9

CAD-Based Techniques in Task Planning and Programming of Robots in Computer-Integrated Manufacturing

Bijan Shirinzadeh
Monash University

The key to future robotic system utilization in flexible assembly systems (FASs) relies heavily on rapid task planning and programming of robots. Furthermore it has become evident that the complete integration of the design, analysis, and off-line verification of robotic operations in computer-integrated manufacturing (CIM) environment is required. These requirements place an important emphasis on techniques and systems for planning and programming of the assembly tasks within the framework of CIM systems.

This chapter presents techniques for integrated task planning and programming of robotic operations. Such techniques generally employ computer-aided design (CAD) platforms to fully exploit their database and modeling capabilities. Various approaches to off-line programming is presented. The techniques for modeling, task specification, and database structuring are described. Such systems require methods for robot-independent program generation and the subsequent simulation and verification of the application programs including calibration of fixtures and local frames and are also presented here.

© 2001 by CRC Press LLC

9.1 Introduction

Dynamic global competition has compelled many manufacturing industries to be more concerned with productivity, cost reduction, and flexibility due to the reduction of the product life cycles [1, 2]. Recent research efforts have focused on the development of CIM systems to increase productivity and resource management. Robots are chosen, in the manufacturing environment, primarily for their flexibility and rapid response to changeovers. However, in order to take advantage of this flexibility, the production engineer needs to rapidly plan the task and develop application programs for the robotic operations.

There exists a large number of "on-line" and "off-line" robot programming systems and languages [3, 4]. Most industrial robots are still programmed using traditional "on-line" programming systems, commonly referred to as "teach-mode-programming." In this approach, a programmer physically moves the robot by a teach-pendant through the desired locations [5]. The locations are then recorded and can be replayed when needed. "On-line" programming of robots is a time-consuming process and not very efficient in a flexible manufacturing environment. Furthermore, the "on-line" programming technique requires the use of the actual robot which is physically put through the desired sequence of actions [6, 7]. This method is mainly used for large batch manufacturing environments.

A more advanced approach allows task specification and programming a robot off-line using task planning and off-line programming techniques [5]. The task planning and off-line programming (OLP) systems have the potential to reduce the programming time [8]. These combine computer modeling and simulation to perform task specifications and generate programs off-line. Such systems do not require direct physical access to the robot or its environment during the planning and programming phase [9]. However, they do require reprogramming of critical locations or frames on the actual manufacturing fixtures. This chapter briefly describes the different approaches to programming systems. Such off-line programming systems provide the basic building blocks for more advanced task planning systems and their integration within the CIM environment. This chapter also presents techniques for integrated task planning and verification of robotic operations. The focus of attention will mainly be directed at robotic assembly operations.

9.2 Strategies for Off-Line Programming (OLP)

Text-Level Programming

The most basic level of off-line programming is the manipulator-level or *text-based programming*. At this level, the focus of attention is the manipulator and other associated devices, such as grippers. The task description consists of how the manipulator is to be moved through the required points [9]. This method provides better control over the robot and it is generally identical to the robot native language [10]. It also provides for a relatively cost effective and reliable programming facility. However, writing text-based programs in languages such as VAL II (Staubli) and ARLA (ABB) is not easy. The user must generally think in 3-D space without any visual aids. In addition, the user must have a background in programming, since most manipulator-level languages are usually extensions of languages such as PASCAL and BASIC.

Graphic-Level Programming

Graphic-level programming is a combination of teach-mode and manipulator-level programming techniques [9]. Graphic-level programming eliminates some of the difficulties in 3-D visualization during off-line programming. In graphic-level programming, the user programs the positions to which the robot is required to move by assigning frames on objects and storing their locations [5]. This level of programming is off-line but robot-dependent, requiring the robot model to be a resident in the system [10]. Most commercial systems use this technique and examples of such systems include IGRIP (Deneb), CimStation (SILMA), and ROBCAD (Technomatix). This technique is sometimes referred to as teach-mode programming on screen [11].

© 2001 by CRC Press LLC

Object-Level Programming

In an object-level program, the user decides the order of assembly, approach, and departure vectors, and the user's description of the task is in terms of the objects that have to be handled [9]. The user must provide a set of commands that describes spatial relationships among objects that have to be handled. For example, a situation where a plate is placed on a block with two holes aligned may be described by:

AGAINST/	top of block, bottom of plate;
COPLANAR/	side of block, side of plate;
ALIGNED/	holeA of block, holeB of plate;

This level of programming is off-line and robot-independent; however, translators are still required to generate manipulator-specific programs. Object-level programming systems, such as RAPT, require the user to provide detailed definition of various features; such as top of block, side of plate, holeA, etc. This specification of features on objects must generally be carried out in conjunction with an advanced modeling system. In addition, this level of programming requires special care in the development of spatial relationships and thorough testing of the program [12]. This is mainly due to the fact that the possibility of ambiguous situations, and thus errors in the program, is extremely high using this approach. Research in this area focuses attention on improving the feature specification and integrated modeling capabilities.

Task-Level Programming

A task-level programming system generates motion and strategy plans based on the specified task or the final goal [9]. A task-level programming, sometimes referred to as objective-level programming, requires a very complete model of the robot and its work cell environment. The system must also generate all possible steps for task planning, grasp planning, path planning, trajectory planning, and post-processing [13]. The system must also be able to deal with uncertainties and sensor actions. The user is not required to define spatial relationships between objects, only high-level task commands. In the proposed objective-level programming language, the user specifies the task the robot must achieve.

The user must also provide information such as the parts to be used, their initial layout, and the final assembly layout. The system is responsible for planning how the assembly is to be done; determining the approach and departure vectors; how and where to grasp the parts; how to use sensors; and how to maneuver parts around obstacles. The final result must be simulated on the graphics screen in order to assist with the final decision (i.e., the engineer must have the final say). A fully functional task-level planning and programming system does not yet exist, although it must be emphasized that attention has been focused on research and development of the modules or sub-systems for such task-level programming systems [14]. Therefore, experimental systems with some of the task-level functionalities are being developed by researchers.

9.3 Strategies for Task Planning and Programming

An idealized task planning and programming system consists of a number of sub-systems or sub-modules. Fig. 9.1 shows a schematic diagram of important sub-modules which have received attention within the framework of off-line task planning and programming systems. These include geometric modeling, task specification, grasp planning, path planning, simulation, and calibration. Individual sub-modules will be described within this section.

Geometric Modeling

Task programming systems require a very complete model of the assembly environment to allow computer-aided planning operations to be performed. Therefore, a geometric modeling platform is an essential component of such planning systems (i.e. planners). In fact, a versatile computer-aided task planning and programming system is generally built on an accurate and user-friendly modeling platform [15].

© 2001 by CRC Press LLC

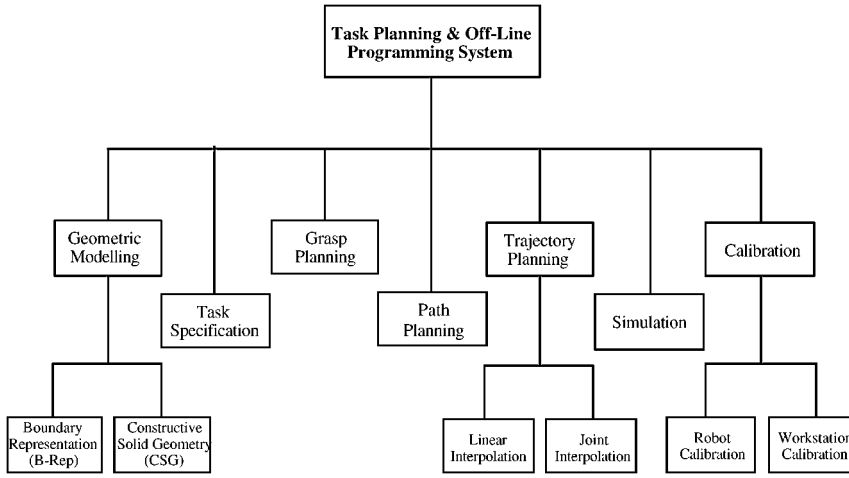

FIGURE 9.1 The hierarchical structure for task planning and off-line programming.

Geometric models must be developed before any task actions can be expanded into sequences of robot operations [16].

Objects in the workspace have to be modeled using solid models rather than wire frames. This approach will assist in retrieving the 3-D information about the objects from the database because this information is required during computer-assisted planning operations. The planning operations may include task specification, grasp planning, path planning, and interference detection of robotic tasks and workpieces [17]. For example, the interference detection may be carried out by testing the intersection of the robot solid model and the union of all other objects. This technique can also be employed for grasp planning which will be described later. Two categories for modeling include constructive solid geometry (CSG) and boundary representation (B-Rep).

Constructive Solid Geometry

Constructive solid geometry (CSG) constructs solid objects from a set of solid geometric shapes [18]. This method uses motional and combinatorial operators to combine a few primitive shapes such as cube, wedge, cone, cylinder, etc. A CSG scheme can be specified by a context-free grammar, as listed below:

```
<mechanical part>     ::= <object>
<object>              ::= <primitive> |
                          <object> <motion op> <motion
                        arguments>|
                          <object> <set operator> <object>
<primitive>           ::= cube | wedge | cone | cylinder | ...
<motion op>           ::= rotate | translate | scale | ...
<set operators>       ::= union | intersection | difference |
                        complement | ...
```

Each primitive of a solid object is described by its volumetric parameters such as length, width, height, etc., and its position relative to a reference frame. These primitives are stored in leaf nodes, as an ordered binary tree and the operators are stored in the non-terminal nodes. During CSG's implementation, the depth of the tree should be monitored very carefully because it can lead to inefficient data retrieval when the tree is too deep. This problem can be overcome by representing each subassembly as a tree in itself and all the subassemblies are combined into the final tree. This method can generally be used for task planning because the final goal can be described as a tree of subassemblies.

© 2001 by CRC Press LLC

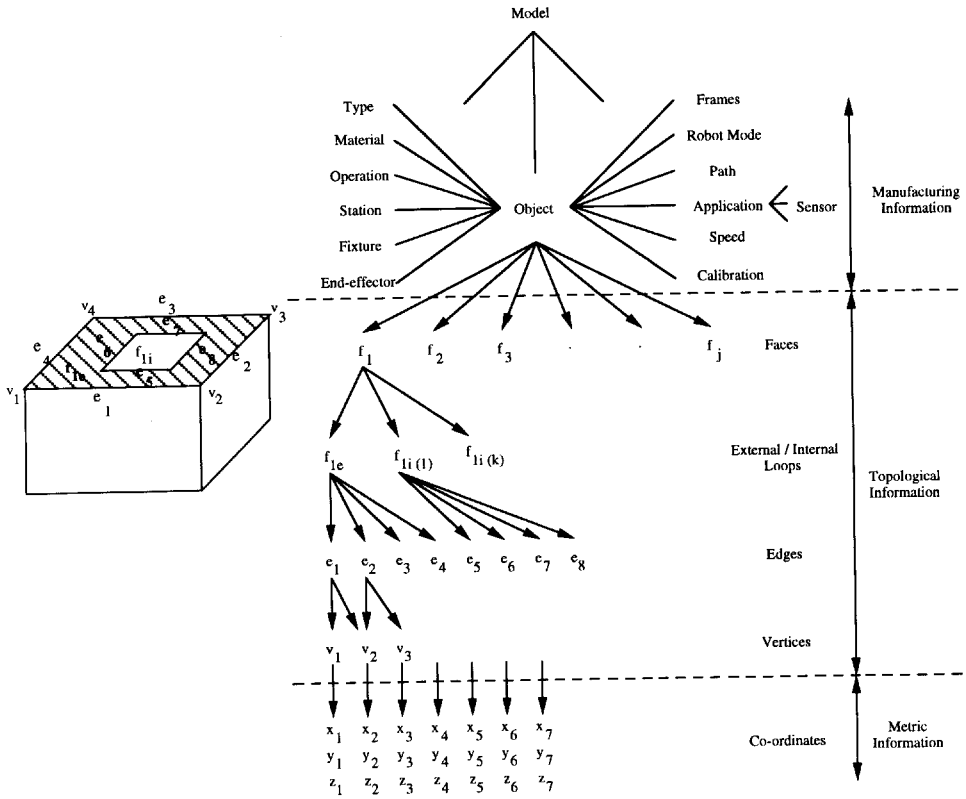

FIGURE 9.2 Structure of boundary representation (B-Rep) for a cuboid.

Boundary Representation (B-Rep)

Boundary representation (B-Rep) technique segments a solid object into its non-overlapping faces or patches, and models these according to their bounding edges and vertices [19]. The resulting data structure constructs a directed graph containing object, face, edge, and vertex nodes. The geometric representation of the object may be described as follows in Fig. 9.2:

- Each face is described by it bounding edges
- Each edge is described by its vertices
- Each vertex is stored as Cartesian coordinates

The external loop of a face (f_{1e}) follows a right hand rule where a face is transversed in counter-clockwise direction. An internal hole in a part may also be considered in B-Rep for possible fixture contact point. The boundary loop (f_{1i}), a hole in a face, is transversed in clockwise direction which is opposite to the external loop [19]. This method produces more efficient sources of geometric data for producing line drawings, graphic interaction (e.g., accessing a face, edge, or vertex), task specification, and graphic simulation as compared to CSG.

Therefore, a model may contain several objects in a scene (i.e., an assembly). Each object is referenced by the topological information and metric information. In addition, strategies are being developed to include information about manufacturing characteristics as a list added to the object characteristics [20]. These may include information such as fixture identification number; manufacturing cell type (e.g., automated, manual); material and manufacturing operation to be performed together with machine type (e.g., robot, milling, etc.); and other programmable aspects (e.g., static frames, dynamic frames, sensor application, calibration, etc.). A schematic diagram of the three types of information list is shown in Fig. 9.2.

© 2001 by CRC Press LLC

Task Specification

Task specification details the task-and-assembly sequence operation for robot programming. The final goal is decomposed into a set of subgoals and is governed by the process sequence, feasibility, and geometric constraints [21]. These constraints imply precedence relationships that will guarantee the correct order for the execution of the operation (i.e., drill object A first before inserting object B into A). Feasibility constraints verifies that all the objects and the extra features, such as "hole in the object," are available or exist. Geometric constraints can be determined by analyzing the geometry of the operation. These constraints are more concerned with the collision problems between a robot and the objects and also between the objects being handled with other objects. Task specification must also work very closely with path planning to ensure a collision-free path.

Many strategies for obtaining the sequence of operations have been developed. These methods are generally based on the three constraints mentioned above. Laperriere et al [22] have developed strategies by using a 'Relation Diagram' method. This method is based on geometric and dimensional information. Rocha and Ramos [21] have developed a strategy called TPMS, where the sequence is derived by using the symbolic plan operation method. Their method accounts for pre-existing manufacturing systems and the robot programming platforms.

Once the sequence has been determined, a computer program which executes that sequence must be generated and stored in the robot controller's memory or operation data files. Each step in the assembly sequence has to be translated into robot executable motions and operations. Every motion and operation are the elements that can be used to change the state of the assembly setup or the environment (i.e., approach object A, close gripper, and screw object A to object B).

The choice of an assembly sequence may depend on additional facts such as time, cost, stability, or ease of assembly. In the past decade, research efforts have focused on the establishment of strategies for automated assembly sequence generation. Such techniques make use of rule-based decision-making approaches to generate precedence for assembly sequence [23]. It must be noted that a complete survey of such research efforts is out of the scope of this chapter. There has also been an increase in research efforts to develop strategies for multiple manipulators [24]. Such studies have developed approaches that regard each manipulator and machine as an agent. The strategy analyzes the behavior and the geometric restriction to sequence of operations for each manipulator and machine.

Grasp Planning

Grasp planning is an important issue that must be considered in order to achieve a truly flexible task planning and off-line programming system. Once the task has been specified, a grasp planner provides all feasible grips of the assembly part and provides an optimal grip [25]. Detailed information, such as description of the part to be grasped, feeder, and assembly task are required in the planning. The planner determines the nonfree regions (Fig. 9.3), based on the gripper size, contact with feeder or assembly, and forbidden part faces such as threaded or fragile faces [26]. These regions are defined as "the regions on a part where the fingertip cannot be placed." The free regions are then used to determine the finger domain, which is an area on a part where a fingertip can be placed.

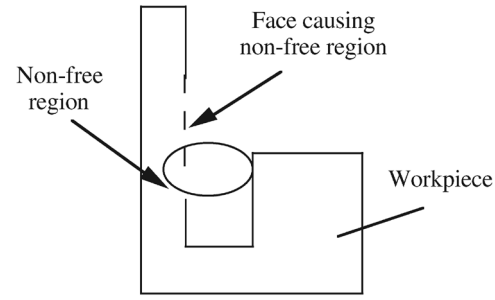

FIGURE 9.3 Non-free region caused by a face on the workpiece.

© 2001 by CRC Press LLC

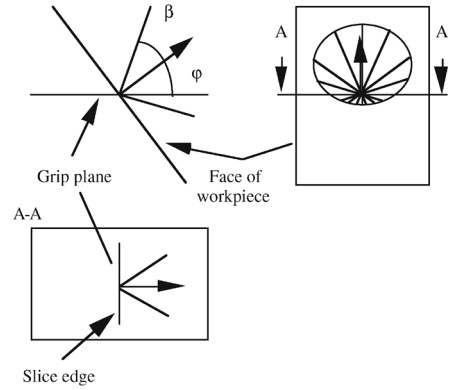

FIGURE 9.4 Schematic illustration of angle β_{gp} of a friction cone in the grip plane.

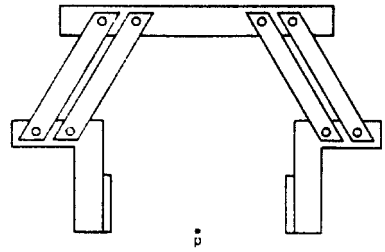

FIGURE 9.5 Schematic diagram of a parallel-jaw gripper.

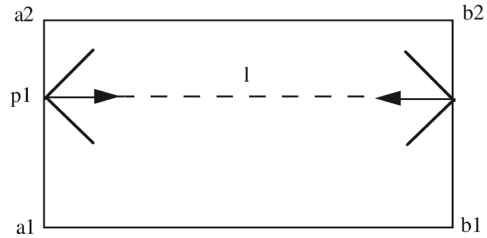

FIGURE 9.6 Grasp domain for a 2-fingered gripper.

A grip plane is a plane where the finger contact points are located. This plane is normally situated on the perpendicular side of the insertion direction, as it will reduce the degrees of freedom [27]. In the finger domain, edges are excluded if grid plane is outside the friction cone belonging to the edge's face, as shown in Fig. 9.4. The calculation of the angle of the friction cone in the grip plane may be performed using the following:

$$\beta_{gp} = \text{atan}((\tan^2\beta - \tan^2\varphi)/(1 + \tan^2\varphi))^{1/2} \tag{9.1}$$

To grasp an object without slipping, it is normally grasped at the centre of mass. In this way, torque on the fingertip surfaces is minimized [27]. However, different types of grippers will have their own effective method of grasping the objects to be manipulated. A parallel-jaw gripper is one of the most commonly used grippers (shown in Fig. 9.5). It uses a parallelogram mechanism and the fingertips extend and retract slightly in order to remain parallel. When the two edges are finally resolved, two grip points can be assigned to the edges by using following equation:

$$P2 = b1 + ((b2 - b1) \cdot (|p1 - a1|)/(|a2 - a1|)) \tag{9.2}$$

These grip points are normally placed on the two parallel surfaces that are not obstructed by other features of the part. The grasping force must be within the friction cones of the grip points as shown in Fig. 9.6 [28]. Thus, an object is normally grasped on the center of mass to minimize the torque on the

© 2001 by CRC Press LLC

fingertip surfaces. The length l is checked against the maximum gripper opening. If the length l is longer than the maximum gripper opening, then the grip points are not feasible and have to be changed. The optimal grasp sites may be analyzed and rated based on the stability, grasp time, and potential mating trajectories [25].

Path Planning

Path planning is an important phase within the task planning process. Research efforts have been mainly focused on establishment of strategies to automatically search all feasible paths and then find the optimal path. The planner automatically excludes those paths that result in collisions between the robot and objects in the work cell, and between objects that are being handled with other objects [29, 30]. There are several advanced strategies for path planning [31]; however, the general path planning can be categorized into three broad techniques: *hypothesize-and-test, penalty function,* and *free space.*

Hypothesize-and-test algorithm hypothesizes a candidate path from start to finish and tests the immediate path for possible collisions. Collision can be detected by examining the intersection between the solid objects. If any collisions are detected, then collision avoidance planning should be executed. This avoidance planner analyzes the obstacles involved in the collision and stores them in the interference data file so that the system is able to avoid the same case. The planner then defines a new immediate configuration by moving the colliding objects. It examines the new path again for possible collision and this is executed recursively until a collision-free path is found.

Penalty function encodes the presence of objects for each manipulator configuration. If there are any collisions, the planner yields an infinite value; however, the value will drop off sharply as the manipulator moves away from the obstacles. Overlaying the workspace with a 3-D grid and calculating the function at each grid point can reduce the computational time.

Free-space finds a path within the known subsets of free space. This method will miss a path if it has not been mapped. The free space is represented by regular geometric shapes with links. The planner only deals with free space rather than the space occupied by obstacles. It detects the free space of the object against the width of the robot plus clearance. If the path is determined to be too narrow, then the path will be eliminated. This procedure is repeatedly executed until an optimum path is found. It must be mentioned that there are also application areas where a desired path is required to be found to follow complex contoured surfaces [32]. Examples of these application areas include deburring, painting, and welding [33–36].

Trajectory Planning

After an appropriate path has been found (i.e., path planning), the task planner has to convert this path description into a time sequence of manipulator configurations. Trajectory planning deals with this problem. There are two types of constraints. These include task related and robot related constraints, shown in Fig. 9.7.

Task constraint is related to the task itself and the time taken for that task to be executed. For instance, a typical task constraint is to maintain the orientation of the gripper unchanged relative to a datum or world coordinate frame during the execution. On the other hand, the time constraint focuses attention

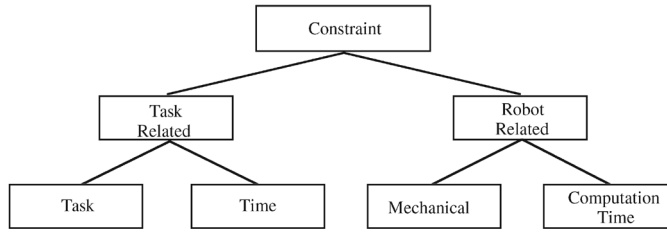

FIGURE 9.7 Constraints for trajectory planning.

© 2001 by CRC Press LLC

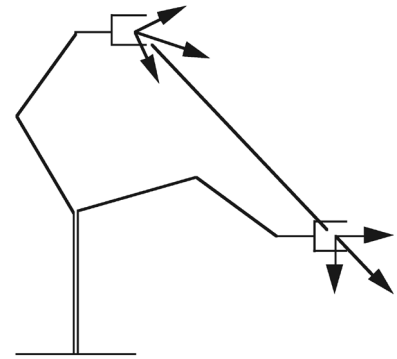

FIGURE 9.8 Example of a robot path undergoing linear interpolation.

on limiting the time period for the action to be performed and also it tries to maximize the equipment use. The robot-related constraint deals with the ability of the system to smooth the chosen path. There are two types of general path-following strategy available: linear interpolation motion and joint interpolated motion. These strategies will be explained, in detail, in the following subsections.

Linear Interpolation Motion

Linear interpolation motion causes the end-effector of the robot to follow the straight line segment drawn between two adjacent points [5]. It continually calculates the inverse transformation in order to obtain joint controller set points. Figure 9.8 illustrates the straight line motion of a revolute robot. The end-effector coordinate frame rotates about an axis k, fixed on the end-effector coordinate frame by an angle θ during the travel of the gripper from the initial to the goal position [37]. The formulation of this method is given below and it is based on the intermediate configuration:

$$P(j) = (P_{\text{goal}} - P_{\text{initial}}) \cdot (j/n) + P_{\text{initial}} \tag{9.3}$$

$$R(j) = R_{\text{initial}} \cdot Rot(k, \theta \cdot (j/n)) \tag{9.4}$$

$$T(j) = P(j) \cdot R(j) \tag{9.5}$$

where n represents the number of intermediate configurations plus one, P is the matrix representing the position of the manipulator with respect to the world coordinate frame, R is the matrix representing the rotation of the manipulator with respect to the world coordinate frame, and, T is the matrix describing the transformation of the end-effector coordinate frame with respect to the world coordinate frame.

There are some drawbacks to straight line motion in terms of real time control and this is due to the continual computation of inverse kinematics. Hence, velocity and acceleration controls are needed in order to perform a smooth end-effector movement. However, a predictable trajectory path can be obtained and any collisions during the motion can also be detected.

Joint Interpolation Motion

Joint interpolation strategy calculates the change in the joint variables required to achieve the final location. Figure 9.9 illustrates a joint interpolated motion of a revolute robot. This strategy controls the joint motors so that all the joints reach their final values at the same time [38]. Leu and Mahajan [39] suggested that the joint variables are linear with respect to the path length and the motion should follow the formulation given below:

$$q(j) = [q_{\text{goal}} - q_{\text{initial}}] \cdot (j/n) \tag{9.6}$$

where j represents the identification number for (i.e., j-th) intermediate configuration increment; q defines the vector representing joint variables; and n is the number of intermediate configurations plus one. It must be emphasized that in the joint interpolation, the end-effector does not produce a predictable

© 2001 by CRC Press LLC

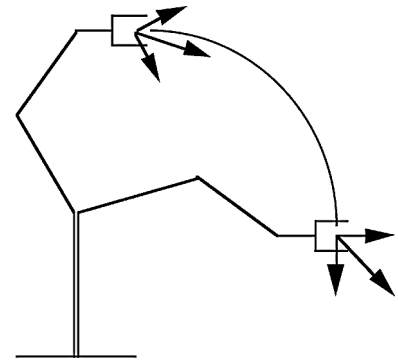

FIGURE 9.9 Example of a robot path undergoing linear interpolation.

trajectory path. This is due to the fact that the joint angles are interpolated between via points; therefore, unexpected collisions between obstacles may occur during the movement.

Calibration

Off-line programming may produce inaccurate robot programs in terms of positioning and orientation [40]. This is due to the differences between the computer models of the manipulator and objects in the working space to that of the actual ones. Offsets of the zero positions of the robot, differences in the link lengths, and misalignment of the joints are some sources of errors [2, 41]. An idealized off-line programming system should take into account mechanical deficiencies such as:

- Inaccuracies in the arm element dimensions
- Internal play of joints
- Internal non-linearities of gearing
- Deflection of arm elements and servo-positioning errors

The gap between model and reality can be narrowed by calibrating the robot and obtaining the actual working paths [2]. This can be done by reteaching the robot manually for various working points (e.g., corner of a feeder). A relationship has to be established between the working coordinate system and the measuring coordinate system. A mathematical model for this relationship can be derived based on the nominal coordinate system ($X_{R,i}$), actual coordinate system ($X_{M,i}$), scale (μ), and rotational matrix (D) about x, y, and z axes.

$$X_{R,i} = T + \nu DX_{M,I} \tag{9.7}$$

$$D = D(x, \alpha) \cdot D(y, \beta) \cdot D(z, \gamma) \tag{9.8}$$

where i represents the identification number of position (e.g., i-th position), $X_{R,i}$ represents resultant position vector, $X_{M,i}$ defines the actual measured position vector, and T defines the desired position vector. In general, robot calibration can be classified into two types, static and dynamic [2, 41]. Static calibration identifies those parameters which primarily influence the static positioning of a manipulator. The dynamic calibration identifies those parameters that influence motion characteristics. Various procedures have also been developed to identify calibration parameters. Further, there are several accuracy measurement techniques including cable measuring system [41], machine vision [42], theodlite system, and laser tracking [43]. Accurate calibration will lead to significant accuracy improvement (generally by a factor of 10).

Post Processing

Once a robot program has been fully developed, it is necessary to translate it into the control language of the target robot [44]. Post processing is divided into three stages: reformatting, translation, and downloading [45]. Reformatting verifies the input file for syntactical errors and adds appropriate motion and function information, determines coordinate frame for command, and establishes sequences for

© 2001 by CRC Press LLC

sub-tasks. Translation phase translates the list of information as per rules. This stage deals with the robot's natural language and it will be different for robots that have different format/grammar and thus manipulator-level languages [46]. The program that is generated by the translator must be loaded onto the robot's controller memory. As with any numerically controlled (NC) machines, there are two methods to download the program: downloading the program using a serial interface and downloading the program using a portable medium (e.g., floppy disk) that is compatible with the robot controller.

9.4 CAD-Based Task Planning Implementation

The underlying strategies and individual components for task planning and programming were described in the previous section. This section will focus on procedures and implementation to perform task planning and programming. Although, the procedure has been developed based on a CAD modeling facility, it is similar to a majority of other research-based as well as commercial planning and programming systems [5]. Furthermore, the focus of our attention will be confined to layered and flexible assembly/disassembly operation, as this is one of the most difficult robotic applications.

System Structure

The detailed structure of the system consists of six modules, shown in Fig. 9.10. These include:

- design modeling of assembly parts and work cell
- task specification
- sorting and task decomposition
- verification and simulation
- program transfer to the work cell controller
- work cell controller communication link

Here, we direct our attention to the structure and procedure for a planning and programming system built on a CAD modeling platform (initially on Medusa, currently on AutoCAD). Therefore, assembly parts and fixtures are the components that must be physically manipulated and are required for planning. The assembly parts and other components in the work cell are modeled using the solid modeler in CAD environment. The tasks to be performed are specified during the task planning session. This procedure tags information about the tasks onto the workpiece model. An automated grasp planning may be added at this level. However, for the purpose of the overall system development, the grasp locations are specified manually via frame assignment. Grasp planning is still an important area of research and no commercial systems provide such capabilities.

The planner retrieves the information tagged onto the models and performs sorting and matching of assembly parts for various stages of assembly. This function of the planner, known as access/retrieval, has been the focus of automated object-oriented task decomposition for layered assembly. Thus, the appropriate software modules decompose the higher level task commands and automatically create a "robot-independent" program. It must be emphasized that automated path planning would generally be integrated into the planner at this level of operation.

A trajectory planner solves the inverse kinematic equations that perform linear and joint interpolation. The time and robot related constraints can also be included at this level. This module evaluates the joint angles for linear and joint interpolation and generates a 'robot-dependent' program. The 'robot-dependent' program contains commands and point-to-point motion data files for a specific manipulator. The off-line verification or simulation of the program is the primary aim of any planning and programming system. Therefore, at this level, the user can view the robot movement and assembly operation for final verification and validation of the robot program.

The final process is to download the command and point-to-point motion data files to the work cell controller. The work cell control software retrieves the command and "point-to-point" motion data files, generates commands in a format that is understood by the robot, and sends these commands to the

© 2001 by CRC Press LLC

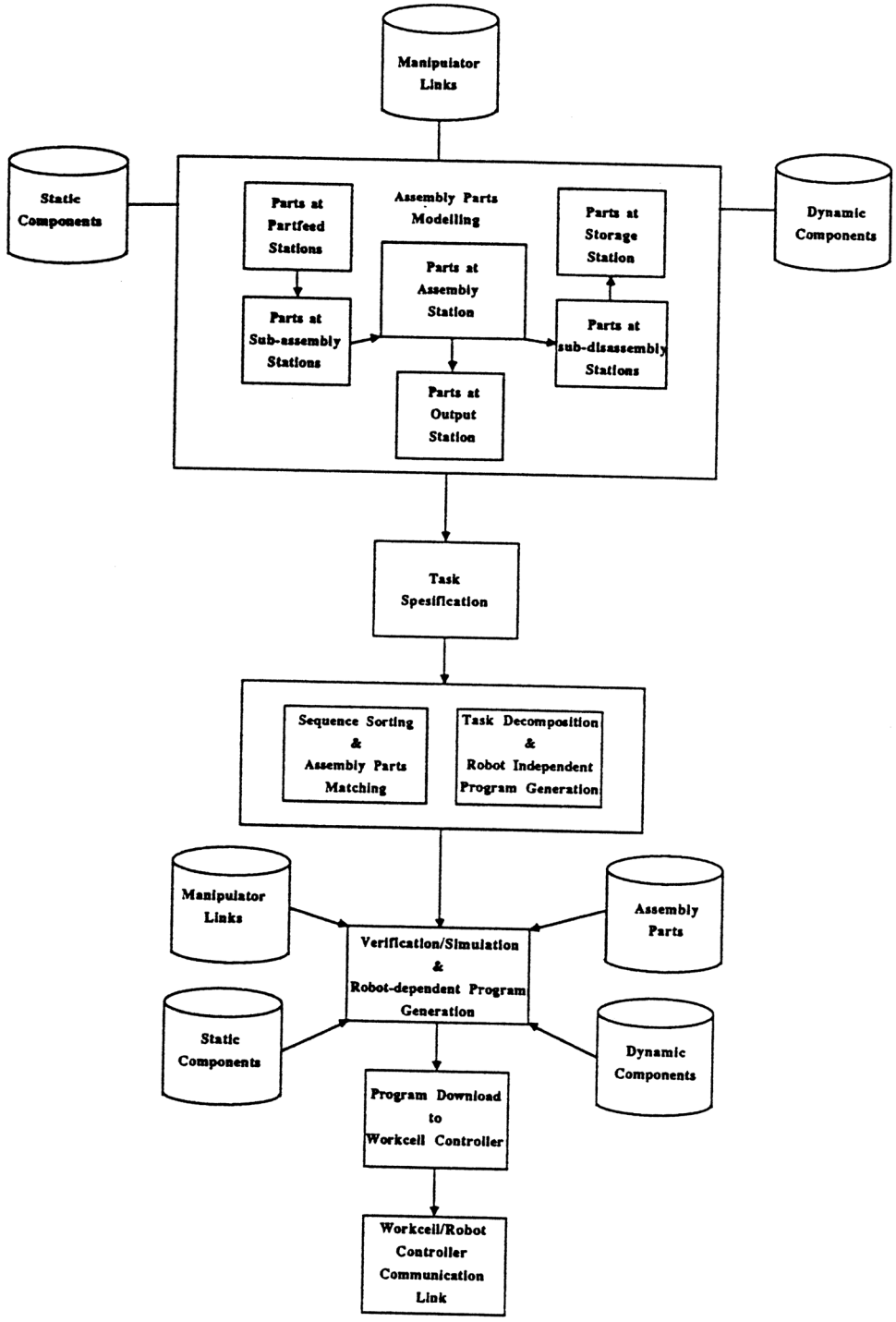

FIGURE 9.10 Structure of the task planning and programming.

© 2001 by CRC Press LLC

robot which in turn performs the assembly operation. The capability to perform workstation calibration is also built to readjust positions in the robot programs.

Design Modeling of Assembly Parts and Work Cell

Modeling of various components of a robot manufacturing cell is divided into four categories: static components, dynamic components, robot manipulator, and assembly parts. The static components of the manufacturing cell are elements which are stationary and may include assembly tables, partfeed machines, fixtures, and part storage. The dynamic components of the cell are elements which may change their locations during the manufacturing operation. The dynamic components are designated with a dynamic reference datum and examples of these components are interchangeable grippers, turntables, and pallets a on conveyor system. The static and dynamic components are modeled as convex polyhedrals and stored in the appropriate format in the database [46]. The models are retrieved and displayed by the verification module during simulation of robotic assembly operations.

The manipulator is modeled as a series of links in the synchronized (i.e., home) position and with a coordinate frame attached to each link [47]. This information is not required for assembly task specification or robot-independent program generation. As with static and dynamic components within the cell, it is only used for the final verification of the assembly task and simulation purposes during the robot-dependent program generation. Assembly parts are modeled with respect to the reference coordinate frame fixed to each station. Thus, the models of the assembly parts are stored within the CAD database just as they would appear at various assembly stages. This procedure will also produce the correct "pose" of the assembly parts referenced by a local coordinate frame fixed to each station throughout the robotic assembly process. The procedure follows a practical structure for assembly/disassembly operations (Fig. 9.11). Figure 9.12 shows the structure for access/retrieval of 3-D models representing various assembly stages.

Therefore, the assembly process is represented in 3-D model drawings and the task planning is performed on the these models. Figure 9.13 shows an example of the assembly part coordinate frames

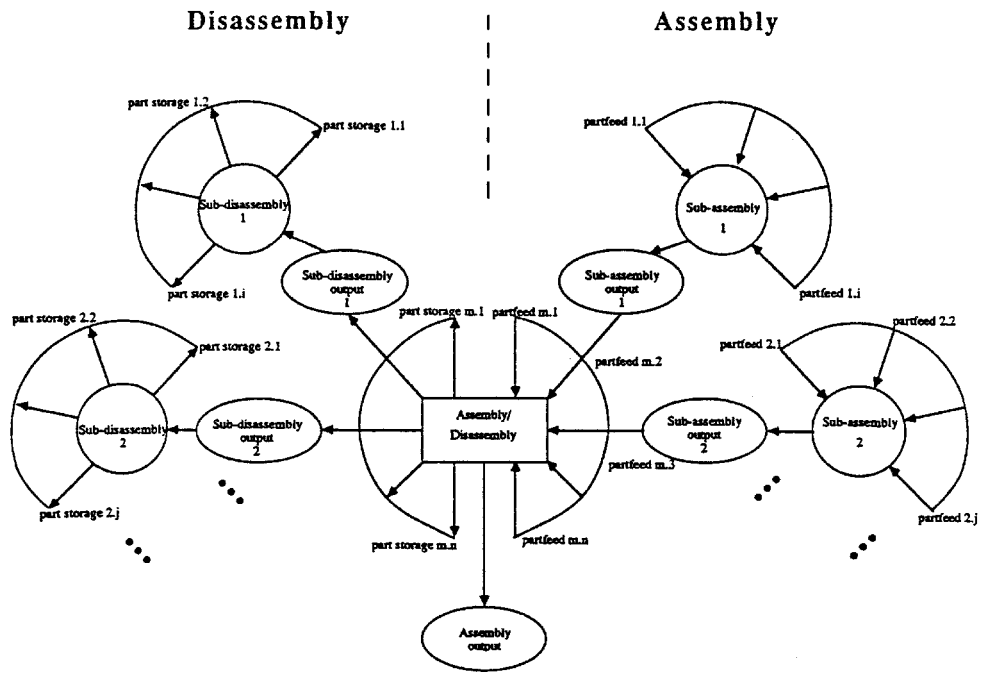

FIGURE 9.11 Representation of assembly/disassembly stages.

© 2001 by CRC Press LLC

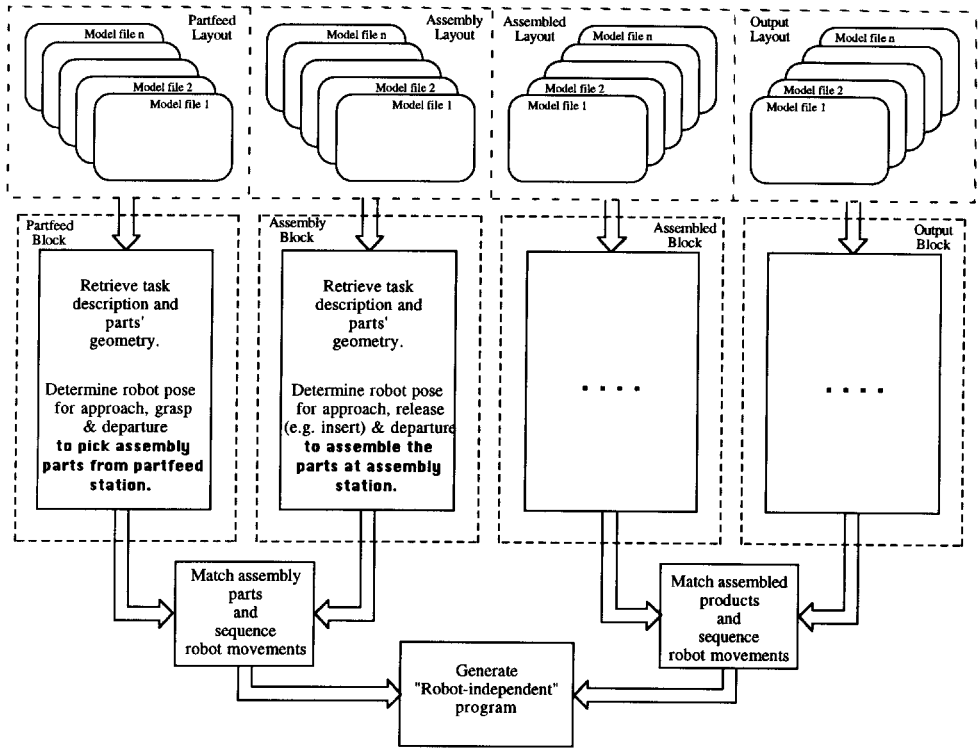

FIGURE 9.12 Structure for access/retrieval and processing of assembly task specifications.

attached to each station. It must be noted that the design modeling of the individual assembly parts and the assembled product is a common practice in industry today. Hence, minimal effort is required in adding appropriate frames and description to these models for planning and programming purposes.

Task Specification

The specification of the task can be carried out using objective commands such as "Pick," "Place," "Insert," etc. In general, frames are assigned (i.e., tagged) to the model to signify the grasp locations. It must be emphasized that task planning has been partially built into the design modeling procedure and it will also be completed by the task decomposition software modules [48]. Other information such as robot speed and robot motion mode (e.g., R: rectangular, J: joint) may also be attached to the model (Fig. 9.14). The layer on which these commands are placed may be turned off, if required.

A macro program has also been developed to allow the specification of complex approach and departure vectors. The program forms construction lines and computes the approach vector, (a_H), orientation vector (o_H), and normal vector (n_H). Hence, the hand orientation is fully defined with respect to a specified frame.

$$T = \left[\begin{array}{c|c} n_F\, O_F\, a_F & p_F \\ \hline O\ O\ O & 1 \end{array}\right] \tag{9.9}$$

The approach (a), orientation (o), and normal (n) vectors in their normal form do not have much meaning to the manufacturing engineer and carrying all the elements of the matrix is not very efficient.

© 2001 by CRC Press LLC

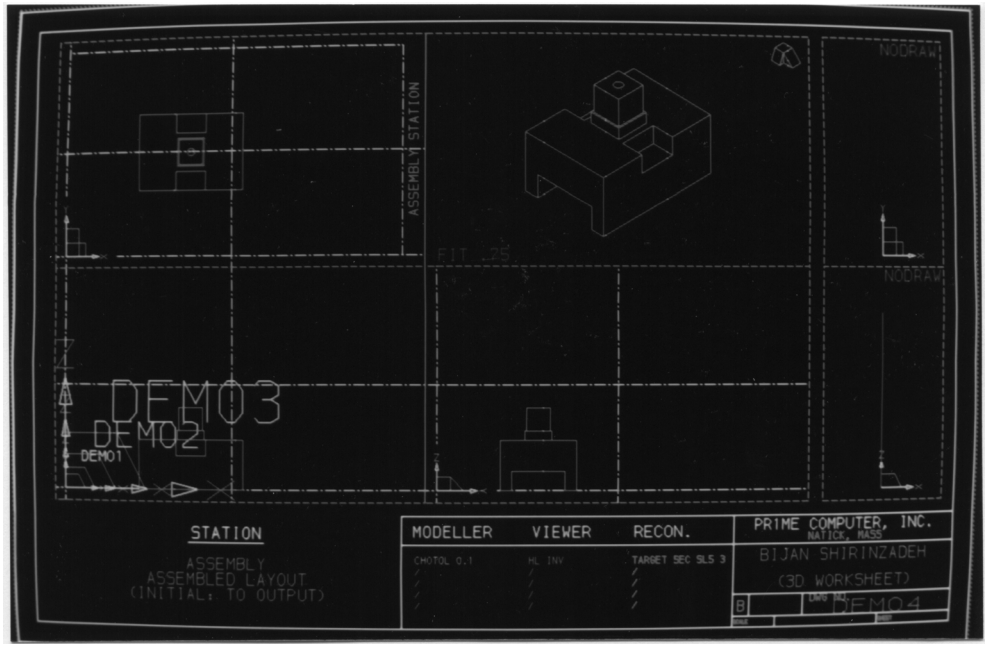

FIGURE 9.13 Design modeling of assembly parts.

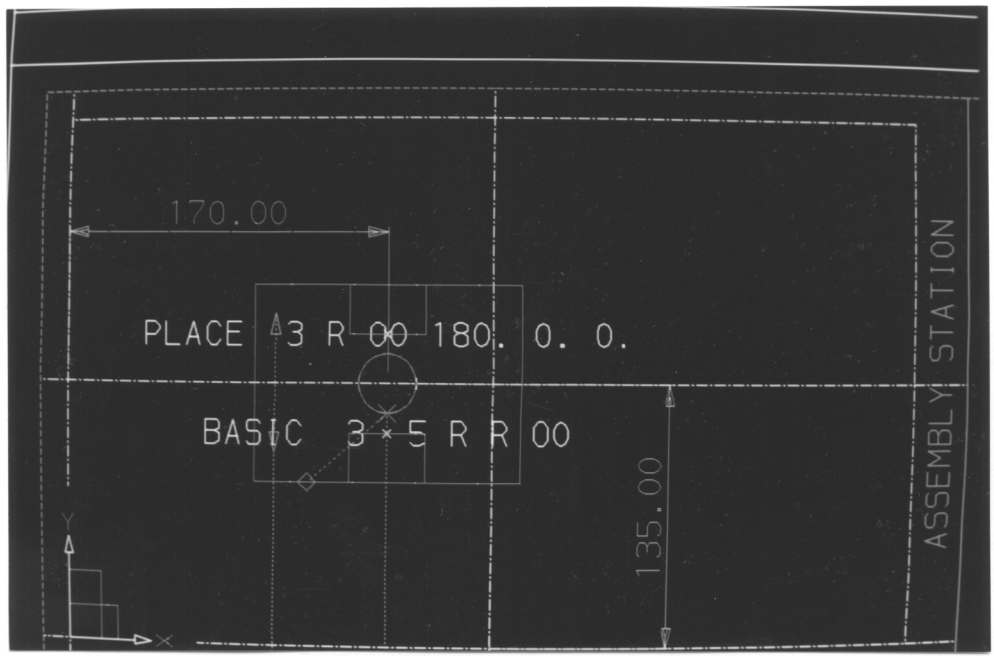

FIGURE 9.14 Example of the task specification on CAD model.

© 2001 by CRC Press LLC

Therefore, once all the components of the vectors in the matrix T_H (i.e., hand coordinate frame) are determined and normalized, the orientation angles, Roll, ϕ (i.e., rotation about the Z axis), Pitch, θ (i.e., rotation about the Y axis), and Yaw, ψ (i.e., rotation about the X axis) are determined using the following relationships:

$$\tan\phi = n_{H_Y}/n_{H_X} \tag{9.10}$$

$$\tan\theta = -n_{H_Z}/(\cos\phi\, n_{H_X} + \sin\phi\, n_{H_Y}) \tag{9.11}$$

$$\tan\psi = (\sin\phi\, a_{H_X} - \cos\phi\, a_{H_Y})/(\sin\phi\, o_{H_X} + \cos\phi\, o_{H_Y}) \tag{9.12}$$

where n_H, a_H, and o_H represent the normal, approach, and orientation vectors in hand coordinate frame, respectively. The above orientation angles (specified in degrees) are reported to the designer and included in the command. Wire frames may be used to create intricate paths when these are required. It must be emphasized that the above procedure is not required for layered assembly (i.e., assembly directions from the above are generated automatically using an offset vector).

Task Decomposition

In commercially available systems such as CATIA, IGRIP, and ROBCAD, the task and individual manipulator motion must be specified sequentially by the user. However, in the approach adopted system a software program was developed to access the CAD data base (i.e., 3-D model data) and automatically retrieve the geometrical data. This software program also retrieves task instructions tagged to the model (i.e., stored in the CAD data base during the task specification phase). The attributes tagged to each planned task are also retrieved. The objects' names are used to match assembly parts in various stages such as "partfeed" and "assembly" within the assembly process. Thus, the software module performs sorting and sequencing of the assembly operations and robot movements and, finally, generating a 'robot-independent' program. Figure 9.12 shows the sequence of retrieval of model data together with the subsequent task and object matching operations. The robot program will then be generated in a format appropriate for subsequent simulation. The geometrical data will also be stored in a structured boundary representation (B-rep) format as discussed previously (Fig. 9.2).

Figure 9.15 shows the structure of the software for the "main block" (e.g., assembly block) of Fig. 9.12, and thus a detailed description of the above software operations. Other blocks utilize a similar structure. As relevant robotic manufacturing information is gathered, these are stored in the data base. The software program initially accesses the model file (which contains the model of the assembly layout), extracts the designer's instructions, and begins to work backward to subassembly and partfeed models (i.e., feeders, pallets, etc.). The name of each object from the assembly layout is then matched against those from the subassembly and partfeed layouts and sequencing of assembly operation is performed, thereby completing the process of programming of the assembly operation. The program then sequences the output operation by retrieving the necessary information from the model files containing the initial location of the assembled product in the assembly station and the final location of the assembled parts in the output station.

Transformation of Robot Locations and Workstation Calibration

The output from the previous software module is a series of tasks, generally referred to as a "robot-independent" program. This contains robot poses fixed to the individual station coordinate frame. The next phase involves transformation of these movements into grasp locations attached to the universe (i.e., world) coordinate frame. This is generally defined as the coordinate frame at the base of the robot (i.e., robot coordinate frame). This software module utilizes transformation vectors to describe the

© 2001 by CRC Press LLC

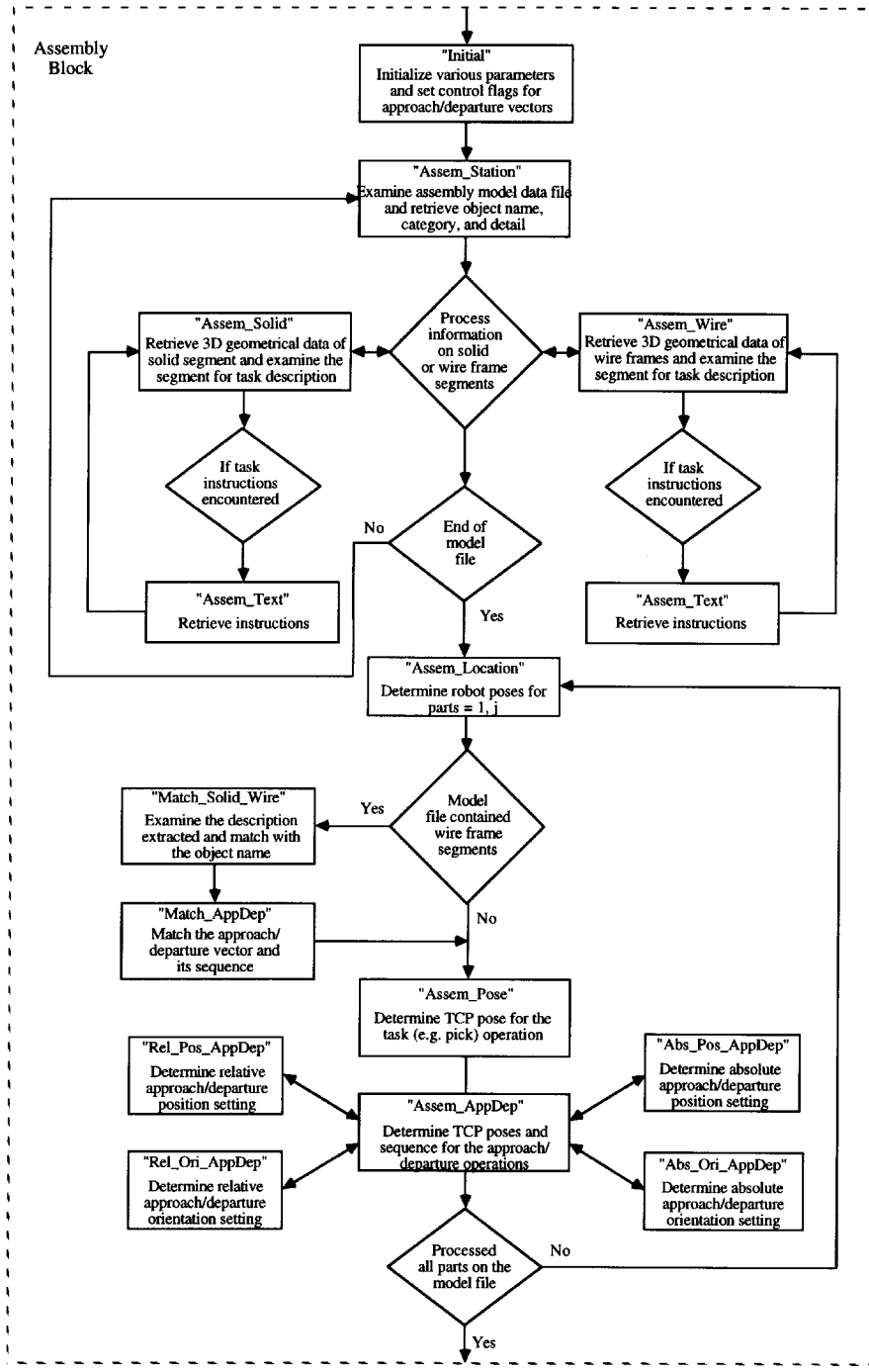

FIGURE 9.15 Detailed flow diagram of the 'Main Block' in the software program.

relationships between station coordinate frames and the universe coordinate frame. These transformations may be modified by simply specifying the translation transform (i.e., vector components x, y, and z) and orientation transform (i.e., roll, pitch, and yaw orientation about x, y, and z). Figure 9.16 shows the schematic arrangement of the coordinate frames.

© 2001 by CRC Press LLC

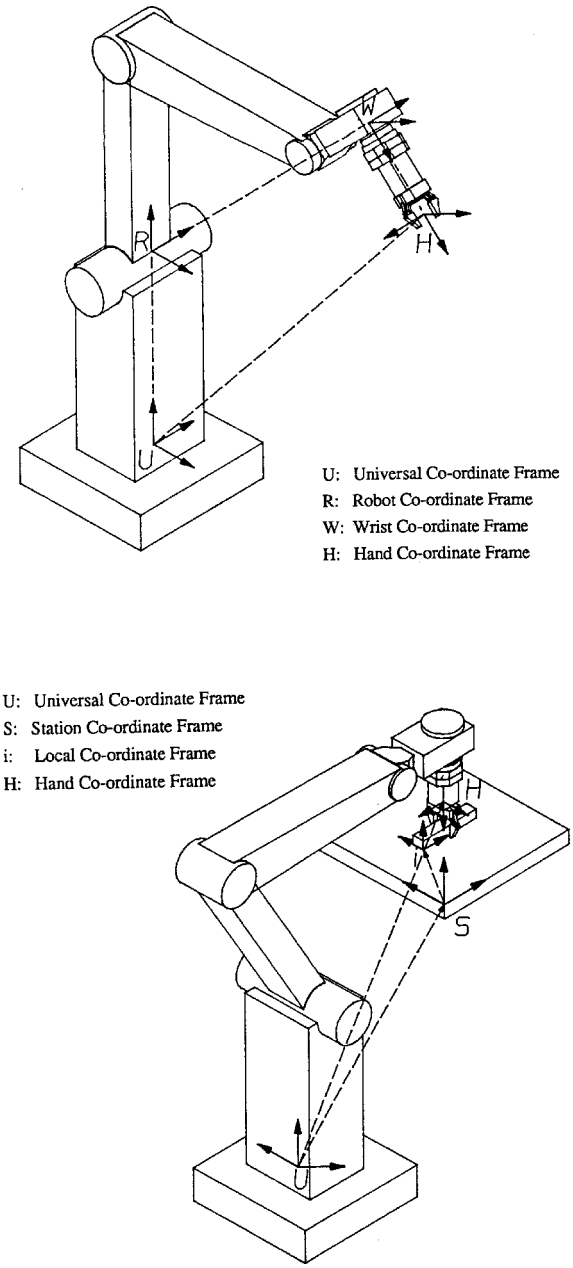

FIGURE 9.16 Schematic representation of assigning coordinate frames.

The following relationship may be used to determine the pose of the manipulator with respect to the universe coordinate frame:

$$ {}^{S}T_{U} \cdot {}^{U}T_{H} = {}^{S}T_{i} \cdot {}^{i}T_{i-1} \cdot \ldots \cdot {}^{1}T_{H} \tag{9.13} $$

where ${}^{S}T_{U}$ is the transformation describing the location of the universe coordinate frame with respect to a particular station coordinate frame, ${}^{U}T_{H}$ is the transformation matrix describing the location of the hand with respect to the universe coordinate frame, ${}^{S}T_{i}$ is the transform describing the location of the

© 2001 by CRC Press LLC

i-th local coordinate frame with respect to the station coordinate frame, and ${}^{i}T_{i-1}$ is the transform describing the (i-1)th local coordinate frame with respect to the i-th local coordinate frame. Finally, ${}^{1}T_{H}$ is the transform describing the hand coordinate frame on the assembly part with respect to the 1-th local coordinate frame. Pre-multiplying both sides of the equation by ${}^{S}T_{U}^{-1}$ will result in the position and orientation of the hand with respect to the universe coordinate frame:

$$ {}^{U}T_{H} = {}^{S}T_{U}^{-1} \cdot {}^{S}T_{i} \cdot {}^{i}T_{i-1} \cdot \ldots \cdot {}^{1}T_{H} \tag{9.14} $$

If there is no local frame and the location of the hand is with respect to the station coordinate frame, then the equation can be simplified to:

$$ {}^{U}T_{H} = {}^{S}T_{U}^{-1} \cdot {}^{S}T_{H} \tag{9.15} $$

where all the transformation matrices are of the form:

$$ T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.16} $$

In practice, the transform ${}^{S}T_{U}^{-1}$ is obtained by jogging the robot (i.e., moving the manipulator arm under manual control) to the coordinate frame of a particular station and performing the status request. The position vector acquired in this manner may be directly used in the above software module. Furthermore, the technique may also be used to calibrate the robot workstation and thus, reduce the robot positioning errors associated with advanced off-line programming techniques. However, it must be emphasized that this approach is concerned with digitizing the workstation and it is not a replacement for robot calibration.

The calibration procedure utilizes a precision-made rod referred to as the "calibration rod." This is attached to the center line of the end-effector. This procedure requires the operator to jog the robot to various control locations ($C_1 - C_4$), align the tip of the calibration rod with the control locations on a particular station, and record their positions [50]. Fig. 9.17 shows a schematic illustration of the procedure and a schematic diagram of the geometrical information. The calibration software determines the reference datum for the workstation (i.e., C_1). It also generates transformation vectors with respect to the robot base coordinate frame U, and determines the calibration factors for the workstation (as shown in Fig. 9.17). It must be noted that the tool center point (TCP) offset for the robot end-effector must be set accordingly.

The industrial robot manipulators possess good repeatability but not absolute accuracy. The approach described employs the manipulator arm to obtain the reference coordinate frame fixed to each station and the programmed movements are calibrated in relation to this frame. This provides a better absolute accuracy over the range of assembly workstation. The technique reduces the inaccuracies in positioning due to long range movement in relation to the robot coordinate frame. The approach has been developed so that it can also use a vision system to perform the calibration [51].

Robot Kinematics and Assembly Process Simulation Module

The simulation module is the final step in order to generate the command and point-to-point motion data files. This is generally referred to as the "robot native program." This module retrieves the hand poses generated by the previous software operations; determines the pose of the wrist with respect to the robot coordinate frame, and solves the robot kinematic equations to determine the robot joint angles for the

© 2001 by CRC Press LLC

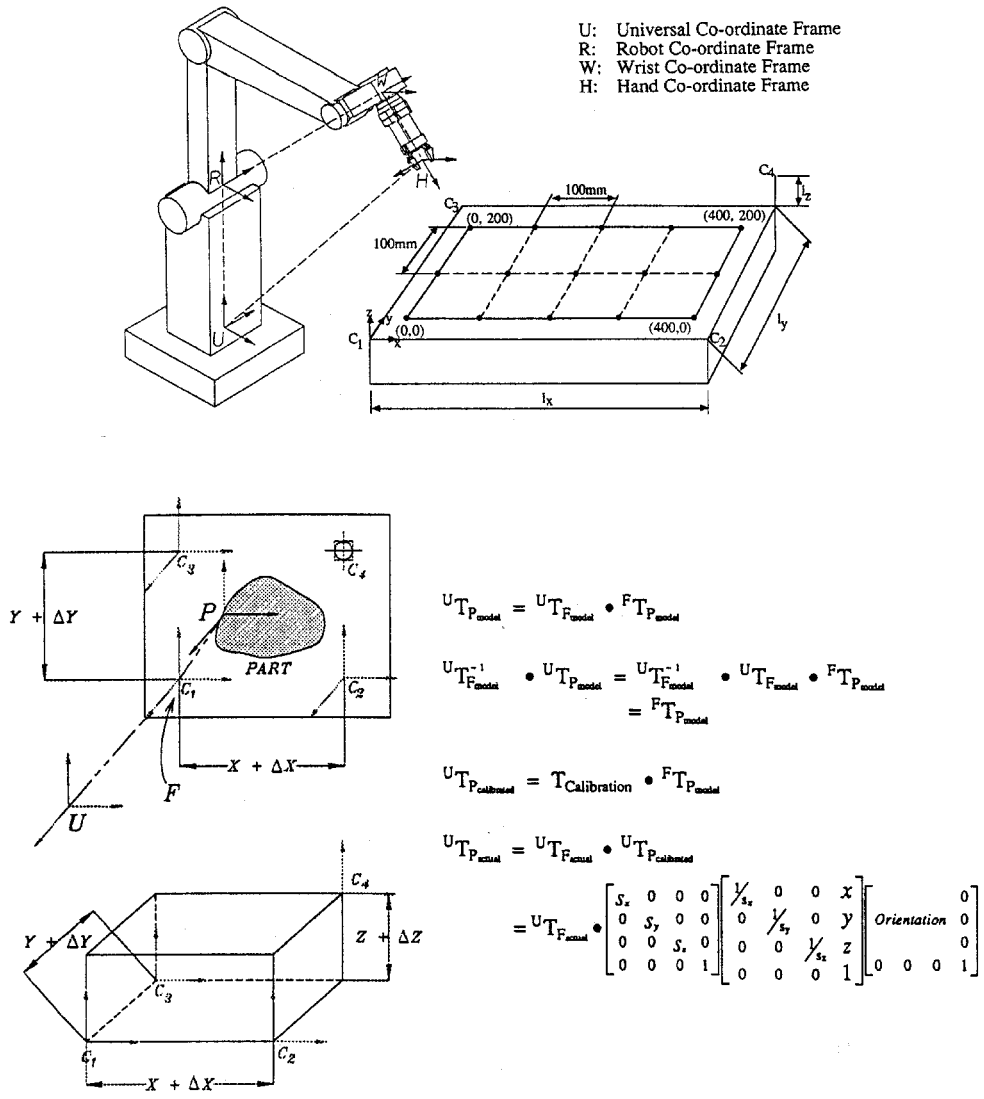

FIGURE 9.17 Calibration procedure and geometrical aspects of the workstation.

purpose of simulation. It also determines the location of the robot wrist in quaternion, as required by the robot controller (e.g., ABB IRB 2400), for the purpose of generating the command and motion data file(s).

The simulation can generate motion trajectories according to straight line motion increment or joint interpolated motion increment. As joint angles for a given pose of the wrist are determined, they are checked against the limits of the robot joint angles. If, for a given pose, any of the calculated robot joint angles exceeds the limit, then the module reports this to the user and logs the event. The module retrieves the geometrical data describing robot linkages, stations, feeders, pallets, and assembly parts at various stages of the assembly process and any other equipment and objects that are required to be in the scene. The module determines the location of all the objects in relation to the universe frame using the appropriate transformations. Simulation process is based on the method of coordinate transformation for describing both the manipulator and objects (Fig. 9.18).

The main intention of such graphics systems is to provide the visualization of the entire work cell as well as the animation of the movements of robots and peripherals. In addition, such systems can be

© 2001 by CRC Press LLC

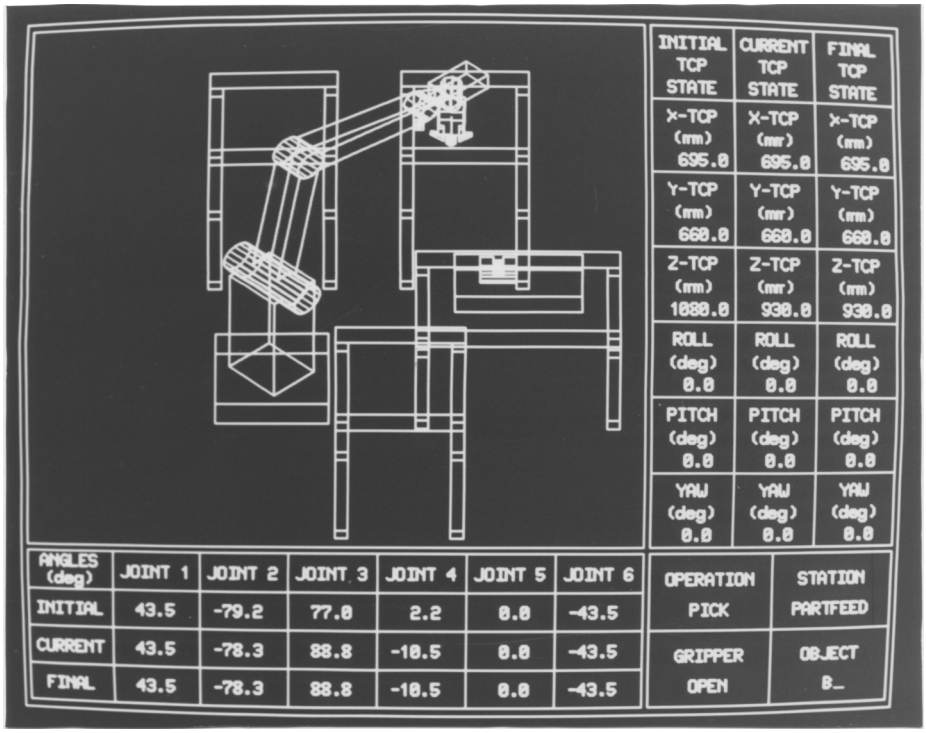

FIGURE 9.18 Photograph showing the simulation of the assembly.

viewed as building blocks toward the integrated CIM solutions that support the manufacturing cycle, from CAD to CAM and distribution [52–54].

9.5 Future Research

The task planning and programming system described is currently being ported to a PC-based CAD package (AutoCAD). The technique is also being examined for implementation on a large modeling and simulation platform such as Deneb's Envision system. In addition, the strategies and formulation are being further developed to include automatic sequence and assembly directions using screw theory. In this approach, disassembly sequence and direction may be generated first and then reversed to determine possible assembly sequence and directions. The planner is being developed further to include a more extensive library of manufacturing operations and to include rules and mathematical formulations for engagement and disengagement of assembly parts.

9.6 Conclusions

Computer-aided design (CAD) has become an integral part of design in manufacturing and many other fields of engineering. It has also become obvious that the cost of programming robots is an important issue in robot-based flexible manufacturing and CIM environment.

This chapter has briefly presented strategies for task planning and programming. This chapter has also described individual modules (i.e., building blocks) for such systems. The structure and development strategies for a CAD-based and off-line task planner has also been described. The development employed a commercial CAD package for the process of design modeling of the assembly parts and task specification.

© 2001 by CRC Press LLC

The approach adopted in development of a dedicated software program to retrieve parts' geometry and instructions which are embedded into the design models was presented.

The strategies to retrieve the appropriate task information from the assembly layout and work backward to automatically generate lower level task commands was also presented. The system is being modified and further developed for operation on PC-based platform and Unix-based Envision (Deneb) modeling and simulation platform.

Acknowledgments

This research is partly supported by an Engineering Research Council (ERC) Small Grant from Monash University, a Monash Research Fund (MRF) grant, and a Harold Armstrong Research Fund. The author wishes to thank his research assistants at Robotics & Mechatronics Research Laboratory, Department of Mechanical Engineering, Monash University.

References

1. H. K. Rampersad, State of the art in robotic assembly. Int. J. Industrial Robot, Vol. 22, No. 2, pp. 10–13, 1995.
2. K. Schroer, Robot calibration—closing the gap between model and reality. Int. J. Industrial Robot, Vol. 21, No. 6, pp. 3–5, 1994.
3. J. C. Latombe, Une analyse structuree d'outils de programmation pour la robotique industrielle. Proc. Int Seminar on Programming Methods and Languages for Industrial Robots. INRIA, Rocquencourt, France, June 1979.
4. S. Bonner, K. G. Shin, A comparative study of robot languages. Computer Dec, pp. 82–96, 1982.
5. D. M. Lee, W. H. ElMaraghy, ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators. Computer-Aided Engineering Journal, Vol. 7, No. 5, pp. 141–148, October 1990.
6. A. R. Thangaraj, M. Doelfs, Reduce downtime with off-line programming, Robotics Today, Vol. 4, No. 2, pp. 1–3, 1991.
7. Y. Regev, The evolution of off-line programming. Industrial Robot, Vol. 22, No. 3, pp. 3, 1995.
8. W. A. Gruver, B. I. Soroka, J. J. Craig, T. L. Turner, Evaluation of commercially available robot programming languages. Proc. 13th Int. Symposium on Industrial Robots, Chicago, 1983.
9. J. J. Craig, Issues in the design of off-line programming systems. Fourth Int. Symposium on Robotics Research, University of California, Santa Cruz, pp. 379–389, 1987.
10. M. Kortus, T. Ward, M. H. Wu, An alternative approach to off-line programming. Industrial Robot, Vol. 20, No. 4, pp. 17–20, 1995.
11. P. Sorenti, GRASP for simulation and off-line programming of robots in industrial applications. Proc. of Conference on Welding Engineering Software, Essen, DVS N0 156, pp. 55–58, Sept 1993.
12. R. J. Popplestone, A. P. Ambler, T. M. Bellos, An interpreter for a language for describing assemblies. Artificial Intelligence, Vol. 14, pp. 79–107, Aug. 1980.
13. L. I. Liberman, M. A. Wesley, AUTOPASS: an automatic programming system for computer controlled mechanical assembly. IBM Journal of Research and Development, pp. 321–333, July 1977.
14. R. Mattikalli, D. Baraff, P. Khosla, Finding all stable orientations of assemblies with friction. IEEE Trans. Robotics and Automation, Vol. 12, No. 2, pp. 290–301, 1996.
15. J. J. Craig, Introduction to Robotics, 2nd, Addison-Wesley Publishing, 1989.
16. B. Shirinzadeh, A simplified approach to robot programming from CAD system. Fourth Int. Conference on Manufacturing Engineering, pp. 141–145, Brisbane, Australia, 1988.
17. T. N. Wong, S. C. Hsu, An off-line programming system with graphics simulation. Int. J. Advanced Manufacturing Technology, Vol. 6, pp. 205–223, 1991.
18. J. Shah, P. Sreevalsan, A. Mathew, Survey of CAD/feature-based process planning and NC programming techniques. Computer-Aided Engineering Journal, Vol. 8, No. 1, pp. 25–33, 1991.

© 2001 by CRC Press LLC

19. B. Shirinzadeh, A CAD-based design and analysis system for reconfigurable fixtures in robotic assembly. Computing & Control Engineering Journal, Vol. 5, No. 1, pp. 41–46, 1994.
20. B. Shirinzadeh, Strategies for planning and implementation of flexible fixturing systems in a computer integrated manufacturing environment. J. Computers in Industry, Vol. 30, pp. 175–183, 1996.
21. J. Rocha, C. Ramos, Task planning for flexible and agile manufacturing systems, McGraw-Hill Inc, 1987.
22. L. Laperriere, H. A. ElMaraghy, Automatic generation of robotic assembly sequences. Int. J. Advanced Manufacturing Technology, Vol. 6, pp. 299–316, 1991.
23. V. N. Rajan, S. Y. Nof, Minimal precedence constraints for integrated assembly and execution planning. IEEE Trans. Robotics and Automation, Vol. 12, No. 2, pp. 175–186, 1996.
24. H. Chu, H. A. ElMaraghy, Integration of task planning and motion control in a multi-robot assembly workcell. Robotics & Computer-Integrated Manufacturing, Vol. 10, No. 3, pp. 235–255, 1993.
25. H. L. Welch, R. B. Kelley, The analysis of potential mating trajectories and grasp sites. Int. J. of Advanced Manufacturing Technology, Vol. 8, pp. 320–328, 1993.
26. Z. Shiller, S. Dubowsky, Robot path planning with obstacles, actuator, gripper, and payload constraints. Int. J. Robotics Research, Vol. 8, No. 6, pp. 3–18, 1989.
27. S. Ahmad, J. T. Feddema, Static grip selection for robot-based automated assembly systems. J. Robotic Systems, Vol. 4, No. 6, pp. 687–717, 1987.
28. Y. L. Xiong, D. J. Sanger, D. R. Kerr, Geometric modeling of bounded and frictional grasps, Robotica, Vol. 11, pp. 185–192, 1993.
29. S. Cameron, Obstacle avoidance and path planning. Industrial Robot, Vol. 21, No. 5, pp. 9–14, 1994.
30. T. D. Luk, Planning collision-free paths in Cartesian space. Robotics Today, Vol. 5, No. 3, pp. 1–3, 1992.
31. T. Lozano-Perez, Automatic planning of manipulator transfer movements. IEEE Trans. Systems, Man, Cybernetics SMC-11, 10, pp. 681–689, 1981.
32. Y. Itoh, M. Idesawa, T. Soma, A study on robot path planning from a solid model. J. Robotics Systems, Vol. 3, No. 2, pp. 191–203, 1986.
33. S. Stifter, Collision detection in the robot simulation system SMART. Int. J. Advanced Manufacturing Technology, Vol. 7, pp. 277–284, 1992.
34. M. J. Tsai, S. Lin, M. C. Chen, Mathematical model for robotic arc-welding off-line programming system. Int. J. Computer Integrated Manufacturing, Vol. 5, No. 4 & 5, pp. 300–309, 1992.
35. R. O. Buchal, D. B. Cherchas, F. Sassani, J. P. Duncan, Simulated off-line programming of welding robots. Int. J. Robotics Research, Vol. 8, No. 3, pp. 31–43, 1989.
36. S. H. Suh, J. J. Lee, Y. J. Choi, S. K. Lee, Prototype integrated robotic painting system: software and hardware development. Journal of Manufacturing Systems, Vol. 12, No. 6, pp. 463–472, 1993.
37. H. Wapenhans, J. Holzl, J. Steinle, F. Pfeiffer, Optimal trajectory planning with application to industrial robot. Int. J. Advanced Manufacturing Technology, Vol. 9, pp. 49–55, 1994.
38. K. S. Moon, K. Kim, F. Azadivar, Optimum continuous path operating conditions for maximum productivity of robotic manufacturing systems. J. Robotics & Computer-Integrated Manufacturing, Vol. 8, No. 4, pp. 193–199, 1991.
39. M. C. Leu, R. Mahajan, Computer graphic simulation of robot kinematics and dynamics. Proc. 8th International Conference on Robots, pp. 1–20, June, 1984.
40. G. Wiitenberg, Developments in off-line programming: an overview. Industrial Robot, Vol. 22, No. 3, pp. 21–23, 1995.
41. J. F. Quinet, Calibration for offline programming purpose and its expectations. J. Industrial Robot, Vol. 22, No. 2, pp. 10–13, 1995.
42. B. Shirinzadeh, Y. Shen, Three dimensional calibration of robotic manufacturing cell using machine vision techniques. Proc. of Pacific Conference on Manufacturing, pp. 360–365, Seoul, Korea, 1996.

© 2001 by CRC Press LLC

43. M. Vincze, J. P. Prenninger, H. Gander, A laser tracking system to measure position and orientation or robot end-effectors under motion. Int. J. Robotics Research. Vol. 13, No. 4, pp. 305–314, 1994.
44. K. V. Kamisetty, Development of a CAD/CAM robotic translator for programming the IBM 7535 SCARA robot off-line. J. Computers in Industry, Vol. 20, pp. 219–228, 1992.
45. K. V. Steiner, M. Keefe, Interactive graphics simulation with multi-level collision algorithm. Journal of Manufacturing Systems, Vol. 11, No. 6, pp. 462–469, 1992.
46. B. Shirinzadeh, H. Tie, Object-oriented task planning and programming system for layered assembly and disassembly operations. Proc. of the ARA/IFR International Conference, Robots for Competitive Industries, pp. 354–360, Brisbane, 1993.
47. P. Fanghella, C. Galletti, E. Giannotti, Computer-aided modeling and simulation of mechanisms and manipulators. J. Computer-Aided Design, Vol. 21, No. 9, pp. 577–583, 1989.
48. B. Shirinzadeh, H. Tie, G. Lin, Computer integrated task planning and programming of robotic assembly operations. Proc. Second International Conference, CIM, Vol. 2, pp. 599–605, Singapore, 1993.
49. P. K. Venuvinod, Automated analysis of 3-D polyhedral assemblies: assembly directions and sequences. Journal of Manufacturing systems, Vol. 12, No. 3, pp. 246–252, 1993.
50. B. Shirinzadeh, H. Tie, Experimental investigation on the performance of a reconfigurable fixturing system. International Journal of Advanced Manufacturing Technology, Vol. 10, pp. 330–341, 1995.
51. B. Shirinzadeh, C. Paragreen, W. Lee, Calibration of robotic cell for CAD-based planning using machine vision techniques. Proc. of Twelfth International Conference on Robotics and Factories of the Future, pp. 132–137, London, 1996.
52. T. S. Kang, B. O. Nnaji, Feature representation and classification for automatic process planning systems. Journal of Manufacturing Systems, Vol. 12, No. 2, pp. 133–145, 1993.
53. S. Chakrabarty, J. Wolter, A structure-oriented approach to assembly sequence planning. IEEE Trans. Robotics and Automation, Vol. 13, No. 1, pp. 14–29, 1997.
54. C. P. Tung, A. Kak, Integrating sensing, task planning, and execution for robotic assembly. IEEE Trans. Robotics and Automation, Vol. 12, No. 2, pp. 187–201, 1996.
55. G. Kim, S. Lee, G. Bekey, Interleaving assembly planning and design. IEEE Trans. Robotics and Automation, Vol. 12, No. 2, pp. 246–251, 1996.

© 2001 by CRC Press LLC