterization of the solution space, usually in terms of simultaneous polynomial equations. Solutions to such a set of simultaneous polynomial solutions exist for a few types of robots with revolute joints or simple geometry. It is very difficult to obtain the inverse kinematics for an arbitrary modular reconfigurable robot in this manner.

Here we adopt the numerical approach to solve the inverse kinematics of modular robots. The inverse-kinematics algorithm will construct the differential kinematic model using the local POE formula. The differential kinematic equation of a single branch of a branch-type robot is considered first. Based on the AIM of the robot, one can extend this differential kinematics model to include multiple branch structures. Then the Newton-Raphson iteration method is used to obtain the numerical inverse-kinematics solutions. The differential kinematic model can be easily modified to solve the pure-position, pure-orientation, and hybrid-inverse kinematics problems (Yang, Chen, and Kang 1998).

4.2.1. A Single Branch

Let \( T_{a_0a_n} \) be the forward transformation of path \( k \), as indicated in eq. (6). The differential change in the position of the end-link \( a_n \) can be given by

\[
dT_{a_0a_n} = \sum_{i=1}^{[a]} \frac{\partial T_{a_0a_n}}{\partial q_{a_i}} dq_{a_i} = \sum_{i=1}^{[a]} \left[ T_{a_0a_{i-1}} \frac{\partial (T_{a_{i-1}a_i}(0)e^{s_{a_i}q_{a_i}})}{\partial q_{a_i}} \right] \cdot dq_{a_i} = \sum_{i=1}^{[a]} \left[ T_{a_0a_{i-1}} \frac{\partial T_{a_{i-1}a_i}}{\partial q_{a_i}} \right] dq_{a_i}.
\]

Left-multiplying \( T_{a_0a_n}^{-1} \), eq. (8) becomes

\[
T_{a_0a_n}^{-1} dT_{a_0a_n} = \sum_{i=1}^{[a]} T_{a_0a_{i-1}}^{-1} \frac{\partial T_{a_{i-1}a_i}}{\partial q_{a_i}} \frac{\partial q_{a_i}}{\partial q_{a_i}}.
\]

Equation (9) is the differential kinematic equation of a path. Let \( T_{a_0a_n}^d \) denote the desired position of the end effector. When it is in the neighborhood of a nominal position of the end effector, \( T_{a_0a_n} \), we have

\[
dT_{a_0a_n} = T_{a_0a_n}^d - T_{a_0a_n}.
\]

Left-multiplying \( T_{a_0a_n}^{-1} \) to eq. (9), and using the matrix logarithm,

\[
\log(T_{a_0a_n}^{-1} T_{a_0a_n}^d) = (T_{a_0a_n}^{-1} T_{a_0a_n}^d - I) - \frac{(T_{a_0a_n}^{-1} T_{a_0a_n}^d - I)^2}{2} + \frac{(T_{a_0a_n}^{-1} T_{a_0a_n}^d - I)^3}{3} - \ldots.
\]

We can obtain the following equation by a first-order approximation:

\[
T_{a_0a_n}^{-1} dT_{a_0a_n} = \log(T_{a_0a_n}^{-1} T_{a_0a_n}^d). \tag{12}
\]
Substituting eq. (12) into eq. (9), we obtain

\[ \log(T_{\text{adj}}^{-1} T_d^d) = \sum_{i=1}^{[\alpha]-1} T_{\text{adj}}^{-1} S_{\text{adj}} T_{\text{adj}} dq_{\text{adj}}. \] (13)

Explicit formulae for calculating the logarithm of elements of SO(3) and SE(3) were derived by Park and Bobrow (1994). Definitely, \( \log(T_{\text{adj}}^{-1} T_d^d) \) is an element of \( \text{se}(3) \), so that it can be identified by a \( 6 \times 1 \) vector denoted by \( \log(T_{\text{adj}}^{-1} T_d^d) \) in which the first and last three elements represent the positional and orientational differences between \( T_{\text{adj}} \) and \( T_d^d \).

Converting eq. (13) into the adjoint representation, we get

\[ \log(T_{\text{adj}}^{-1} T_d^d)^\top = AdT_{\text{adj}}^{-1} \sum_{i=1}^{[\alpha]-1} AdT_{\text{adj}} dq_{\text{adj}}. \] (14)

Conveniently, eq. (14) can also be expressed as the following form:

\[ DT_k = J_k dq_k, \] (15)

where

\[ DT_k = \log(T_{\text{adj}}^{-1} T_d^d)^\top \in R^{6 \times 1} \] is referred to as the pose-difference vector for path \( k \);
\[ J_k = A_1 B_1 S_k \in R^{6(\alpha-1) \times 1} \] is termed the body-manipulator Jacobian (Murray, Li, and Sastry 1994);
\[ A_k = AdT_{\text{adj}}^{-1} \in R^{6 \times 6}; \]
\[ B_k = \text{row} [AdT_{\text{adj}}^{-1}, AdT_{\text{adj}}^{-2}, \ldots, AdT_{\text{adj}}^{-[\alpha]-1}] \in R^{6 \times 6([\alpha]-1)}; \]
\[ S_k = \text{diag} [s_{\alpha 1}, s_{\alpha 2}, \ldots, s_{\alpha \alpha}] \in R^{6(\alpha-1) \times ([\alpha]-1)}; \] and
\[ dq_k = \text{column} [dq_{\alpha 1}, dq_{\alpha 2}, \ldots, dq_{\alpha \alpha}] \in R^{([\alpha]-1) \times 1}. \]

Equation (15) defines the differential kinematics for path \( k \). It can be utilized in the Newton-Raphson iteration to obtain an inverse-kinematics solution for a given pose.

### 4.2.2. The Entire Manipulator

The paths of a branch-type manipulator may not be independently driven, because of the common sharing modules. This forbids us to treat each path as an independent serial-type manipulator. Hence, with a given set of the pendant end-effector poses for all branches, the inverse kinematics must be solved simultaneously. With the assistance of the path matrix, we are able to identify the connected and related modules in a path. Then, we can orderly combine the differential kinematic equations (eq. (15)) of all constituting paths into a single matrix equation of the following form:

\[ DT = J dq, \] (16)

where

\[ DT = \text{column} [DT_1, DT_2, \ldots, DT_{\text{m}}] \in R^{6m \times 1} \] is termed the generalized pose-difference vector;
\[ J = \text{ABS} \in R^{6m \times n} \] is termed the generalized body-manipulator Jacobian;
\[ A = \text{diag} [A_1, A_2, \ldots, A_m] \in R^{6m \times 6n}; \] and
\[ B = \begin{bmatrix}
   p_{11} AdT_{\text{D}1} & p_{12} AdT_{\text{D}2} & \cdots & p_{1n} AdT_{\text{D}n} \\
   p_{21} AdT_{\text{D}1} & p_{22} AdT_{\text{D}2} & \cdots & p_{2n} AdT_{\text{D}n} \\
   \vdots & \vdots & \ddots & \vdots \\
   p_{m1} AdT_{\text{D}1} & p_{m2} AdT_{\text{D}2} & \cdots & p_{mn} AdT_{\text{D}n}
\end{bmatrix} \in R^{6m \times 6n}.
\]

The coefficient \( p_{ij} \) is an entry \((i, j)\) of the path matrix \( P \), and \( m \) is the total number of paths;
\[ S = \text{diag} [s_{11}, s_{12}, \ldots, s_{nn}] \in R^{6n \times n}; \] and
\[ dq = \text{column} [dq_1, dq_2, \ldots, dq_n] \in R^{n \times 1}. \]

Rewriting this equation in an iterative form, we get

\[ dq^{i+1} = J^* DT, \] (17)

\[ q^{i+1} = q^i + dq^{i+1}, \] (18)

where \( i \) represents the number of iterations, and \( J^* \) is the Moore-Penrose pseudoinverse of \( J \). Using the Newton-Raphson method, a closed-loop iterative algorithm similar to that of Khosla, Neuman, and Prinz (1985) is employed (Fig. 8). The iterative algorithm determines the necessary changes in the joint angles to achieve a differential change in the position and orientation of the end effector. Given a complete robot assembly (or the AIM) and a set of desired poses \( T^d \), this algorithm starts from an initial guess, \( q^0 \), somewhere in the neighborhood of the desired solution. It is terminated when prescribed termination criteria is reached. As one can see, the structure of \( J \) depends on the path matrix, which is implied in the kinematic graph of the robot. Therefore, once the assembly configuration of a modular robot is determined and all module parameters are obtained, the differential kinematic model (eq. (16)) can be generated automatically.

Computational examples of the inverse-kinematics algorithms for branch-type and serial-type modular robots are given by Yang, Chen, and Kang (1998) to illustrate the algorithm’s applicability and effectiveness. When compared to the other numerical inverse-kinematics algorithm using D-H parameters, our method always uses fewer iterations and less computing time for the same given pose. This is due to the use of the pose-difference vector, computed from the matrix logarithm in eq. (16), and not the difference of homogeneous transformation matrices. Actual implementation of the algorithm using C++ code shows that the computation time for
each solution can take less than 20 msec on a Pentium II 300-MHz PC, which satisfies the basic requirement for real-time control and simulation.

4.3. Kinematic Calibration

The machining tolerance, compliance, and wear of the connected mechanism and misalignment of the connected module components may introduce errors in positioning the end effector of a modular robot. Hence, calibrating the kinematic parameters of a modular robot to enhance its positioning accuracy is important, especially in high-precision applications such as hard-disk assembly.

Current kinematic calibration algorithms for industrial robots that are designed for certain types of serial manipulators are not suitable for modular robots with arbitrary geometry. Here we propose a general singularity-free calibration–modeling method for modular reconfigurable robots, based on the forward kinematics discussed in the previous section. This method follows local POE formulae. The robot errors are assumed to be in the initial positions of the consecutive modules. Based on linear superposition and differential transformation, a six-parameter model is derived. This model can be generated automatically once the AIM of the robot is given. An iterative least-squares algorithm is then employed to find the error parameters to be corrected.

The calibration starts with a serial-type manipulator-kinematics model:

\[
T_{0n}(q) = T_{01}(q_1) T_{12}(q_2) \cdots T_{n-1,n}(q_n) 
\]

\[
= T_{01}(0) e^{\xi q_1} T_{12}(0) e^{\xi q_2} \cdots T_{n-1,n}(0) e^{\xi q_n}. 
\]

(19)

(20)

Extension to a general branch-type modular robot is similar to the treatment of the inverse-kinematics model in the previous section. Basically, eq. (20) can be treated as a function of the joint angles, \( q = (q_1, \ldots, q_n) \), locations of the joint axes, \( \tilde{s} = (\tilde{s}_1, \ldots, \tilde{s}_n) \), and the relative initial positions of the dyads, \( T_0 = (T_{01}(0), \ldots, T_{n-1,n}(0)) \):

\[
T_{0n} = f(T_0, \tilde{s}, \tilde{q}). 
\]

(21)

Written in differential form,

\[
dT_{0n} = \frac{\partial f}{\partial T_0}dT_0 + \frac{\partial f}{\partial \tilde{s}}d\tilde{s} + \frac{\partial f}{\partial \tilde{q}}d\tilde{q}. 
\]

(22)

The differential \( dT_{0n} \) can be interpreted as the difference between the nominal position and the measured position.

4.3.1. Error Model of a Dyad

Our kinematic calibration is based on the local frame representation of a dyad, described in eq. (4). Two assumptions are made in the dyad of links \( v_{i-1} \) and \( v_i \) of a modular robot chain: first, small geometric errors only exist in the initial position \( T_{i-1,i}(0) \); second, the twist and joint angle \( q_i \) assumes nominal values throughout the calibration analysis. Hence, instead of identifying the module’s actual initial positions, joint twists, and angle offsets, we look for a new set of local initial positions (local frames, called calibrated initial positions), in the calibration model, so that the twist of the joint remains a nominal value. In other words, the errors in a dyad are lumped with the initial position. Therefore, \( ds \) and \( dq \) can be set to 0. Because SE(3) has the dimension of six—three for positions and three for orientations—there can be only six independent quantities in \( T_{i-1,i}(0) \), and there will be six independent error parameters in a dyad.

Denote the small error in the initial position of dyad \( (v_{i-1}, v_i) \) as \( dT_{i-1,i}(0) \), then

\[
dT_{i-1,i}(0) = T_{i-1,i}(0) \hat{\Delta}_i, 
\]

\[
= \begin{bmatrix} 0 & -\delta z_i & \delta y_i & dx_i \\ -\delta z_i & 0 & -\delta x_i & dy_i \\ -\delta y_i & \delta x_i & 0 & dz_i \\ 0 & 0 & 0 & 0 \end{bmatrix}, 
\]

(23)

where \( dx_i, dy_i, \) and \( dz_i \) are infinitesimal displacements along the \( x-, y-, \) and \( z- \) axes of link-frame \( i \), respectively, and \( \delta x_i, \delta y_i, \) and \( \delta z_i \) are infinitesimal rotations about \( x-, y-, \) and \( z- \) axes of link-frame \( i \), respectively.

4.3.2. Gross-Error Model of a Robot

Similar to the error model of a dyad, the gross-geometric error, \( dT_{0n} \), between the actual end-effector position the nominal position, can be described as

\[
dT_{0n} = \hat{\Delta}_{0n} T_{0n}. 
\]

(24)

and

\[
\hat{\Delta}_{0n} = dT_{0n} T_{0n}^{-1} 
\]

\[
= \begin{bmatrix} 0 & -\delta z_{0n} & \delta y_{0n} & dx_{0n} \\ -\delta z_{0n} & 0 & -\delta x_{0n} & dy_{0n} \\ -\delta y_{0n} & \delta x_{0n} & 0 & dz_{0n} \\ 0 & 0 & 0 & 0 \end{bmatrix}, 
\]

(25)

(26)

where \( \delta x_{0n}, \delta y_{0n}, \) and \( \delta z_{0n} \) are the rotations about the axes of the base frame, and \( dx_{0n}, dy_{0n}, \) and \( dz_{0n} \) are the displacements along the axes of the base frame, respectively. Note that the gross error, \( dT_{0n} \), is expressed in the base frame. Equation (25) follows the left multiplicative differential transformation of \( T_{0n} \). The calibrated position of the end effector becomes

\[
T_{0n}'(q) = T_{0n} + dT_{0n}. 
\]

(27)

4.3.3. Linear Superposition

Based on these assumptions, the errors in the dyads will contribute to the gross error in the end-effector’s position, \( dT_{0n} \).
Since the geometric errors are all very small, the principle of linear superposition can be applied. We assume that the gross errors $dT_{0n}$ are the linear combination of the errors in the dyads $dT_{i-1,i}(0), (i = 1, 2, \ldots, n)$; then

$$dT_{0n} = \sum_{i=1}^{n} T_{0,i-1} dT_{i-1,i}(0) e^{i\theta x} T_{i,n}.$$  

Equation (28) converts and sums the initial position errors of the dyads in the base-frame coordinates. The forward kinematics of link-frame $j$ relative to link-frame $i$ ($i \leq j$) is represented by $T_{ij}$. Especially, $T_{ij} = I_{4\times4}$ when $i = j$. Substituting eq. (23) into eq. (28), and right multiplying $T_{0n}$,

$$dT_{0n} T_{0n}^{-1} = \Delta_{0n},$$  

$$= \sum_{i=1}^{n} T_{0,i-1} T_{i-1,i}(0) \Delta_i T_{i-1,i}^{-1} T_{0,i-1}^{-1}.$$  

From the first-order approximation, we have

$$\Delta_{0n} = dT_{0n} T_{0n}^{-1} \approx \log(T_{0n}' T_{0n}^{-1}).$$  

Converting eq. (31) into the adjoint representation, we have

$$\log^y(T_{0n}' T_{0n}^{-1}) = \sum_{i=1}^{n} AdT_{0,i-1}(AdT_{i-1,i}(0)\Delta_i).$$  

Equation (32) can also be expressed in the following matrix form:

$$y = Ax,$$  

where

$$y = \log^y(T_{0n}' T_{0n}^{-1}) \in R^{6\times1};$$

$$x = \text{column}[\Delta_1, \Delta_2, \ldots, \Delta_n] \in R^{6n\times1};$$

and

$$A = \text{row}[AdT_{0,1}(0), AdT_{0,1}(AdT_{1,2}(0)), \ldots, AdT_{0,n-1}(AdT_{n-1,n}(0))] \in R^{6\times6n}.$$  

In eq. (33), $x$ represents the error parameters to be identified in a modular robot assembly. The quantities in matrix $A$ and $T_{0n}$ are determined from the nominal model. $T_{0n}'$ comes from the actual measured data.

To improve the accuracy of the calibration model, the kinematic calibration procedure usually requires the position of the end effector to be measured in several different robot postures. For the $i$th measurement, we obtain $y_i$ and $A_i$. After taking $m$ measurements,

$$\tilde{Y} = \tilde{A}x,$$  

where

$$\tilde{Y} = \text{column}[y_1, y_2, \ldots, y_m] \in R^{6m\times1};$$

and

$$\tilde{A} = \text{column}[A_1, A_2, \ldots, A_m] \in R^{6m\times6n}.$$  

The least-squares solution for $x$ can be obtained by

$$x = \tilde{A}^T \tilde{Y},$$  

where $\tilde{A}^T$ is the pseudoinverse of $\tilde{A}$ and $\tilde{A}^T = (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T$ for $m > n$; $\tilde{A}^T = \tilde{A}^T (\tilde{A} \tilde{A}^T)^{-1}$ for $m < n$; and $\tilde{A}^T = \tilde{A}^{-1}$ for $m = n$. The calibration procedure is illustrated in the diagram of Figure 9. Computer simulations and actual experiments on the modular robot systems described by Chen and Yang (1997) have shown that this calibration method can improve the accuracy in end-effector positioning by up to two orders of magnitude.
4.4. Dynamics

The dynamic model of a robot can be formulated with an iterative method through a recursive Newton-Euler equation. This method can be generally applied to branch-type robots without modification. Here we present a method for generating the closed-form dynamic models of modular robots using the AIM and the recursive algorithm.

4.4.1. The Newton-Euler Equation for Link Assembly

Assume that the mass center of link-assembly \( j \) is coincident with the origin of the link-module frame \( j \). The Newton-Euler equation of this rigid-link assembly with respect to frame \( j \) is (Murray, Li, and Sastry 1994)

\[
\mathbf{F}_j = \begin{bmatrix} f_j \\ \tau_j \end{bmatrix} = \begin{bmatrix} m_j I & 0 \\ 0 & J_j \end{bmatrix} \begin{bmatrix} \dot{v}_j \\ \dot{w}_j \end{bmatrix} + \begin{bmatrix} w_j \times m_j v_j \\ w_j \times J_j w_j \end{bmatrix}, \tag{36}
\]

where \( \mathbf{F}_j \in \mathbb{R}^{6 \times 1} \) is the resultant wrench applied to the center of mass relative to frame \( j \). The total mass of link-assembly \( j \) is \( m_j \) (which is equal to the sum of link \( v_j \) and joint \( e_j \)). The inertia tensor of the link assembly about frame \( j \) is \( J_j \).

Transforming eq. (36) into the adjoint representation, we have

\[
\mathbf{F}_j = M_j \dot{V}_j - a_d^T v_j (M_j V_j). \tag{37}
\]

The following notations are adopted:

- \( M_j = \begin{bmatrix} m_j & 0 \\ 0 & J_j \end{bmatrix} \in \mathbb{R}^{6 \times 6} \) is the generalized mass matrix;
- \( V_j = \begin{bmatrix} v_j \\ w_j \end{bmatrix} \in \mathbb{R}^{6 \times 1} \) is the generalized body velocity, where \( v_j \) and \( w_j \) are \( 3 \times 1 \) vectors defining body translational velocity, \( v_j = (v_x, v_y, v_z)^T \), and the angular velocity, \( w_j = (w_x, w_y, w_z)^T \), respectively;
- \( a_d^T v_j \in \mathbb{R}^{6 \times 6} \) is the transpose of adjoint matrix \( a_d v_j \) related to \( V_j \):

\[
a_d^T v_j = (a_d v_j)^T = \begin{bmatrix} \dot{w}_j & \dot{v}_j \\ 0 & \dot{w}_j \end{bmatrix} = \begin{bmatrix} -\dot{w}_j & 0 \\ -\dot{v}_j & -\dot{w}_j \end{bmatrix}; \tag{38}
\]

- \( \dot{v}_j \) and \( \dot{w}_j \in \mathbb{R}^{3 \times 3} \) are skew-symmetric matrices related to \( v_j \) and \( w_j \), respectively; and
- \( \dot{V}_j = \begin{bmatrix} \dot{v}_j \\ \dot{w}_j \end{bmatrix} \in \mathbb{R}^{6 \times 1} \) is the generalized body acceleration.

4.4.2. The Recursive Newton-Euler Algorithm

The recursive algorithm is a two-step iteration process. For a branch-type robot, the generalized velocity and acceleration of each link are propagated from the base to the tips of all branches. The generalized force of each link is propagated backward from the tips of the branches to the base. At the branching module, generalized forces transmitted back from all branches are summed.

**Forward Iteration**

The generalized velocity and acceleration of the base link are initially given:

\[
V_b = V_0 = (0, 0, 0, 0, 0, 0)^T, \tag{39}
\]
\[
\dot{V}_b = \dot{V}_0 = (0, 0, g, 0, 0, 0)^T, \tag{40}
\]

where \( V_b \) and \( \dot{V}_b \) are expressed in the base frame 0. We assume that the base frame coincides with the spatial reference frame. The generalized acceleration (eq. (40)) is initialized with the gravitation acceleration \( g \) to compensate for the effect of gravity. Referring to Figure 6, the recursive-body velocity and acceleration equations can be written as

\[
V_j = Ad_{T_{ij}}^{-1}(V_i) + s_j \ddot{q}_j, \tag{41}
\]
\[
\dot{V}_j = Ad_{T_{ij}}^{-1}(\dot{V}_i) + adAd_{T_{ij}}^{-1}(s_j \ddot{q}_j) + s_j \dddot{q}_j, \tag{42}
\]

where all the quantities, if not specified, are expressed in link-frame \( j \):

- \( V_j \) and \( \dot{V}_j \) are the generalized velocity and acceleration of link-assembly \( j \);
- \( \dot{q}_j \) and \( \ddot{q}_j \) are the velocity and acceleration of joint \( e_j \), respectively;
- \( Ad_{T_{ij}}^{-1} \) is the adjoint representation of \( T_{ij}^{-1}(q_j) \), where \( T_{ij}(q_j) \in SE(3) \) is the position of frame \( j \) relative to frame \( i \) with joint angle \( q_j \), and \( Ad_{T_{ij}}^{-1} = (Ad_{T_{ij}})^{-1} \); and
- \( s_j \in \mathbb{R}^{6 \times 1} \) are the twist coordinates of joint \( e_j \).

**Backward Iteration**

The backward iteration of the branch-type robot starts simultaneously from all the pendant-link assemblies. Let \( \mathcal{V}_{PD} \subseteq \mathcal{V} \) be set of the pendant links of the branch-type robot. For every pendant-link assembly \( d_i (\mathbf{v}_{d_i} \in \mathcal{V}_{PD}) \), the Newton-Euler equation (eq. (37)) can be written as

\[
F_{di} = -F_{di}^e + M_{di} \dot{V}_{di} - a_d^T v_{di} (M_{di} V_{di}), \tag{43}
\]

where \( F_{di} \) is the wrench exerted on link-assembly \( \mathbf{v}_{d_i} \) by its parent (preceding) link relative to frame \( d_i \); and \( F_{di}^e \) is the
external wrench exerted on \( \mathbf{v}_d \). Note the total wrench, \( \mathbf{F}_d = F_{d1} + F_{d2} \).

Now traverse the links in the robot backward from the pendant links. Let \( \mathcal{V}_H \) be the set of successors of link \( \mathbf{v} \). For every link-assembly \( i \), the Newton-Euler equation (eq. (37)) can be written in the following form:

\[
F_i = \sum_{j \in \mathcal{V}_H} A_{ij}^T (F_j) - F_i^e + M_i \dot{\mathbf{V}}_i - a_{ij}^D (M_i \mathbf{V}_i),
\]

where all quantities, if not specified, are expressed in link-frame \( i \); \( F_i \in R^{6\times1} \) is the wrench exerted to link-assembly \( i \) by its predecessor; \( F_j \in R^{6\times1} \) is the wrench exerted by link-assembly \( i \) to the successor \( v_j \in \mathcal{V}_H \) expressed in link-frame \( j \); and \( F_i^e \) is the external wrench applied to link-assembly \( i \). Basically, the total wrench is \( \mathbf{F}_i = F_i - \sum_{j \in \mathcal{V}_H} A_{ij}^T (F_j) + F_i^e \).

The applied torque/force to link-assembly \( i \) by the actuator at its input joint \( \mathbf{e}_i \) can be calculated by

\[
\tau_i = \mathbf{s}_i^T \mathbf{F}_i.
\]

### 4.4.3. Closed-Form Equations of Motion

By iteratively expanding the recursive Newton-Euler equations (eqs. (39)–(44)) in the body coordinates, we obtain the generalized velocity, generalized acceleration, and generalized force equations in matrix form:

\[
\begin{align*}
\mathbf{V} &= \mathbf{G} \mathbf{S} \dot{\mathbf{q}}, \\
\dot{\mathbf{V}} &= \mathbf{G} \mathbf{T}_0 \mathbf{\dot{V}}_0 + \mathbf{G} \mathbf{S} \ddot{\mathbf{q}} + \mathbf{G} \mathbf{A}_1 \mathbf{V}, \\
\mathbf{F} &= \mathbf{G}^T \mathbf{F}^e + \mathbf{G}^T \mathbf{M} \dot{\mathbf{V}} + \mathbf{G}^T \mathbf{A}_2 \mathbf{M} \mathbf{V}, \\
\tau &= \mathbf{S}^T \mathbf{F},
\end{align*}
\]

where \( \mathbf{V} = \text{column}[\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_n] \in R^{6n\times1} \) is the generalized body-velocity vector; \( \dot{\mathbf{V}} = \text{column}[\dot{\mathbf{V}}_1, \dot{\mathbf{V}}_2, \ldots, \dot{\mathbf{V}}_n] \in R^{6n\times1} \) is the generalized body-acceleration vector; \( \mathbf{F} = \text{column}[\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n] \in R^{6n\times1} \) is the body-wrench vector; \( \tau = \text{column}[\tau_1, \tau_2, \ldots, \tau_n] \in R^{n\times1} \) is the applied joint-torque/force vector; \( \dot{\mathbf{q}} = \text{column}[\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n] \in R^{n\times1} \) is the joint-velocity vector; \( \ddot{\mathbf{q}} = \text{column}[\ddot{q}_1, \ddot{q}_2, \ldots, \ddot{q}_n] \in R^{n\times1} \) is the joint-acceleration vector; \( \mathbf{V}_0 = (0, 0, 0, 0)^T \in R^{6\times1} \) is the generalized acceleration of the base link; \( \mathbf{S} = \text{diag}[s_1, s_2, \ldots, s_n] \in R^{6n\times n} \) is the joint-twist matrix written in the respective body coordinates; \( \mathbf{M} = \text{diag}[M_1, M_2, \ldots, M_n] \in R^{6n\times6n} \) is the total generalized-mass matrix;

\[
\begin{align*}
\mathbf{A}_1 &= \text{diag}[-a_{d1} q_1, -a_{d2} q_2, \ldots, -a_{dn} q_n] \in R^{6n\times6n}, \\
\mathbf{A}_2 &= \text{diag}[-a_{q1} q_1, -a_{q2} q_2, \ldots, -a_{qn} q_n] \in R^{6n\times6n}, \\
\mathbf{F}^e &= \text{column}[F_{e1}^e, F_{e2}^e, \ldots, F_{en}^e] \in R^{6n\times1} \text{ is the external wrench vector;}
\end{align*}
\]

The matrix \( \mathbf{G} \) is called a transmission matrix. Substituting eqs. (46)–(48) into eq. (49), we obtain the closed-form equation of motion for a branch-type modular robot with \( n + 1 \) modules (including the base module):

\[
\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{N}(\mathbf{q}) = \tau,
\]

where

\[
\begin{align*}
\mathbf{M}(\mathbf{q}) &= \mathbf{S}^T \mathbf{G}^T \mathbf{M} \mathbf{G} \mathbf{S}, \\
\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \mathbf{S}^T \mathbf{G}^T (\mathbf{M} \mathbf{A}_1 + \mathbf{A}_2 \mathbf{G} \mathbf{S}) \mathbf{S}, \\
\mathbf{N}(\mathbf{q}) &= \mathbf{S}^T \mathbf{G}^T \mathbf{M} \mathbf{G} \mathbf{T}_0 \dot{\mathbf{V}}_0 + \mathbf{S}^T \mathbf{G}^T \mathbf{F}^e.
\end{align*}
\]

The mass matrix is \( \mathbf{M}(\mathbf{q}) \); \( \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} \) represents the centrifugal and Coriolis forces; \( \mathbf{N}(\mathbf{q}) \) represents the gravitational and external forces. The procedure for obtaining the closed-form equation (eq. (50)) is summarized in the diagram in Figure 10. It has been successfully implemented in Mathematica code.

### 5. Configuration Optimization

Introducing modularity in a robotic system implies that the system performance can be optimized through proper selection and reconfiguration of module components. The task planner for the modular robotic work cell will be able to determine the optimal robot configuration and geometry for a given task from an inventory of robot modules. Figure 11 depicts the general approach for determining the optimal assembly configuration. Shaded blocks represent the basic formulation of the optimization problem. With a given set of modules selected from the component database, all possible and unique assembly configurations can be generated and identified through an enumeration algorithm (Chen and
Fig. 10. Dynamics model generation.

Burdick 1993). In the second step, an objective function is formulated to evaluate the performance of every assembly configuration, based on the task specifications. A basic robot task contains task specifications that are provided by the task planner—the goal positions/orientations, force application, accuracy, and dexterity of the end effectors—and constraints to be overcome—obstacle avoidance, workspace limit, singularity, and kinematic redundancy (Chen and Burdick 1995). A search/optimization procedure is employed in the last step to find the optimal assembly configuration.

Note that all the dimensions of the modules have been previously designed and fixed at the selection stage. With a given set of modules, the possible combinations of robot-assembly configurations is always a finite number. Therefore, the parameter space for the optimization is discrete, and combinatorial optimization methods can be applied. Exhaustive search algorithms can be used to find the exact optimal solution, but the exponential growth of the data set impedes the efficient implementation of such an algorithm. Random-search techniques such as genetic algorithms (GA) (Chen 1994) and simulated annealing (SA) (Paredis and Khosla 1995) are more suitable for such problems. Transition rules for data points required in GA and SA can be easily implemented, based on a data-representation scheme such as AIM.

5.1. The Task-Oriented Objective Function

The crucial task in determining the optimal robot configuration is formulating an objective function that will assign a “goodness” value to every assembly configuration that accomplishes a specified task. The form of the objective function should be general enough so that it is applicable to a wide variety of task requirements. Two components of a robot task—task specifications and constraints—must be considered in formulating the objective function. We call this function an assembly configuration evaluation function (ACEF). The assembly configuration with the greatest ACEF value is deemed optimal. It is also important to note that from a given set of modules, it is possible to construct robots with various topologies, such as serial or parallel kinematic structures. Even with a fixed robot-topology class, the number of degrees of freedom can alter the kinematic functionality of the system. Here we propose a solution strategy for a modular robot with a fixed topology and a fixed number of DOF.

The structure of the ACEF for a serial modular robot is shown in Figure 12. The input is an AIM with a predefined number of DOFs and a predefined topology. The output is the “goodness” of the AIM in terms of a nonnegative real number. An AIM with a large ACEF value represents a good assembly configuration. The ACEF consists of two parts: task and structure evaluations. Task evaluation is performed according to the given task specifications: the task points (or the positions of the end effector) and a designated criteria measure, such as the dexterity or manipulability. A workspace check on the task points is executed before computing the measures for filtering out inaccessible points. Structure evaluation assesses the kinematic constraints (joint singularity and redundancy, link interference) and environmental constraints (workspace obstacles) imposed on the robot in accomplishing the assigned task. The proposed ACEF assumes that the modular robot is operated in a structured environment, and that there are no obstacles in the workspace. An auxiliary function, termed the module-assembly preference (MAP) is defined on the AIM to exclude undesirable kinematic features. Detailed implementation of the task and structure evaluations can be obtained from Chen (1996).

5.2. Evolutionary Algorithms

An evolutionary algorithm is a probabilistic search/optimization method based on the principle of evolution and hereditary of nature systems (Michalewicz 1994). In this algorithm, a population of individuals for each generation is maintained. The individual is implemented with some data structure and
Fig. 11. Determination of a task-optimal configuration.

Fig. 12. The ACEF for serial modular robots.
is evaluated by a “fitness function” to give a measure of its fitness. A new population is formed by selecting the more suitable individuals. Members in the new population undergo transformations by the “genetic operators” to form new solutions. Through structured random information changes, the new generation is more “fit” than the previous generation. After a number of iterations, the individuals will converge to an optimal or near-optimal solution. Here we attempt to use the AIMs as the data structures of the solution, and define AIM-related genetic operators (Chen 1996) as solving the task-optimal problem in an evolutionary approach, because the AIM is a natural representation of the modular robot and is topologically independent.

Figure 13 depicts the application of the evolutionary algorithm in solving the task-optimal configuration problem. An example of optimizing the configuration of a 4-DOF modular robot is provided in the following example. Suppose we wish to find a 4-DOF fixed-base serial robot with revolute joints that passes through two task points \( p_1 \) and \( p_2 \). Also suppose that we require there be no redundant joints, and minimum link interference. Let the performance measure of the robot be the manipulability. The initial set of AIMs randomly generated is shown in Figure 14. The population size is 8, and the evolution stopped after 30 generations. The assembly configuration in the target generation that has the highest fitness value is chosen as the optimal one (Fig. 15). The average and maximum fitness values for every generation are shown in Figure 16. As can be seen, the evolutionary algorithm does increase the fitness values, generation by generation. Although the best solution may not be guaranteed, a suboptimal solution can always be found, and in return, the efficiency of finding the solution is increased.

6. Summary

We have presented a general method for automating the model-generation process for modular reconfigurable robots based on a graph representation of robot geometry (called the assembly incidence matrix) and geometry-independent model-building algorithms for the kinematics, dynamics, and error models of a robot. Furthermore, the AIMs of the assembly configuration of modular robots facilitate the determination of the optimal robot configuration for a specific task, using combinatorial optimization techniques. We also presented an approach for solving the task-optimal problem using evolutionary algorithms with customized genetic operators, based on the AIM of the robot.

The application of this automatic modeling technique is in the design and implementation of the system kernel for a modular reconfigurable robot work cell. Currently, the proposed kernel architecture is being implemented on a modular robot control and simulation software application, termed SEMORS (Simulation Environment for MOdular Robot Sys-
tems). In this system, we do not maintain a library of robot models, since a robot’s possible assembly configurations is not a fixed number. Instead, only a small set of component database and kernel functions are kept in the robot’s controller, and required robot models are generated automatically.

Acknowledgments

The authors thank Mr. Edwin Ho of the School of Mechanical and Production Engineering, Nanyang Technological University, for providing the CAD drawing of the work cell. This research is supported in part by the Ministry of Education, Singapore under AcRF RG64-96 and by the Gintic Institute of Manufacturing Technology, Singapore, under Upstream Project U97-A006.

References


Deo, N. 1974. Graph Theory with Applications to Engineering and Computer Science. New York: Prentice Hall.


