BLDSC no:- DX 170989

Graphical Modelling of Modular Machines

By

Xiu Tian Yan

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of

Doctor of Philosophy

of the Loughborough University of Technology

January 1992

Department of Manufacturing Engineering

Loughborough University of Technology

w9922215

TO MY FAMILY

Acknowledgments

I am grateful to my motherland China for funding my research study from Sino-British Friendship Scholarship Scheme.

I would like to express my sincere thanks to Professor Richard H Weston and Dr. Keith Case, my research supervisors, for their invaluable guidance, encouragement and patience during my study at Loughborough. Their thorough reading, modification on English and constructive comments have greatly improved this work. I also thank Dr. Phil Moore for his friendship and help. Special thanks go to my research project colleagues, Mr. Robb Doyle, Mr. A.S. Goh, Mr. Neil Armstrong, Mr. Jack Gascoigne, Mr. Ian Coutts, Dr. Rob Harrison, Mr. Chris Wright, Mr. Alan Booth, Mr. Andy Carrot. I would like thank those in the department who are kind and helpful. They include Dr. J. Middle, Mr. D. Walters, Dr. S. Newman, Mrs. M. Carden, Mr. J. Zhang, Dr. W Wang, Dr. J Pu, Mrs. H. Jiao, Mr. B Wong.

Finally, and most importantly, I wish to express my love and appreciation to my family for their immeasurable patience, encouragement and support.

Abstract

This research is aimed at advancing machine design through specifying and implementing (in "proof of concept" form) a set of tools which graphically model modular machines. The tools allow mechanical building elements (or machine modules) to be selected and configured together in a highly flexible manner so that operation of the chosen configuration can be simulated and performance properties evaluated. Implementation of the tools has involved an extension in capability of a proprietary robot simulation system. This research has resulted in a general approach to graphically modelling manufacturing machines built from modular elements.

A focus of study has been on a decomposition of machine functionality leading to the establishment of a library of modular machine primitives. This provides a useful source of commonly required machine building elements for use by machine designers. Study has also focussed on the generation of machine configuration tools which facilitate the construction of a simulation model and ultimately the physical machine itself. Simulation aspects of machine control are also considered which depict methods of manipulating a machine model in the simulation phase. In addition methods of achieving machine programming have been considered which specify the machine and its operational tasks. Means of adopting common information data structures are also considered which can facilitate interfacing with other systems, including the physical machine system constructed as an issue of the simulation phase. Each of these study areas is addressed in its own context, but collectively they provide a means of creating a complete modular machine design environment which can provide significant assistance to machine designers.

Part of the methodology employed in the study is based on the use of the discrete event simulation technique. To easily and effectively describe a modular machine and its activity in a simulation model, a hierarchical ring and tree data structure has been designed and implemented. The modularity and reconfigurability are accommodated by the data structure, and homogeneous transformations are adopted to determine the spatial location and orientation of each of the machine elements.

A three-level machine task programming approach is used to describe the machine's activities. A common data format method is used to interface the machine design environment with the physical machine and other building blocks of manufacturing systems (such as CAD systems) where systems integration approaches can lead to enhanced product realisation.

The study concludes that a modular machine design environment can be created by employing the graphical simulation approach together with a set of comprehensive configuration tools. A generic framework has been derived which outlines the way in which machine design environments can be constructed and suggestions are made as to how the proof of concept design environment implemented in this study can be advanced.

Table of Contents

Chapter 1 Introduction

Computer Integrated Manufacturing (CIM) is considered by many to be strategically important in achieving responsive and effective methods of realising products. The CIM approach emphasises the effective utilisation of information which is generated at all stages of the life cycle of products and the manufacturing systems used to realise them. Information exchanged amongst many computer based manufacturing machines can lead to faster and better decisions and actions. As a result, any given CIM implementation may include a diverse range of equipment, computer systems and people, ranging from product design, manufacturing operation planning (e.g. process planning), task definition, and equipment requiring real-time computer control etc.

With respect to the broad groupings of building elements of CIM systems listed above, real-time computer controlled equipment is often characterised by a need to utilise equipment as much as possible in producing products, thereby justifying capital investment levels. Another typical characteristic of real-time computer-controlled equipment is the need for them to interact with other CIM sub-systems to realise manufacturing requirements in an efficient and timely manner. Often the efficient utilisation of automation equipment implies the need to generate machine control programmes (such as NC [Numerically Controlled] part programmes and robot task programmes) in an off-line manner, i.e. while the machine is used to produce other products. Off-line programming of automated machines has been used very successfully for NC machines and to a lesser extent for robots. The need for interaction between automated machines increases the complexity of associated operational planning and off-line machine programming activities.

The derivation of a machine design and simulation environment has been considered as a potential promising approach to advancing the objectives of CIM and to providing an aid to evaluate the operational performance of automated equipment. Within this environment, the operation planning of a machine and its interactive equipment is enabled, off-line programming for the machine and its interaction with related equipment is made and evaluated for real time control, and information sharing is also achieved.

Computer controlled modular machines are defined as machines configured by using some of the control and mechanical building elements known as modules. They are inherently reconfigurable with regard to both their control and mechanical modules and also have the potential advantage of lower initial investment compared with conventional robots. Research on modular machines is attracting increased attention but significant work remains.

This study chooses modular machines as key modelling subjects, uses computer graphical techniques, and aims at creating a graphical design and simulation environment for aggregating and simulating a modular machine along with its interactive equipment environment. The main objectives within such an environment include the establishment of a library of modular machine primitives for the frequent use of machine designers; the generation of machine configuration tools to facilitate the modelling of modular machines; the provision of simulation control and programming of a machine model; and the derivation of methods to achieve integration of simulation environment with other systems. The methodology employed in this study is to use a set of configuration tools to select modular machine primitive from their library and aggregate them into a machine model, to simulate the operational performance of the machine model as discrete event and to use common data format to achieve the integration. The study concludes that the provision of a library of machine primitives is efficient in constructing a model by using configuration tools; a hierarchical ring and tree data structure is appropriate for the purpose of simulating modular machines; potential machine configurations of a machine can be evaluated by using the machine design and simulation environment.

The main body of the thesis comprises four sections relating the establishment of a library of machine modules, the generation of configuration tools for modelling modular machines, establishing means of simulating/animating machines and finally specifying and using a common data format which can facilitate system integration.

Following a literature review which reviews important literature in the areas of modelling and simulation systems, Chapter 4 discusses the issues of establishing a modular machine library of building primitives including single motion primitives and higher order primitives. Means of establishing different configurations of mechanical mechanisms are also considered. Chapter 5 furthers the discussion of the previous chapter and illustrates application areas of the library primitives created in chapter 4. The creation of a set of supporting tools for modular machine modelling, design and simulation is also described. The main components of these tools comprise modular machine configuration tools for building a machine model within the design environment. Spatial relationships and control logic to enable simulation of different operations are also defined by using these tools. A user friendly interface window is also described in chapter 5.

Chapter 6 considers aspects of the kinematic modelling of modular machines. Two major classes of manufacturing configuration (or axis groups), which will be referred to as articulated and distributed devices, are considered. In addition means of describing and implementing forward and inverse kinematics of these two types of device are described. Means of defining different motion types and associated position, path and velocity information are also illustrated in this chapter.

The kinematic specification of the two device classes and solutions to their inverse kinematics are described in chapter 7 and a discussion of the issues encountered in the simulation of modular machines is presented. The idea of using various simulation mechanisms and processors is introduced in order to cope with complexity when designing and simulating them by catering for demanding manufacturing requirements.

The programming of a modular machine provides a means for an end user to specify the tasks executed by such a machine. Chapter 8 describes a programming approach which involves a three level programming environment leading to the simulated execution of tasks performed by modular machines.

Finally chapter 9 discusses the issues of integrating the design and simulation environment with other computer based systems in the context of computer integrated manufacturing. Thus a common data format approach is proposed and the proof of concept implementation of such a common data format is detailed. Chapter 10 concludes that a design and simulation environment is required for modular machines and such an environment is feasible and beneficial for both machine users and designers. As part of the methodology means of achieving such a design and simulation environment have been devised which build on the use of a robot simulation system (which employs proven computer modelling technology).

Chapter 2 Literature Survey

2.1 Introduction

With increasing product competition world-wide, current manufacturing industry has been challenged by the demand to manufacture in small batches and reduce product engineering life cycle time. As a result modern manufacturing systems need to be responsive in facilitating quick product changes, the production of short lead times and achieving cost effective machine utilisation. Consequently, various types of automation equipment with programmable capability, such as industrial robots, numerically controlled machines, automated guided vehicle (AGVs) and devices controlled by Programmable Logic Controllers (PLCs) are the result of modern technology and market requirements [Huang and Houck 1985, Hasegawa et al. 1990].

The various types of automated machines used in industry are extremely diverse both in their inherent building methods and functionality. Depending on the application requirements and the machine designer's expertise and experience, the approaches or methods adopted by machine designers can be quite different. This diversity in methodology occurs even when resultant machines produced have similar mechanical construction and function properties within their control system. Furthermore, the same manufacturing task may be automated by designing very different machines. One outcome of this diversity of methods and solution is a lack of standards leading to "islands" of manufacturing automation which cannot easily interoperate with other machines, people and software systems in their host environment.

Computer integrated manufacturing (CIM) advances the philosophy that improved productivity and efficiency levels can be realised through the effective utilization of information created at various stages of product realisation [Weston et al. 1988, Edwards 1990, Kusiak and Heragu 1988, and Mahieddine et al. 1990]. CIM systems cover various manufacturing activities in all sectors ranging from planning and design to manufacture of a saleable product [Allen 1987 and Crookall 1987].

In striving to achieve goals which can be realised through using CIM it is necessary to consider the design of machines[1] with a view to facilitating their integration. Furthermore in designing such machines the assistance of a computer aided machine design environment (and hence the availability of design tools which assist the machine designer) are becoming imperative. Contemporary robots and other industrial automation machines are typically designed so that they possibly work with people, tools, fixtures, etc. but only with reference to their local manufacturing environment. In order to enable symbiotic operation of a machine within its environment (with high levels efficiency and as-required flexibility), individuals with expertise in industrial, mechanical, electronic and software engineering are required: it being necessary during design to consider control system functionality, mechanical properties and integration requirements before task programming is realised. This thesis aims to advance the notion that an integrated design environment covering the various design aspects can be realised so as to enable machine performance to be analysed as an aid to design modification. Iteration is necessary to enable the design meet the established requirement particularly as system complexity grows. Using conventional approaches a design iteration may be extremely costly both in terms of extending lead times

1. In this context and indeed throughout this thesis, the term "***machine***" not only implies the single machine but also a possible machine grouping (e.g. into a cell or a production line).

and in constructing ill conceived solutions. As manufacturing tasks become more complex and diverse, the conventional approach becomes even less responsive and efficient. A comprehensive set of design tools which assist the machine designer are much in demand [Jayaraman and Levas 1988, Miller and Lennox 1990].

Modular machines can provide hardware flexibility, high levels of functionality and cost-effectiveness. Such machines can be built from primitive machine building elements both in terms of mechanical component elements and control system elements [Wurst 1986, Weston et al. 1989b, Tesar and Butler 1989]. The concept of modularity in machine design, tool design and software design is not new. It has been used in robotic applications for spot welding [Smith and Cazes 1982], various materials handling applications [Kamm 1983], and inspection [Gleason and Agin 1979]. Conventional industrial automation machines, such as robots and NC machines, have essentially fixed mechanical construction coupled with software flexibility. In contrast modular machines can be designed and used in industry with increased levels of mechanical configurability and system hardware flexibility to cater for changes in application area or products. Due to the feature of physical reconfigurability and inherent optional choices offered, the demand to provide an integrated set of computer-aided design and evaluation tools for modular machines is even more imperative than when establishing the workplace design of many conventional fixed mechanical structure machines. The availability of responsive and powerful tools to aid design can greatly expand the application domains of modular machines and produce higher functionality and more cost-effective flexible automation. The observation that there is a lack of appropriate design support tools to promote the application of modular automation machines is the principle motivation for this research study.

With emphasis on the design of a set of computer aided machine design tools which can support the design and application of modular machines, the literature review studies conventional machine design approaches along with conventional and computer aided modelling and simulation tools. Particular emphasis is on robotic modelling and simulation in the literature study due to prominent recent advances in this area. As a representative of one type of automatic machine, robots have developed quickly in terms of their operational capabilities which have instigated corresponding advances in design and simulation tools. It is appropriate to study evolving methodologies and identify limitations of contemporary robotic simulation systems, as this is instructive in specifying a design and evaluation environment for modular machines. A review of modular machine design is also given to briefly illustrate the state of the art in this area. This chapter concludes with some limitations of using a robot simulation system for modelling modular machines.

2.2 Modular machines and modelling terminology

To date, both the research literature and commercial modelling and simulation systems focus primarily on robots and means of off-line programming [Levas and Jayaraman 1989]. Although Dillman and Huck [1986] have described a general simulation system, Pai and Leu [1986] have proposed an interactive computer graphics simulation system, and Pinson [1985] has postulated a general simulation environment, their work primarily focuses on aspects of simulation and programming for conventional robots with largely fixed mechanical structures. Recently Jayaraman and Levas [1988] have described a workcell application design environment (WADE) aiming at providing an aid for designers to model industrial equipment, but WADE is still at a research development stage and does not support various important requirements for modular machines. The increasing importance of modelling industrial equipment with reference to its working environment has been realized by various researchers [Durr et al. 1989, Milberg at al. 1988 and Duffey et al. 1988], but the author has not found literature on this topic in the area of modular machine modelling and simulation.

As this thesis aims to extend the use of simulation methods (previously applicable only to conventional robots) to provide tools for more general modelling and simulation of automated modular machines and their environmental interactions, it is necessary to define terminology which will facilitate the discussion of machine modelling and simulation. The aim here is to adopt appropriate terms from contemporary robot simulation literature, with additional terms defined to accommodate the modelling of extended objects. Reference is also made to terminology used within the Modular System Research Group in the Department of Manufacturing Engineering at Loughborough University of Technology [Case 1990 and Harrison 1990]. The terminology which will be adopted is defined in the following section with the aim of avoiding subsequent ambiguity.

Automatic Machine: An automatic machine not only implies the single form of automatic machine but also a possible machine grouping e.g. a manufacturing machine cell or a production line configured for a manufacturing task.

Conventional robot or articulated robot arm: An articulated robot or robot arm is a general-purpose, programmable machine which possesses certain authropomorphic or humanlike characteristics. A robot can be reprogrammed to move objects through variable programmed motions for the performance of a variety of tasks. A robot is one type of automatic machine.

Primitives or Modular Machine Elements: Based on a functional decomposition, a machine can be divided into various more basic mechanical and control elements. A minimum functional element at the level of basic operation will be called a machine primitive. The mechanical constituents of a machine primitive can be further classified into two categories based on their elementary functionalities, namely motion primitives (e.g, an axis of motion) or non-motion primitives (e.g, a fixture or gravity feeder).

Device or Functional Device: A functional device is a general constituent of a modular machine, which is constructed from several machine building elements which need to have certain logical and spatial relationships to achieve some manufacturing operation. A modular machine can be comprised of several functional devices. Each of these devices performs a specific operation and collectively they achieve a manufacturing task.

Articulated Device: If a functional device includes a serial kinematic chain linking constituent modular building elements (or primitives), the device is called an articulated device. Therefore the articulated device is a class of functional device. Within such a device, the chained primitive axes have close dependence in terms of ownership and spatial manipulation. This type of device is very useful in achieving the spatial flexibility or dexterity available with conventional robots.

Distributed Dĕvice: A distributed device is another class of functional device for which the constituent elements of the device are configured in such way that some of the modular elements in the device are not mechanically coupled, i.e., there is no close ownership and spatial dependence relationship explicitly established amongst some primitives of the device. This type of device can be used to achieve an operation with particular requirements for cooperation, and/or coordination and/or synchronization.

<u>Modular Programmable Machine or Modular Machine:</u> A modular programmable machine is constructed from modular machine building elements and devices, where both mechanical and control modules can exist within a library of primitives. Such a machine can be configured in a modular manner and the modular elements used can be reused in some other machine building exercise. Programmability implies the flexibility in the resulting automatic machine.

<u>Modular Machine Simulation Environment:</u> A modular machine simulation environment is a computer interactive environment which provides an integrated set of aids to the designer of modular machines, i.e. through integrating tools for the modelling and simulation of multiple modular programmable and non-programmable devices. A hierarchical data structure and modularity are maintained within the environment to enable the extremely diverse range of modular machines application areas to be supported.

<u>Model:</u> The term model will refer to a computer simulation model which represents the machine (or some part of it) in the form of graphics and data structure.

<u>Configuration Tools:</u> The simulation environment for modular machines provides a set of tools which can be viewed as configuration tools to assist a designer in efficiently constructing a modular machine model. Such tools range from those for modular machine design, primitive selection, machine building (more classical use of the term configuration) to verification of the machine model.

<u>Aggregation:</u> Aggregation is the process of constructing a modular machine model through a selection and binding of machine primitives into a modular machine model.

Reconfigurability: Due to the inherent properties of a modular machine and its simulation model, it can be disaggregated and subsequently re-aggregated into another configuration, i.e the machine or model possesses reconfigurability or is reconfigurable.

Task: This is a relatively high level description (in terms of abstraction) of the operations performed by a machine. A task is usually comprised of a number of elemental actions and their temporal and logical relationships which result in the device performing the desired operation. Obviously any specified task should be within the functional capabilities of the particular modular machine which is required to perform the task.

Event: A event refers to a user designed action or I/O requirement (for a device or primitive). The occurrence of events are simulated when performance of the device (or primitive) is evaluated within the modular machine simulation environment.

Concurrency: Since a modular machine usually consists of several devices, it may be a requirement for a number of devices to perform tasks (and operations) at the same time (i.e. concurrently). Based on state relationships and time relationships governing the operation of devices, concurrency can be further categorized into two types, viz, coordination and synchronization.

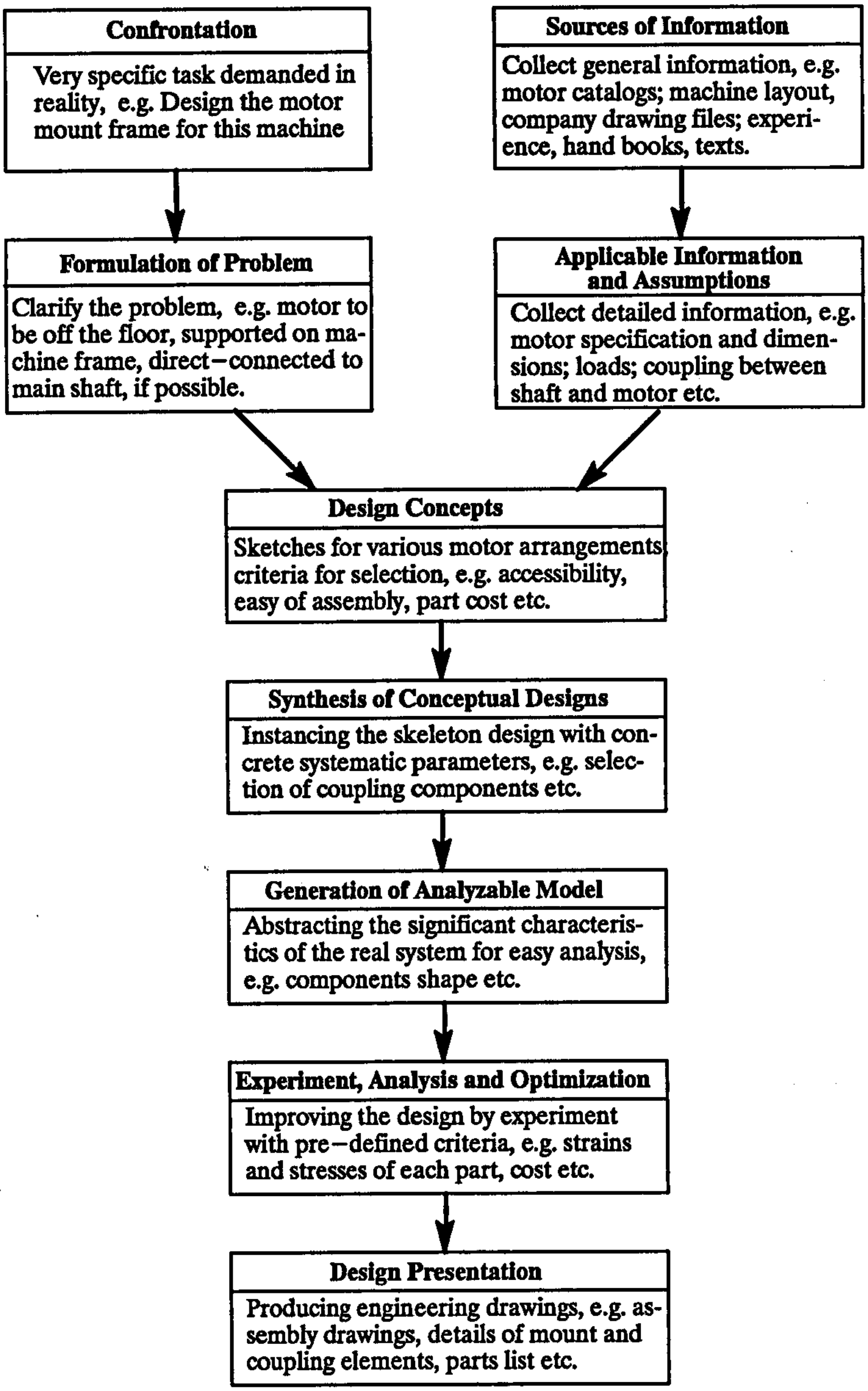
Coordination: Coordination is characterized by the need to establish a sequence of actions or state relationships between two or more concurrently moving primitives (or devices) to meet a set of pre-defined criteria. For example, the arrival of a PCB board may activate an insertion operation which may be achieved through co-ordinating the motion of a PCB transporting device with that of a manipulation device for insertion.

Synchronization: Synchronization refers to the need to establish time relationships between actions (or state changes) of two or more concurrently operating primitives (or devices). For example two devices or primitives may begin and terminate execution of an action at the same instant, thus the actions are performed with synchronization. An alternative example is the need to continuously synchronise the motion of two or more primitives so that their relative motion follows a defined spatial contour. Thus different types of synchronisation (i.e. time dependency) need to be supported (and simulated).

2.3 Conventional approaches to machine design

Machine design processes traditionally demonstrate wide variation. Due to the complex nature of typical design processes, and the different experience and knowledge that a machine designer may have, the proposals and the final designs generated by different engineers are likely to demonstrate significant variation even for the same design problem. In an effort to provide a basis for structuring design processes, Sandor [1964] proposed the systematic approach illustrated in the simplified flowchart of Figure 2.1. Other similar approaches are reported by Taguchi and Wu [1980] and Sandgren [1990]. In summary the steps followed are:

- The first step is for an engineer to attempt to solve a problem with vague information; to solve the problem the engineer typically needs to consult available information of various types.
- The second step is for the engineer to clarify the problem and define the problem precisely for engineering action; thus the engineer applies the available information to the problem.
- The third step is for the engineer to devise some conceptional designs and select one



Confrontation
Very specific task demanded in reality, e.g. Design the motor mount frame for this machine
Sources of Information
Collect general information, e.g. motor catalogs; machine layout, company drawing files; experience, hand books, texts.
Formulation of Problem
Clarify the problem, e.g. motor to be off the floor, supported on machine frame, direct−connected to main shaft, if possible.
Applicable Information and Assumptions
Collect detailed information, e.g. motor specification and dimensions; loads; coupling between shaft and motor etc.
Design Concepts
Sketches for various motor arrangements criteria for selection, e.g. accessibility, easy of assembly, part cost etc.
Synthesis of Conceptual Designs
Instancing the skeleton design with concrete systematic parameters, e.g. selection of coupling components etc.
Generation of Analyzable Model
Abstracting the significant characteristics of the real system for easy analysis, e.g. components shape etc.
Experiment, Analysis and Optimization
Improving the design by experiment with pre−defined criteria, e.g. strains and stresses of each part, cost etc.
Design Presentation
Producing engineering drawings, e.g. assembly drawings, details of mount and coupling elements, parts list etc.

Figure 2.1 Sandor's Y shaped structure of the design process

design based on comparison with criteria.

- The design is synthesized to fill in a skeleton design with correct parameters at step 4.
- In order to analyse the physical system so designed, it is important to generate a model to characterize the physical system at step 5.
- At step 6, the engineer analyses and optimizes the design with reference to the experimental results based on the established criteria.
- Finally the design is presented to the system user and builder, before ultimate delivery to the user.

Traditionally at all stages of the design process, an engineer's intuition and judgement plays a very important role, this being based on experience and knowledge. This potentially increases the possibility of sub-optimal machine design. Although there are some general and typical methods and tools (e.g, finite element methods for structural and flow analyses), there exist few scientifically-based general design strategies and procedures [Sandgren 1990]. In an effort to generate a generally acceptable theory (or set of generalized principles) to guide the machine design processes, Duffey and Dixon [1990] have proposed a taxonomy of design problem types. Figure 2.2 summarizes the six types of mutually exclusive abstracted design problem, viz: "Perceived Need" from customers, required "Function" for the need, the principle of a design for the "Physical Phenomena", "Embodiment" of the design, the "Artifact" (or attribute of) design and finally the (physical) "Artifact Instance".

2.4 Current practice in CAD based machine design and modelling

2.4.1 CAD in general machine and mechanism design and modelling

Computer aided design has been widely used in generating machine design and engineering

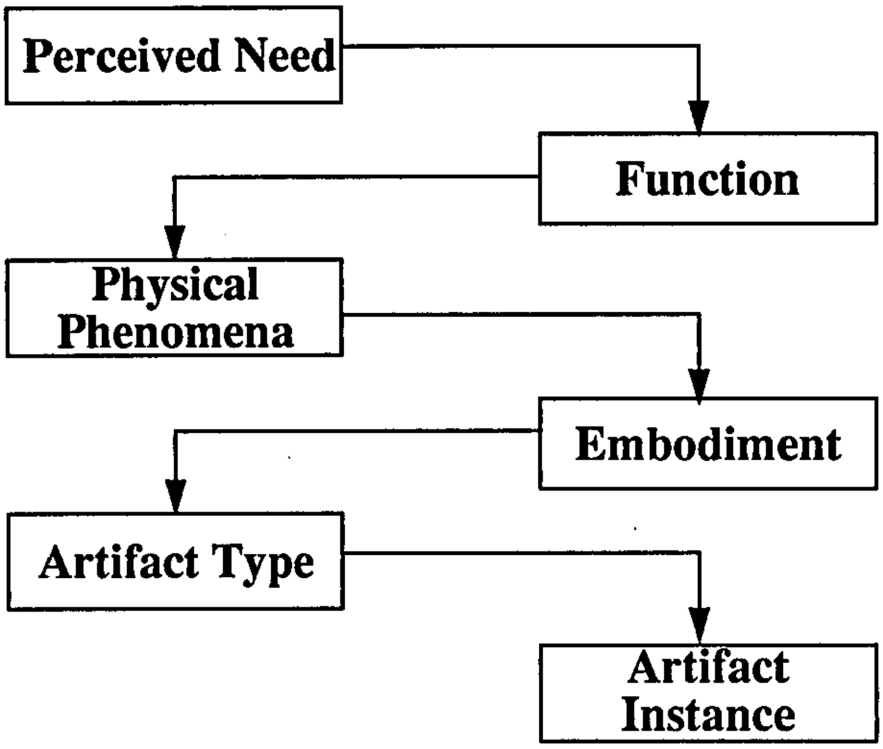


Figure 2.2 Six types of design problems and a design approach

drawings due to its powerful computation capability and enabling graphics. Within each stage of the complete design process CAD based computer packages have played an important role in optimizing the design, reducing the design cycle and improving design quality [Requicha and Voelcker 1982, Han et al. 1990]. For example, computer modelling techniques have been used in stress analysis by utilizing the finite-element method [Zienkiewicz 1977 and Rockey et al. 1975], and in dynamic analysis of spatial linkages [Langrama and Bartel 1975].

Having realized that previous CAD approaches over stressed mechanism analysis rather than synthesis, various researchers have started to look at computer aided design tools for synthesis and simulation of mechanism design optimization. Typically this has involved non-linear programming and has shown some promise [Sandgren 1990]. Sandgren (see

Figure 2.3) has proposed a design tree structure to optimize the design of mechanisms based

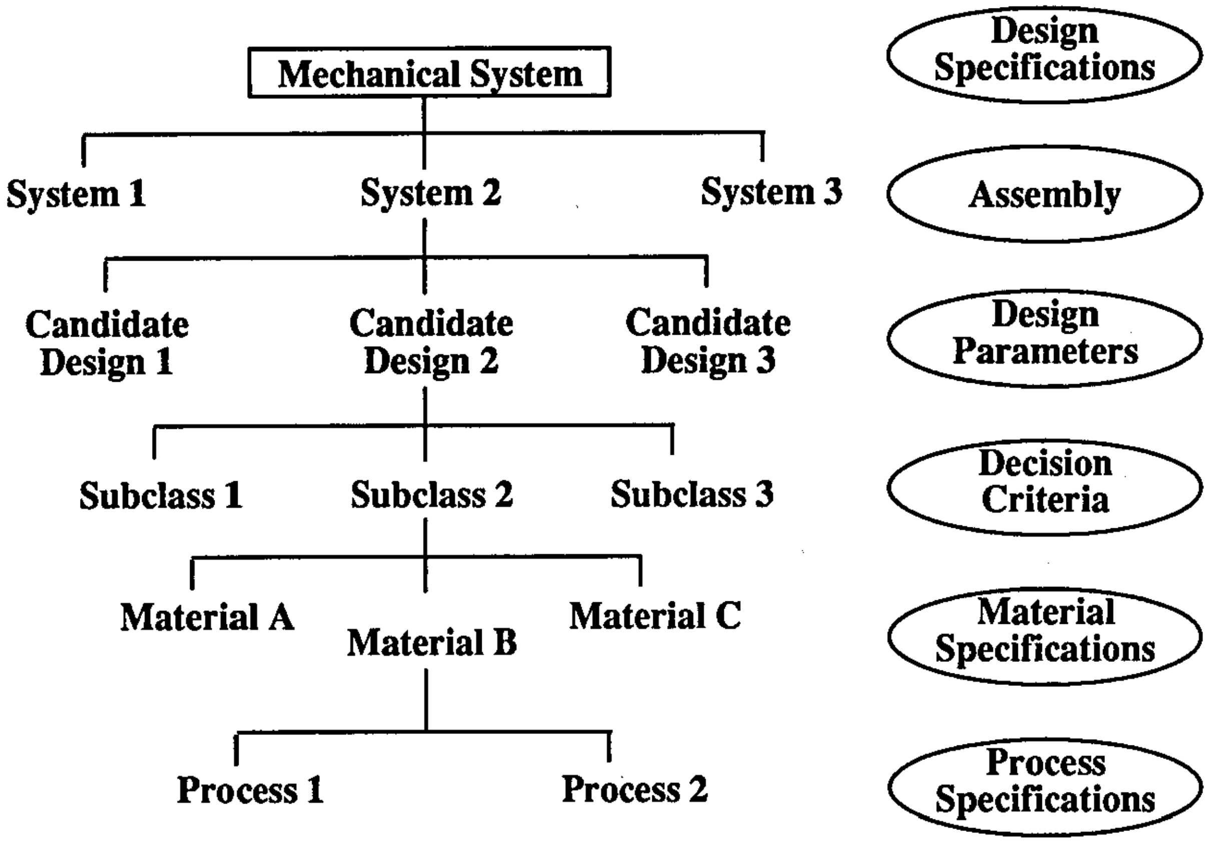


Figure 2.3 Generic design tree structure

on non-linear goal programming which allows consideration of multiple design objectives and handles both hard and soft design specifications. The author reports that a combination of a tree structure and the non-linear goal programming provides a flexible design environment to deal with design optimization problems. Figure 2.4 (a) (a simplistic view of the design process) and (b) (a more realistic view) outline the design tree structure and two types of non-linear programming problem, where outer noises represent uncontrollable variations in design parameters while inner noises are unavoidable but in part controllable variations in the design variables caused by time (e.g. wear) and manufacturing (e.g. tolerances) [Bartel and Marks 1974].

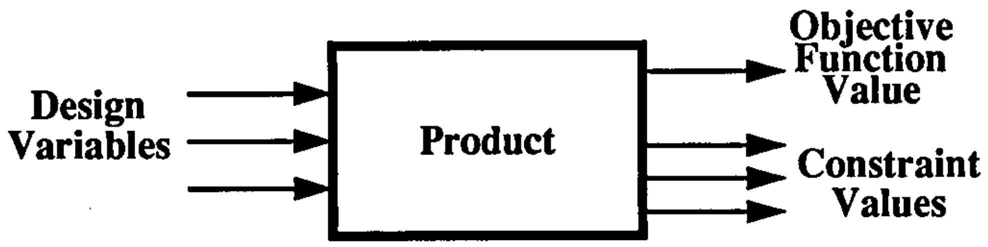


Figure 2.4 (a) Standard non-linear programming problem model

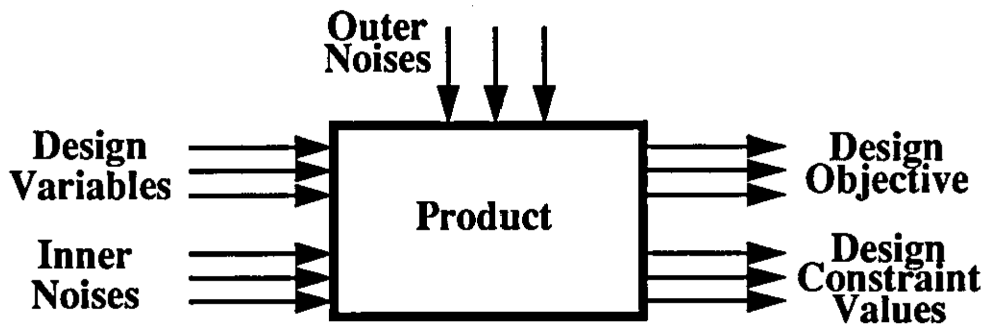


Figure 2.4 (b) Desired non-linear programming problem model

Increasing interest in robot manipulators and three-dimensional biomechanics has greatly advanced the state of the art in mechanism design and synthesis [Fanghella et al. 1989]. Most relevant literature in the field of machine design and modelling with consideration of its environment is found in publications related to robot simulation systems, off-line programming of robots and robot modelling.

2.4.2 Robot simulation and modelling

The sophistication of automatic machinery, e.g., robots, mechatronic mechanisms, and other programmable equipment, has increased substantially partially due to their multi-technological nature and partially because of the high desire for greater flexibility and responsiveness in systems. The levels of complexity faced when designing and installing these machines within their host environment has greatly outstripped the capability of even a team of engineers from various engineering sectors optimising machine design processes. It is time-consuming and inefficient to design install, and maintain automatic equipment without an acceptably accurate pre-evaluation of the system. The increased demand for rapid product change over, equipment utilization and short lead times leads to a need for a flexible simulation environment, e.g. to maximise the utilisation of robots [Yong et al. 1988, Siegler et al. 1987, Van Aken and Van Brussel 1988, Ambler et al. 1982, Larson and Donath 1985 and Heginbotham et al. 1979]. As robot simulation systems evolve they become increasingly comprehensive and user-friendly. In an effort to classify contemporary robot simulation systems, both in terms of available functionality and openness[2]/completeness, the author will use a combination of the comparative criteria used by Dillmann and Huck [1988], and by Chan [1989]. The criteria chosen include the following:

(1) The means and capabilities of achieving the solid modelling of robot geometry; thereby generating a data representation of the geometry of a robot within its working environment;

(2) Available capabilities of workcell development; which deal with model establishment, of devices and other equipment in the robot workcell;

(3) The means and capabilities of achieving kinematic modelling and control; which provide essential data structures and algorithms for manipulating a robot's motion with respect to its working environment;

2. Openness in this context implies the ability to extend (e.g. by adding functional capability) to provide an integrated support environment for machine design.

(4) The ability and means of achieving the modelling of robot dynamics; thereby enabling the user to simulate dynamic characteristics of a robot, e.g. representing inertias of a robot manipulator and motor torque characteristics;

(5) The means and capabilities of achieving robot off-line programming; which facilitates the transfer of robot programs (probably at a relatively high level) from the simulation environment to the robot control system, in an executable form, (i.e. post-processing the off-line robot program and downloading it into a robot controller to control the robot, possibly with consideration of calibrating offset and model errors between simulation and real world systems).

In reality it is not necessary for every robot simulator to possess all of the above features. Depending on the complexity of a particular simulation task, it may be necessary to utilise only some of the above mentioned capabilities, and even within each category the capabilities typically offered and required vary from one system and situation to another. A summary of current available robot simulation systems and their brief functional description is listed in section 2.5.

2.4.3 Robot solid modelling

Solid modelling has been used for the simulation of robots [Requicha 1988, Hornick and Ravani 1986] over the last decade. Various geometric modelling techniques are used in modelling robots and their workplaces. For instance, the Constructive Solid Geometry (CSG) modelling technique is used in some systems, such as the robot simulation system - ROSI developed at the University of Karlsruhe [Dillmann and Huck 1986]. On the other hand, the Boundary Representations (BRep) models are also used within simulation systems, such as the GRASP simulation package developed in the Department of

Production Engineering and Production Management at the University of Nottingham [Dooner 1984]. However, because of the need for computationally intensive capabilities, like collision detection and graphics, Theveneau and Pasquier [1988] suggest that BRep method should be chosen in preference to CSG for the internal representation of geometric data. Kemper [1986] suggests that CSG tree describes only the history of modelling in a tree (with the operators [union, difference and intersection] in the nodes) and the geometric primitive (cube, block, cone, cylinder, etc.) in the leaves, and hence there is no explicit information representing boundary conditions in the model. Since BRep has explicit displayable information about an object whereas CSG has implicit one, the use of BRep can significantly facilitate the applications where a great deal of computation are required for displaying. In the case of modelling a modular machine workcell, like a robot workcell, an intensive computation is demanding due to the requirement of displaying many views of a machine model with reference to its kinematic performance and surely BRep can substantially facilitate such an animation.

The limitations of applying pure geometric modelling methods within engineering domains of solid modelling have been realized by many researchers [Pratt 1984, Shah and Rogers 1988a and 1988b, Requicha 1984, Luby et al. 1986, Vaghul et al. 1985 and Floriani and Bruzzone 1989]. In an effort to overcome these limitations form features (i.e. groups of geometric entities defining attributes of a component's nominal size and shape) have been proposed. Pratt and Wilson [1985] have established the functional requirements needed to support an adequate description of form features in a solid modelling environment.

It has also been realised by several robotic modelling researchers that it is inappropriate to directly use the geometry model to model and simulate a robot and its workcell. From a data base point of view, Kemper and Wallrath [1987] have suggested that all known commercially available CAD/CAM systems for robotic modelling and simulation are based on a customised file system rather than a comprehensive database management system, which leads to difficulties when interfacing one system to other systems. Three representational methods, for describing solid objects, were examined with respect to aspects of importance in the design of databases to support robot modelling. The first representational scheme - known as primitive instancing (which requires the definition of every geometric object as a special instance of a geometric primitive object) is not appropriate in general-purpose robot modelling since it requires the specification and creation of an abundance of different relations, each of these relations consisting of only a small number of tuples.

The second - Constructive Solid Geometry, is widely used representation in existing CAD/ CAM systems partly due to the relative ease with which input can be achieved. However since the CSG tree of a complex object can become very deep, and the displayable geometry is held in an implicit rather than explicit form, the computation of a particular view can be time consuming for an application which requires a quick display, such as robot graphical simulation.

The last representation - Boundary Representation has the advantage of explicit edge representation. This can prove valuable in robot graphical simulation and animation. Kemper and Wallrath also surveyed some of the more recent proposals for object oriented engineering databases to support the retrieval and manipulation of engineering objects. Here two kinds of object orientation are normally distinguished: viz: the structural and the behavioural object orientation. Structurally object-oriented databases provide facilities for mapping complex objects onto a database structure and for retrieving these objects as entities with lack of object manipulation definition [Lorie 1982, Lorie and Plouffe 1983]. The behavioural databases provide object type data manipulation and this approach was originated from programming languages, such as GLIDE [Eastman 1986]. The proposed database systems can be summarised as follows based on the findings reported in four publications [Kemper and Wallrath 1987, Dillmann and Huck 1988, Zaniola 1983, Stonebraker et al. 1983 a].

QUEL as a Datatype [Stonebraker et al. 1983 a], is an extension to the database management system INGRES [Stonebraker et al. 1976]. It provides a very general referencing mechanism to increase the expressive power of the query language and to allow the retrieval of tuples from one or more different relations. However, it has disadvantages of additional insertion complexity when new objects are created. In general terms, it supports structural object orientation via a very general reference mechanism, but the system does not provide any facilities for behavioural object orientation, i.e, the model does not allow the definition of application-specific operations.

ADT-INGRES was proposed by Stonebraker et al. [1983 b] and provides a new way of specifying data types and corresponding operators which can be arbitrarily complex in a database management system. However, it requires highly trained staff (with knowledge of QUEL and the C language), and it does not provide support for handling hierarchical data structures (which are very important in engineering applications). Generally, ADT-INGRES provides some facilities for behavioural object orientation by allowing the user to define application-specific ADT operations. However, these operations are difficult to implement due to their internal non-structured object orientation.

GEM was developed at Bell Laboratories by Zaniola [1983] as a general-purpose query and update language for the entity-relationship data model. It is an extension to the database language QUEL. The general database GEM has the same expressive power as "QUEL, as a Datatype" for robotics. Sets of composite data types need supporting to achieve an improvement for hierarchical structured modelling.

The Complex Object Data Model (an extension to System R) is a relational database management system developed at IBM in San Jose, USA. The concept of a complex object is to define a hierarchical cluster of tuples with different relations for geometry modelling. This supports the structurally object-oriented modelling of hierarchical engineering objects. With enhancements to the query language to support retrieval operations, the System R extension allows retrieval of a complex object as entity even though the object representation may be segmented over different relations. However there still exist doubts concerning the sufficiency of data types for technical applications since only one new data type (long field) is introduced [Lorie and Plouffe 1983].

The Functional Data Model and the language **DAPLEX** were proposed by Shipman [1981] and provide a conceptually natural database interface language through the basic constructs of DAPLEX - the entity which is intended to model real-word objects and the function, defining the object properties. The language DAPLEX has the advantages of supporting the representation of hierarchical relationship (i.e, having structural object modelling orientation), allowing users to derive functions to represent arbitrary relationships and programming in terms of functions (high level programming) rather than database language. The limitations are that DAPLEX does not allow the user to define computationally complex functions and that inserting geometric data into a DAPLEX schema is extremely expensive in terms of computation.

The NF^2 (Non First-Normal Form) **Data Model** was introduced by Schek and Pistor [1982] and is based on the non-normalized relational model. The database prototype AIM-P (Advanced Information Management Prototype) is an implementation of the NF^2, supports composite attribute types and offers easy modelling of hierarchical relationships among objects [Dadam et al. 1986]. Being a hybrid of the relational and the hierarchical data model, the NF^2 suffers from the problem of data redundancy in their representation. However in general, the NF^2 model provides structural object orientation for hierarchically composed objects through clustering complex objects via sub relations. HDBL (Heridelbery DataBase Language) was used to transfer the data definition language of AIM-P into NF^2 scheme [Dillman and Huck 1988], but HDBL does not provide facilities for the definition of application-specific operations.

R^{2}D^2 (Relational Robotics Database System with Extensive Datatypes) was developed at the University of Karlsruhe and is an extension to the DBMS (DataBase Management System) AIM-P [Dadam et al. 1986]. Using a symbiotic approach to object-oriented database system, R^{2}D^2 provides concepts for structural and behavioural object orientation by integrating the concept of abstract data types into the data definition and manipulation language of a structurally object-oriented DBMS. The Database features are inherited from the object-oriented data model NF^2, which allows the modelling of hierarchical relationships among sub-objects; whereas the behavioural object orientation is achieved by defining operations on these defined abstract data types [Dillman and Huck 1988, Kemper and Wallrath 1987].

From the above brief description, it suggests that the CSG representation, with its inherent recursively defined tree structure, can be used in simulating automatic machines (such as robot solid modelling). However the BRep model consisting of an abstraction hierarchy is more promising for this type of modelling. A structural as well as behavioural representation of object orientation is necessary for flexible and effective engineering application.

2.4.4 Workcell modelling

The modelling of machines, can be very complex involving a variety of representations of physical phenomena which need to be modelled with sufficient precision [Theveneau and Pasquier 1988, Ravani 1988, Milberg et al. 1988, Yoshimara et al. 1990]. On considering methods of modelling the general machine (along with fixed objects and sensor primitives within the machine's environment), Ravani proposed what he termed a complete modelling system for robot programming and simulation [Ravani 1988]. This system, called RWORLD (Robot WORLD), uses a multi-primitive representation of the workcell environment, at an abstract level, to cater for the various entity types encountered in real systems. The system includes: device primitives (which have one or more inherent degrees of freedom); object primitives without any inherent degrees of freedom (but which are capable of being manipulated in space); frame primitives (used to mark a location for the primitives); and finally sensor primitives (which provide a functional representation of sensory interactions). He argues that the provision of various types of primitive makes the system more attractive.

Haurat and Perrard [1988] described a robot programming and simulation system known as ADAR. This system is aimed at defining the required application, programming, debugging and simulation tools required for robot applications. It claims to model complete robot workcells, including conveyors, NC machines etc. Theveneau and Pasquier [1988] discussed several representations for robot modelling, both in numerical and symbolical forms. These representations were used for computing and planning: they include representations of geometric planning items and representations of the physical properties and behaviour of objects. The representation of uncertainty in robot workcell modelling was also considered.

More recently, several research teams have realized the necessity and importance of creating a modelling environment which aids a designer of robot workcells [Jayaraman and Levas 1988]. Fougere et al. [1985] presented a description of the process of workcell design by using the ROBOT-SIM system which is a CAD based workcell design and off-line programming system. Miller and Lennox [1990] also described an object-oriented Robot Independent Programming Environment (RIPE) developed at Sandia National Laboratories at New Mexico USA. RIPE provides an environment for complex system integration with emphasis on robotic programming and integration of sensor technologies. With a realization of the need to also simulate the operation of vision systems, Raczkowsky et al. [1988] described a vision simulation system which used CAD system data for the calculation of the geometrical relationships between a camera light source and the workplaces.

Clearly there is a trend and need to create a comprehensive modelling environment to support system designers (when planning and building automated machines) in which the context of the machine (i.e. the application environment) also needs to be modelled.

2.4.5 Robotic kinematic modelling

Robotic manipulators typically comprise articulated, open kinematic chains of N rigid bodies (links) which are connected serially by N joints [Ang Jr. and Tourassis 1987]. The kinematics of this type of robotic manipulator are very well understood. The Denavit-

Hartenberg representation (and associated parameters) has been used extensively to define homogeneous transformations between link coordinate frames [Paul and Zhang 1986, Paul and Rosa 1986]. The forward kinematics of robots can be easily solved using defined link parameters in the D-H equation, and by multiplying link transformations to find a single transformation which gives the goal location in matrix form [Paul 1981, Craig 1989]. However, the robotic inverse kinematics (which addresses the problem of calculating the joint coordinates for a given position and orientation of the end-effector) is complicated by the demand that the desired motion of the end-effector is frequently described in a Cartesian coordinate frame while the joint servos require that their reference inputs be specified in joint coordinates [Fu et al. 1987, Tourassis 1988 and Walker 1988]. All manipulator systems, which comprise revolute and prismatic joints configured in an articulated form with up to six degrees of freedom, are soluble using numerical methods. In the general case, however analytic or closed-form solutions of the inverse kinematic problem do not always exist for six degree of freedom robots. A principle of decoupling has been extensively used to derive closed-form solutions for robots [Vassilios and Ang.Jr 1989b]. Here a robot is generally decomposed into two groups or subsets of joint, each group mainly accounting for either position or orientation changes respectively.

In his pioneering work, Pieper [1968] showed that for six degree of freedom manipulators in which three consecutive axes intersect at a point, a closed form or analytical solution can be obtained for the configuration. The majority of commercially available industrial robots today are designed to satisfy this requirement. Gupta [1986 and 1988] extended the concept of decoupling by using a Zero Reference Position description for kinematic analysis. For other robots, which do not satisfy Pieper's condition, a closed-form solution may not exist

[Rieseler and Wahl 1990] and in such cases researchers have resorted to the use of numerical and iterative methods to solve the inverse kinematic problem [Gupta and Kazerounian 1985, Goldenberg et al. 1985, Vassilios and Ang.Jr 1989b]. Vassilios and Ang.Jr have suggested that the most common numerical methods used for this purpose are based upon the Newton-Raphson approach. However these numerical methods are deficient in that convergence to the correct solution is never guaranteed and no multiple solution can be made for a manipulator system. Therefore general-purpose inverse kinematic methods involving numerical methods are still not applicable in a practical sense.

Takano [1985] developed a set of analytic solutions to compute sub-sets of joint coordinates given their respective positioning and orientating sub-tasks. Eight different configurations of three degree of freedom robot manipulator system were discussed and up to eight solutions were provided for a given position and orientation, although arbitrary 6 axis manipulators can have a maximum of sixteen solutions [Primrose 1986]. However the algorithm requires forward kinematic computation, and the mathematical basis for the algorithm and convergence conditions were not discussed in the paper.

Tourassis and Ang Jr. [1989 a and 1989 b] proposed a modular architecture for inverse robot kinematics to overcome the current numerical deficiency in inverse kinematic computation. They claimed to have developed a general-purpose and mathematically robust algorithm for inverse kinematic solution, which can handle both closed-form analytic and numerical situations. The proposed algorithm facilitates a mathematical definition of a region in the robot workspace where convergence to the correct solution is guaranteed. The algorithm is also insensitive to the initial estimates and it provides for the computation of multiple solutions. However the algorithm is still very much a numerical computation based method and requires considerable iteration before a solution is derived.

2.4.6 Robot dynamic modelling

The dynamics of physical manipulators is concerned with the relationship between forces and motions in two respects; namely forward dynamics (being the calculation of the accelerations which are then integrated to obtain the manipulators' coordinates and velocities in response to the applied forces or torques), and inverse dynamics (being the calculation of the forces or torques required for manipulators to achieve the desired generalized coordinates and their velocities and acceleration) [Featherstone 1987]. There are two approaches towards obtaining a dynamic model of a motion manipulator from well known physical laws, viz: Newton-Euler approach and Lagrange's-Euler approach.

The Newton-Euler approach is based on the Newton's second law of motion,

$$F = m \times \dot{V}_v$$

where m is the total mass of a manipulator moving part, F is the force to be exerted on the moving part and the $\dot{V}_v$ is the acceleration of the moving part. Whereas Euler's equation

$$N = I_c \times \dot{\omega} + \omega \times I_c \omega$$

where Ic is the inertia tensor of the moving part in frame {c}, where origin is located at the centre of the moving part;

ω is the moving part angular velocity and,

$\dot{\omega}$ is its angular acceleration.

N is the moment applied on the moving part to cause motion.

With the force F_i to cause linear motion and torque N_i to cause rotation, the joint force and torque can be obtained by establishing force and torque balance relationship on the manipulator.

The Lagrange-Euler approach is an energy-based approach to dynamics of manipulators [Uicker 1965]. The Lagrangian of a manipulator is expressed as

$$L(q_i, \dot{q}_i) = K(q_i, \dot{q}_i) - P(\dot{q}_i)$$

where

L: Lagrangian function

K: total kinetic energy of the manipulator chain,

P: total potential energy of the manipulator chain,

q_i: generalized coordinates of a manipulator,

$\dot{q}_i$: first time derivative (velocity) of the generalized coordinate,

The Lagrange-Euler equation of motion for the manipulator is given by

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = T_i \qquad i = 1, 2, \ldots n$$

where T_i is generalized force or torque applied to the manipulator. Both of the above approaches can result in a set of non-recursive equations of motion in the form [Featherstone 1987 and Turney et al. 1980]

$$F_i = \sum_{j=1}^{n} H_{ij}\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\dot{q}\dot{q}_k + g_i$$

or more meaningfully

$$F(t) = H(q(t))\ddot{q}(t) + h(q(t), \dot{q}(t)) + g(q(t))$$

where F, q and $\dot{q}$ are the joint vector force, velocities and acceleration, and H, h and g are inertial, Coriolis and gravitational coefficients. For complex articulated mechanisms, the recursive Newton-Euler formulation applied and developed by Luh et al. [1980, Armstrong 1979 and Orin et al. 1979] is considered to be the most efficient currently known general method for calculating inverse dynamics. However, a number of specialised methods have shown that for any given manipulator customized closed form (symbolic structure for better insight as shown above) dynamics are more efficient than the recursive

Newton-Euler scheme [Armstrong et al. 1986 Kane et al. 1983 and Izaguirre et al. 1985]. These customized closed form equations are extremely attractive for the dynamic modelling and simulation of modular machines as often the use of multiple instances of less complex devices is the main concern of modular machine designer.

The above discussion has illustrated the feasibility of modular machine dynamic modelling and suggests a way forward. However, since the modelling of machine dynamics requires an intimate knowledge of the dynamics of physical machines and because of time limitations and difficulties in obtaining or establishing information of this type, it was decided that dynamic modelling could not be within the scope of this research. However in Chapter 7, the drive and control aspects of modular machines at the simulation phase are outlined.

2.4.7 Robot off-line programming

Although the commercial availability of robot programming languages is an important advance in the evolution of industrial robots, currently available languages are still difficult to use and essentially robots are still considered to work as a stand-alone automated devices [Gini 1987, Lyons and Arbib 1989 and Van Brussel et al. 1987]. Common difficulties encountered when using robots in industry are as follows: a loss of production time when using a robot to teach a program, programming in an uncertain environment making the robot program error-prone, and poor utilization of product information in the context of computer integrated manufacturing (CIM), [Chan et al. 1988, Simkens et al. 1988 and Craig 1988]. Thus many robotic researchers have concentrated on providing user-friendly robot off-line programming systems [Laugier and Pertin 1984, Leu 1985, Tan and Chang 1985, Brantmark and Ramstrom 1985, Wozniak and Warczynski 1988, Imam and Pavis

1988, Ravani and Hornick 1988, Van Aken and Van Brussel 1987, Kitajima 1988, Ball and Smith 1988]. In contrast to robot on-line programming, robot off-line programming and simulation systems offer the following advantages:

(1) There is no need to keep a robot idle whilst generating robot control programs. In some cases only final tuning need be done with the real robot and its workcell;

(2) It is possible to create robot programs with reference to information established when modelling and simulation of the workcell. Knowledge accessed via workcell models can improve robot programs created e.g. avoid collisions;

(3) Access to CAD databases can be achieved to advance the robot simulation and facilitate improved robot programs. This can be achieved through integration of the simulation system and a product design CAD system. This information sharing of product data facilitates a level of correspondence between design and manufacturing phases; e.g. the dimensional definition of a part from a CAD database can be used in determining robot position and sequences [Duelen et al. 1987, Chan 1989, and Dillmann 1987];

(4) It also has financial attraction from the viewpoints of improving the robot workcell layout, reducing set-up time and cost, minimizing the equipment purchase cost based on an evaluation of the robots capabilities to perform a task. The cycle time associated with robot task execution can also be optimized by modifying the positions of parts and the sequences of robot operations.

(5) A structured programming system can be devised for off-line programming, building on the possibility of integrating the simulation system with other CAD and Artificial Intelligence (AI) systems [Rogers et al. 1988]. Thus high levels of abstraction associated with task descriptions can be enabled.

While off-line programming and simulation systems can demonstrate the above advantages, they also suffer disadvantages associated with the inability to accurately model the physical robot workcell. However, various errors caused by inadequate modelling, tolerances related to geometrical descriptions and the specification of spatial relationships between objects, can at least in part be compensated for by employing an appropriate calibration [Ishii et al. 1987, Ravani 1988, Chan 1989]. Another shortcoming of current robot simulators is that they almost invariably have been designed as a stand-alone robot programming package with their own data format and coding language and this leads to significant difficulty when attempting integration [Weck and Clemens 1988, Rui et al. 1988]. Standardization of robot programming has been attempted by various researchers [Arai et al. 1985, Weck and Clemens 1988] and Weck and Clemens described an approach which uses the IRDATA (Industrial Robot DATA) interface to transfer robot programs to robot controllers in an attempt to enable the universal application of off-line programming systems.

Robot programming itself has attracted wide attention in the robot research field [Volz 1988, Van Aken and Van Brussel 1987, and Rock 1988]. It has been realized and accepted by many researchers that contemporary robot languages can correspond to a number of hierarchical levels, i.e the language itself can be used to plan and control robot operation at various levels of abstraction [Bonner and Shin 1983, Lozano-Perez 1983, Leu 1985, Rock 1988 and Volz 1988]. At the NATO workshop in Italy in 1986, a working group proposed a complete structural hierarchy for robot programming languages, as shown in Figure 2.5. In addition to the then three frequently mentioned programming levels - joint, manipulator and task level, two new levels - feature and device levels were added, representing the need

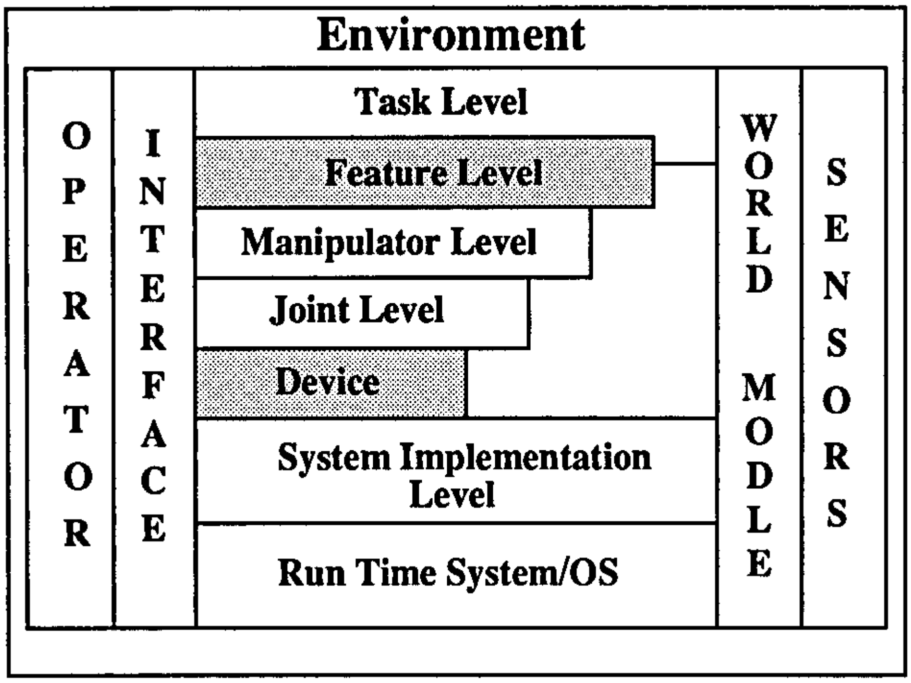


Figure 2.5 Hierarchical structure of robot programming languages

to integrate the design of a product with its manufacturing processes. The programming environment was also highlighted as future robot programming systems will be interfaced to various modelling systems and as they considered it crucial within the design environment to facilitate interfacing between robot programming languages and intelligent knowledge-based systems. [Volz 1988]

2.4.8 The post-processing of robot programs created during simulation

Many simulation systems represent robot programs in their own off-line programming language this often being different to the robot control programming language of the robot simulated and typically specific to a simulation system. Hence post-processing is commonly adopted to convert off-line robot programs created during simulation into real-time physical robot control programs which comply with the target robot language. There were 89 high level robot languages identified by Blaha et al. [1988], nineteen of which robot languages were listed as currently available commercially. On the simulation and CAD/CAM side, there are almost 300 CAD/CAM systems existing to support the manufacturing process at its various life cycle stages [Kemper 1986, Kemper and Wallrath 1987, and Durr et al. 1989]. Among them, some 20 robot simulation and off-line programming systems have been identified by the author. See section 2.5 for a brief description of each system.

Currently most post-processing systems operate on the principle that the system abstracts geometrical information concerning modelled objects, spatial information concerning motions and sequential information in regard to the sense of operations generated at modelling and simulation stages by a user, and transforms them into a particular language format which is required by the target robot, with the consideration of target language capability and restrictions [Craig 1988]. However as earlier discussed, it also important to consider the need for standardised robot programming [Arai et al. 1985] and of interfacing to various manufacturing systems [Weck and Clemens 1988].

2.5 Features of contemporary robot simulation systems

Research into robot simulation systems has received wide attention and many systems have been described in the literature [Dombre et al. 1984, Craig 1985], with many commercially available systems having been used in industry [Yong et al. 1988, Fernandez 1988, and Woodwark 1988]. The author has made a survey of the main features of available simulation systems which are detailed in Appendix A.

2.6 Modular machine design and its requirement for a simulation environment

The concept of modularity is not new in engineering. For example the modularity of personal computers is now an accepted and necessary reality of computer architecture. Unfortunately, to date the use of modularity in robotics has been pursued only at the most elementary level although several mechanism and robotic researchers have realized the importance of the hardware flexibility which can be achieved through adopting a modular mechanical structure [Weston et al. 1989a, Benhabib et al. 1990, Moore et al. 1983, Fukuda and Kawauchi 1990, Ang. Jr and Tourassis 1988, Schmitz et al. 1988, Moore 1986, Rajan and Nof 1990, Rogers and Weston 1990]. Flexible manufacturing automation can be realised through using computer controlled manufacturing machines. Here software flexibility can result in each manufacturing machine being re-programmable. Hence the machine's task can be changed readily although clearly the magnitude of these changes will depend on limitations of the mechanical structure of these machines. Very often the flexibility realised can be further limited, e.g. by the use of specific tools and sensors. Benhabib et al. [1990] suggest that modular robots can offer greater hardware flexibility over and above that of current generation automatic machines.

The concept of modular robot design is derived from the requirement for more flexible manufacturing machines and is achieved by modularity and reconfigurability in the machine's mechanical hardware. Conventional industrial robots have an essentially fixed configuration although optional degrees of freedom may be available as might size variants. Typically a given robot will be best suited to meet the requirements of a particular set of tasks. Hence for a different set of tasks, a solution close to the optimal one (or indeed an acceptable solution) may not be feasible without changing the manipulators. On the other hand, robots are intended to be general purpose machines and hence include redundant functionality. This redundancy can be expensive and leads to difficulties in investment justification [Moore et al. 1990, Kota and Chuenchom 1990]. Alternatively modular robots (and more generally modular machines) can be designed to more fully meet a given set of application requirements, potentially leading to reduced cycle times, improved accuracy and reduced cost. In addition modular machines can demonstrate sufficient flexibility to (i) automate manufacturing tasks for a range of products and (ii) enable re-configuration as required at some future date.

Current research in the area of modular machines focuses on two aspects, namely control architecture design and the provision of configuration tools to simplify the creation of modular machines. Weston et al. [1989a and 1989b]have proposed a control architecture for modular machines which is known as a Universal Machine Control (UMC) architecture. First generation implementations of this architecture has been commercialized, based on the research of the Modular Systems (MS) research group at Loughborough University of Technology. This architecture is used in conjunction with a set of tools to select and configure control system modules dependent of requirements of a given application. However to date the mechanical design of modular robots and machines is at the conceptual design stage and is almost exclusively achieved using traditional approaches due to the lack of computer aided design and simulation tools to aid the design process. Tesar and Butler [1989] presented six undriven joint modules (based on six elementary motion pairs) and three actuator modules. Higher order degree of freedom (DOF) articulated joint modules were also described based on various combinations of these nine basic modules. These included six one DOF actuated joint modules, six two DOF actuated knuckles, four two-DOF parallel planar motion modules, 3 three-DOF planar motion modules, and 3 three-DOF spherical motion modules. However each of these designs are still at a conceptual stage, i.e. methods of selecting, aggregating and evaluating the use of the modules do not yet exist. Clearly there is a need to create and study the use of design and simulation tools which provide a flexible design environment for modular machines.

2.7 The limitations of current robot simulation systems for modular machine modelling

Current robot simulation systems can be used in evaluating and advancing robot performance and off-line programming, but they demonstrate some limitations in the design of modular devices.

Lack of flexibility when designing and configuring a new modular robot configuration is the first limitation. Since most robot simulation systems were designed for commercially available robots, they expect to model a fixed "hardware" structure. This limits the available structural flexibility of the resulting robot model. Clearly it is not the intention of most robot simulation systems to accommodate the simulation of modular structures, hence a new approach to modular machine modelling is required.

From the viewpoint of requiring an integrated design and simulation environment, current robot simulation systems also demonstrate limitations for modelling robot related devices within its workcell. As the demands for shorter product lead-times increases, the need to integrate manufacturing devices and processes becomes more pressing. Since most robot simulation systems were designed to simulate a single, stand-alone robot workcell, insufficient attention has been paid to modelling interactions with other automated devices and therefore the efficient control and programming over these non-robot automated devices are not provided at all. It has been generally realised that design and development of a generic manipulator structure can be very expensive, time-consuming and challenging task in terms of resources and this was especially true in the space industry [Spar Corporation 1975, and Tesar and Butler 1989]. However current robot simulation systems focus primarily upon the robot simulation and off-line programming. Even the modelling aspect is centred mainly on the geometry and kinematics of conventional industrial robots. The lack of efficient analysis tools to evaluate the performance of a complex manufacturing workcell has demonstrated the incapability in modelling wide range devices and aiding workcell designers to develop a new design efficiently. The need to derive such a design and simulation environment for modular machine designers especially with appraisal for modular machine is a major motivation of this research.

Currently most robot simulation systems only provide a programming facility for robots which are defined in an associated robot model library. Any other devices with motion capability are not included in the robot programming facility. Since modules can be aggregated to create a kinematic structure which demonstrates articulated motion (in a similar manner to conventional robots), or alternatively be configured to create a physically distributed structure within a machine, new methods of programming such devices need to be studied. The incapability and insufficiency in multi-device programming of contemporary robot simulation systems also needs improvement.

The limitations of current robot simulators as tools for aiding the design of modular machines has motivated this study in the aim to provide an environment for modular machine design and simulation. Although a proven robot simulator can be used as a building block of a modular machine design and simulation environment, it is necessary to derive and consider the complementary use of various other design methods and tools. The configuration study, kinematics, supporting tools, programming and post-processing under such environment are discussed in this thesis.

Chapter 3 UMC Architecture and an Approach to Modular Machine Simulation

3.1 Introduction

This chapter continues the discussion on existing and potential methodologies which can be used in the design of modular machines. The Universal Machine Control (UMC) architecture is outlined as an example of a particular reference architecture in order to highlight and demonstrate the design principles of modular machines. The modular methods, configuration methods, hierarchical structure methods and data-driven methods used in the design of modular machines are particularly generalised. An approach for deriving an environment for enhancing modular machine design and simulation is proposed based on the UMC approach and modular machine design methodology. Since this study is based on a commercial robot simulation system called GRASP, a detailed description of GRASP is included to illustrate its underlying simulation approach.

3.2 Modular machine design methodology

Due to the potential advantages of a low level of investment in a long term (because of their reconfigurable features), high level of flexibility in their configuration and ease of integration with other machines, modular machines have recently attracted many researchers' attention [Benhabib et al. 1990, Tesar and Butler 1989, Fukuda and Kawauch 1990]. From the total design perspective of a modular machine, the author summarised four methods used in modular machine design, namely: modular method; configuration method; hierarchical method; and data-driven methods. Each of them is detailed in the following sections.

3.2.1 Modular methods in the context of modular machine design

The axiom on which the modular method is based is as follows. It is believed that there are similarities between the functional properties of very many manufacturing machines [Muck and Mammem 1984, Kamm 1983, Tesar and Butler 1989, Weston et al. 1989a, and Yan et al. 1990]. Hence, the functionality of a specific machine can be decomposed into a number of more basic modular elements. These elements can also be considered to belong to a larger family of modules for building machines. Though the decomposed machine modules will vary slightly from one application to the other, a generalization of these modules is possible. This fact has been appreciated by a number of suppliers of industrial machines leading to proprietary families of modular building components of machines [Moore 1986].

Essentially the modularization of manufacturing machines can take one of three forms. Mechanical modularization of the generic manufacturing machine would lead to a decomposition into mechanical elements. For example these elements might be single degree of freedom modular units used to facilitate different types of motion. This level of modularization and decomposition of machine elements can be used by machine designers in constructing machines from well proven building blocks.

Control system modularization results in a family of machine control modules (or building elements), which typically may comprise software and hardware modules. Currently, there are problems associated with contemporary machine control methods which limit their use in controlling modular machines. One of the problems is that these methods intend to provide "hard-wired" solutions which are based on a specific control hardware, and this inevitably results in inflexibility [Doyle and Case 1991]. A generalised solution to this problem, which can form a part of an overall modular machine design strategy, is to create an open (or "standardised") industrial control architecture and to design appropriate control system elements which fit into that architecture. In this architecture "standardised" interfaces exist to proprietary control system elements, thereby enabling those elements to be treated in a generic manner. In this way a generalised approach to modularising high level control hardware and software is possible. Thus using this approach interaction between lower level modules can be sequenced, synchronized, monitored, programmed and generally managed in a consistent manner. Once a vendor specific section of control hardware has been standardised, the hardware and software can then be viewed as a virtual control device and included in the library (or family) of control modules.

A functional modularization of a machine can lead to the delineation and specification of various high level machine modules which themselves may be related in terms of their position and function within a hierarchical structure. Typically these modules will comprise several mechanical and control machine elements. The constituent physical (mechanical and control) elements of a functional module can be physically linked together or closely coupled with physical links (such as mechanical or electro-mechanical connections) between the physical elements. A typical example of a closely coupled functional module is a manipulation / placement manipulator system constructed by physically connecting motion axes and a gripper, which is commonly found in automated assembly operations. On the other hand, the constituent parts of a functional module may be physically decoupled and only be logically linked in some way, e.g. the individual physical components of a functional module may be distributed at various locations along a production line, and their relative motions may be maintained according to some application dependent relationship stored within the control elements or embodied in a task program instead of through mechanical linkages. In this case, physically decoupled machine elements operate in a coordinated fashion (e.g. two or three elements maintain synchronous motion implying a logical link rather than a physical one). It is this logic relationship that can group machine elements within a functional module. These examples quoted can typically be found in packaging and process industries.

In the process of machine design, a modular approach can thus be adopted through aggregating (or combining) modules in a consistent and structured manner, i.e. according to a modular framework for machine design. Such a design method may not only include the selection and building of machine elements, but can also provide a means of evaluating alternative solutions and generally enhancing approaches used in the various stages of machine configuration, implementation and reconfiguration.

3.2.2 Configuration methods in the context of modular machine design

Modular machines can be characterised by their inherent properties of configurability and re-configurability, i.e. a modular machine (built by selecting modules and aggregating them into a machine) can be reconfigured into another form (or configuration) to suit the requirements of a new task or tasks. Therefore configuration tools (which may comprise a set of software tools) can be used to aid the selection of some machine modules and aggregate them into a machine according to some structured framework. At the design stage the configuration tools can be used to build a machine model, through the selection and linking together of machine physical and control modules. This will be referred as to a logical machine and will exist as a software representation of a subset of the general manufacturing machine. Ideally this logical machine (or machine model) should be an open structure with enough functionality to accomplish the required tasks. Through being open in this sense, it can be expanded for future tasks simply by appending new machine elements or modules. At run-time, all machine elements will need to perform their individual tasks which may be defined at a low level (as an individual module) or at a higher level (as a group of low level machine elements or modules). Correspondingly, control hardware and software modules are required to accomplish the tasks. The configuration tools can be considered to be one of two types, viz: (i) for modelling and simulating of mechanical and operational aspects of machines and (ii) for constructing aspects of physical machines and configuring aspects of their required run-time control software. Due to the need to configure a physical machine, there should be hardware configuration tools for both mechanical and control hardware construction. Mechanical hardware configuration tools include standard mechanical connectors between mechanical modules, base support frames, and connector fasteners. Control hardware configuration is determined by the controller design, modular computer structure, flexible internal wiring harness, standard interfaces between control board modules etc. Currently, less attention is directed to the hardware configuration tools and more configuration tools both for mechanical and control are being derived [Benhabib et al. 1990, Karlen et al. 1990]. The configuration tools used to model and simulate mechanical and operational features can be derived to aid the design and evaluation of modular machines.

3.2.3 Hierarchical methods applied to modular machines

It is believed that a physical and functional hierarchy typically exists within a machine [Albus et al. 1981, Kusiak et al. 1988, Jones and Saleh 1990, Jafari 1990] and machine elements existing in this hierarchy can be further decomposed into physical sub-systems which may comprise a number of well defined modules. Therefore modular and hierarchical structures naturally exist within a machine. At each level of the machine hierarchical tree, a machine module is attached to the hierarchy. The adoption of a modular and hierarchical structure can lead to creation of both manageable and flexible machines.

A typical example is the decomposition of an industrial robot. Figure 3.1 shows one approach to such a decomposition. More generally, a machine system can be decomposed into several sub-systems and each sub-system can consist of several modules (see Figure 3.2). Clearly, a hierarchical machine structure can often be established by appropriate decomposition of a machine.

The decomposition of a machine can be achieved in terms of its functionality which will usually be an abstract representation of the purpose of a machine and its constituent parts. Through functional decomposition it is possible to produce a family of control software and hardware modules which can be re-used: a major advantage compared to custom design and build approaches. Considering the functional decomposition of a robot, one possible classification is into three main parts, namely its manipulator, its control functions and sensory capabilities. Each of these functional parts can be further decomposed into even more elemental functional units which exist as a functional hierarchy to form the general purpose machine or robot (see Figure 3.1).

As a general method of designing machines with hierarchical features, hierarchical methods can be employed in modular machine design in terms of both constructing physical machines and modelling and simulating modular machines. Since modular machines can have both an articulated and distributed configuration, the hierarchical

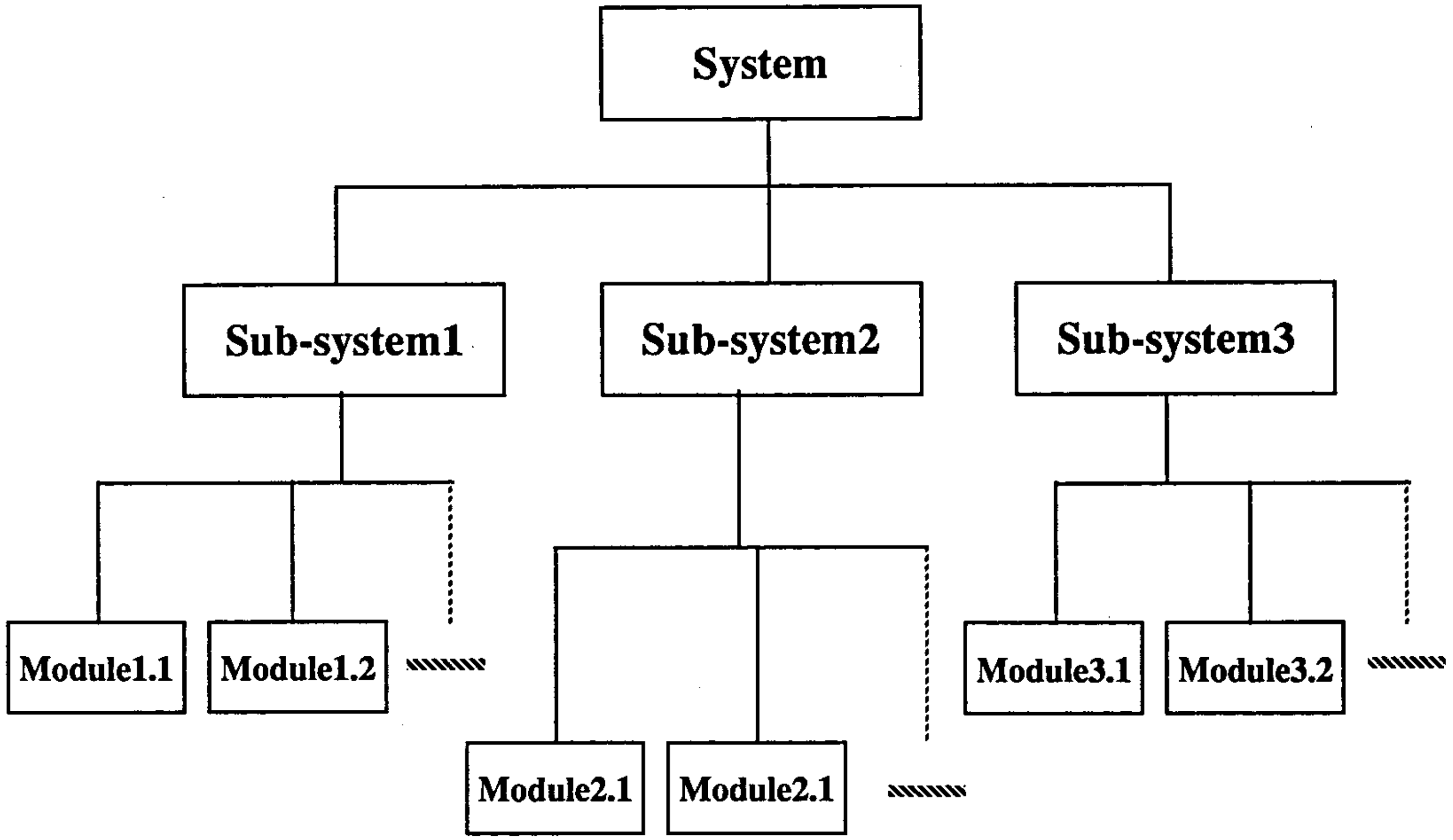


Figure 3.2 The decomposition of a general machine

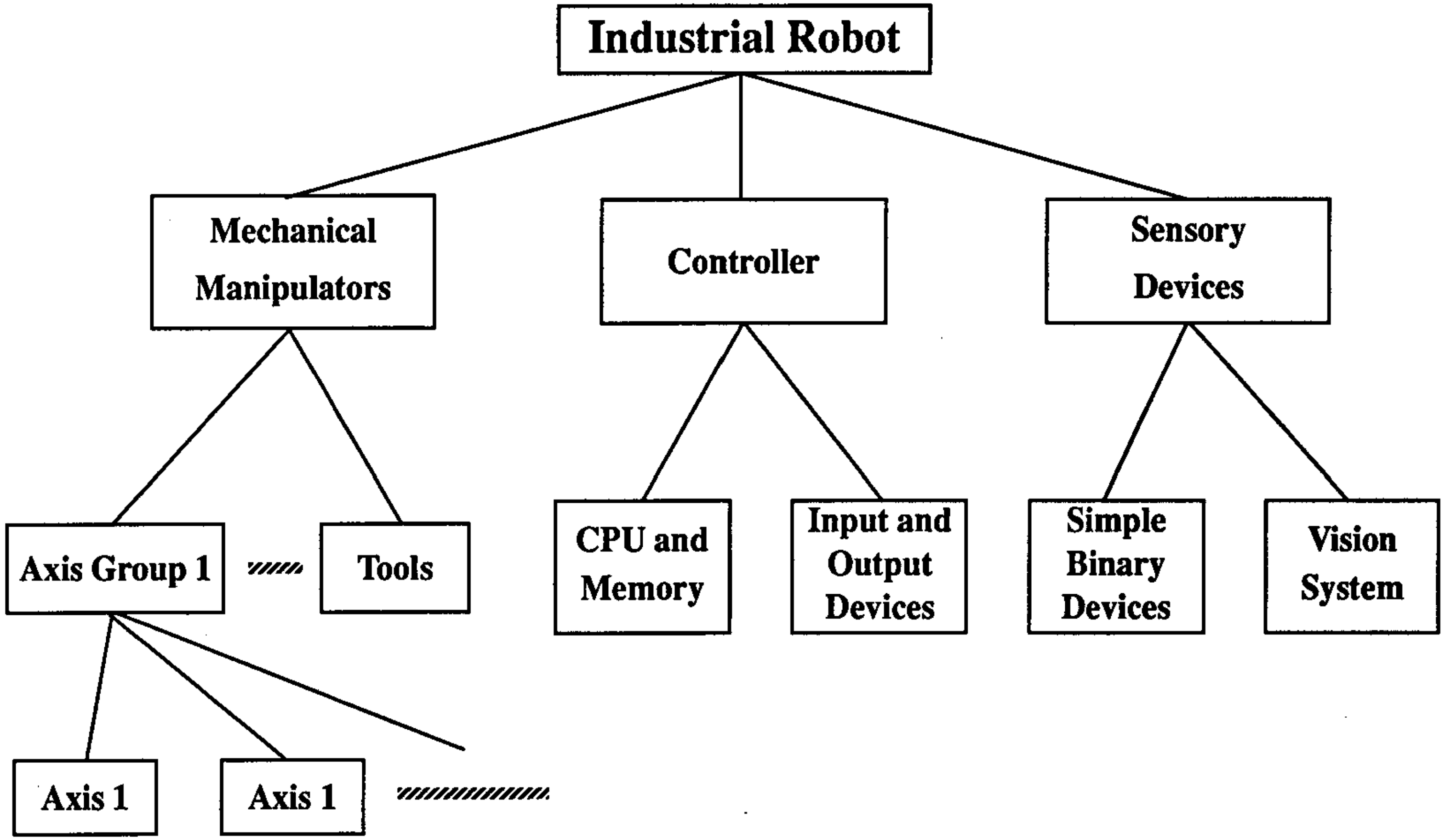


Figure 3.1 The decomposition of an industrial robot

method employed in conventional robot design in a "bottom-up" manner can still be used to model modular machines. A "top-down" hierarchical approach is also needed to design and evaluate a modular machine from the viewpoint of its functionality.

3.2.4 Data driven method in the context of modular machine design

One of the axioms on which modular machine design is based is that any given aggregation of modules (which forms a manufacturing machine) can be described by a data model defining the functions of modules and their interaction [Moore et al. 1990, Durr et al. 1989, Harrison 1991]. A data model can be used to represent various aspects of a modular machine throughout its life-cycle. It can be a kinematic description of a machine element or a description of its motion. All data modules associated with a modular machine can be abstracted from any given manufacturing machine by adopting the underlying concepts of a modular and hierarchical approach. Thus data modules can describe a range of machine characteristics in relation to single machine element or indeed to the structure of the whole machine. Therefore, data modules in the context of modular machine can be the specifications of the machine's construction and functionality.

The data modules representing a modular machine can be classified into a number of types. Figure 3.3 shows data modules for a particular modular approach - the UMC reference architecture. They include an I/O Component Device Descriptor data module, Physical Primitive module, Positional data modules, Task data modules and Machine data module etc. Collectively, all data modules can exist within a defined hierarchical structure. Once a reference data module is required, such as a positional data module, the top level data module is passed onto lower levels with the machine eventually being driven by these data

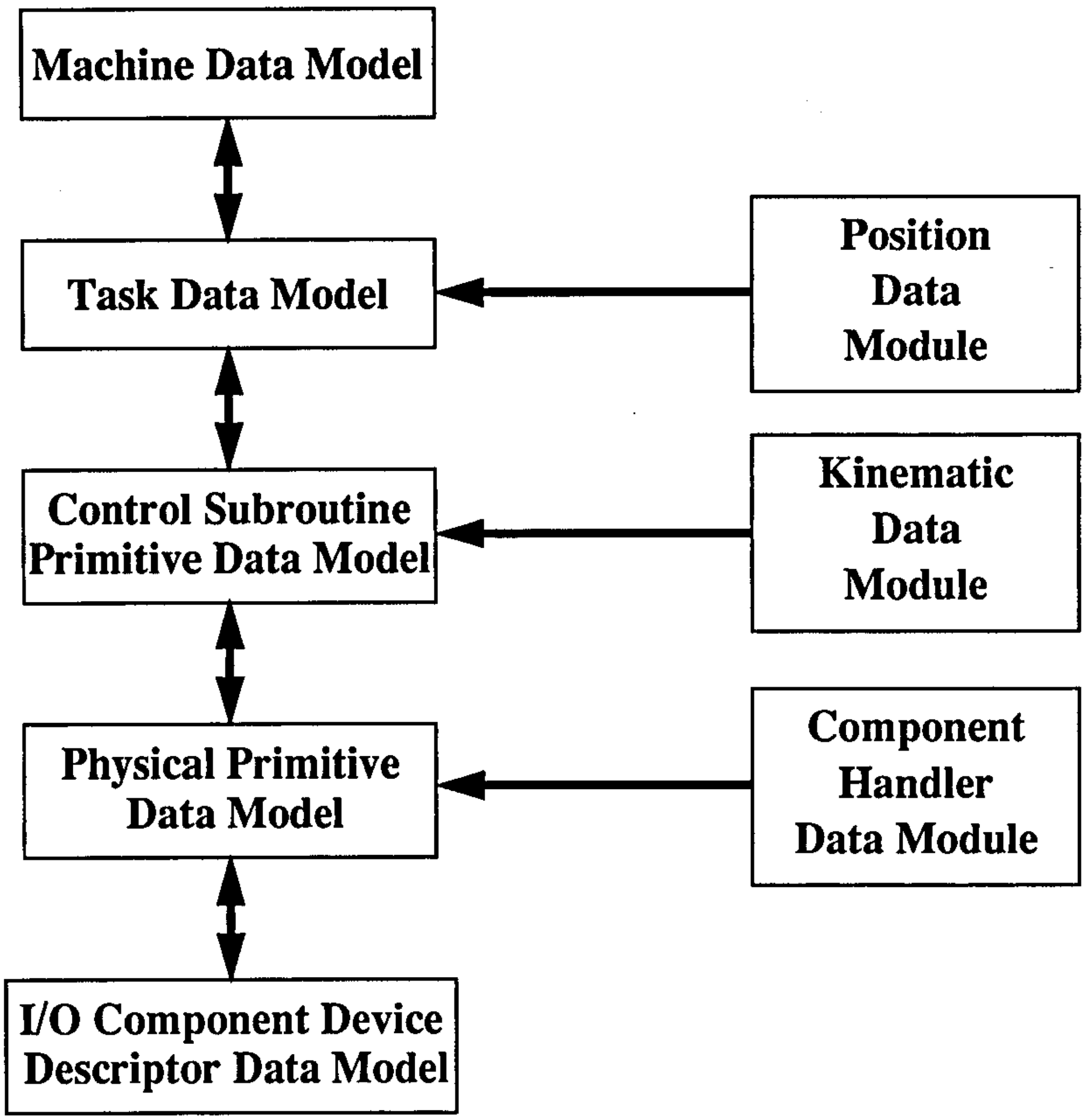

Machine Data Model
Task Data Model
Control Subroutine Primitive Data Model
Physical Primitive Data Model
I/O Component Device Descriptor Data Model
Position Data Module
Kinematic Data Module
Component Handler Data Module

Figure 3.3 Data representation hierarchy in UMC architecture

modules through referencing data relationships determined by the hierarchical structure or framework.

3.3 The UMC strategy - an open approach to modular machine control

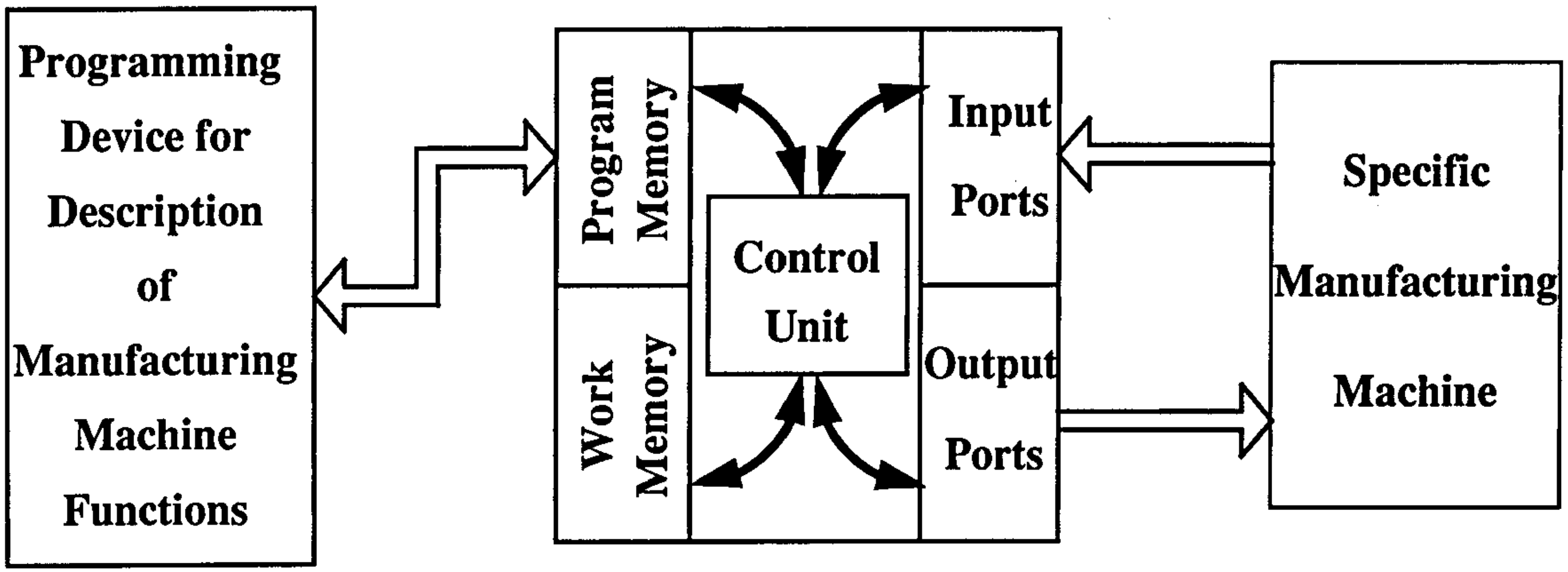
3.3.1 Current practice in machine design

Conventionally, machine design is achieved by one of two methods. The first commonly used approach is to tailor or customise a machine for specific requirements, such as dedicated or semi-dedicated automated machines for assembly, packaging and materials handling. This approach exists widely because it can provide mechanical optimization so that customised machines often demonstrate the required level of functionality (such as short cycle time and good accuracy) when achieving their specific manufacturing purpose. These machines include various computer controlled dedicated and semi-dedicated automatic machines.

Obviously, the customised approach can involve very high levels of highly skilled design and development work, particularly if the machine is relatively complex. Such machines are normally produced in small numbers and therefore the high cost associated with design and manufacturing are often difficult to justify so that automating may not be practical [Weston et al. 1989a, Weston et al. 1989b]. It is also difficult (or even impossible) to modify a custom designed machine, either with regard to its mechanical or its control properties.

The second conventional approach tries to produce a more general purpose machine (or indeed control system) typically of well defined mechanical configuration and flexibility. Thus a reasonable level of functionality can be produced at an acceptable cost to accomplish a variety of tasks. This approach is founded on the fact that producing one-off custom machines is very expensive in terms of development costs, whereas the higher capital cost of more general purpose machine may be outweighed by spreading development costs over many units. At the present time, many automated machines are controlled by PLCs (Programmable Logic Controllers). Such a PLC approach can be an advancement on the hard-wired or mechanical control method (through using a more flexible control device) but often it can only be used as a lower level local control method and tends to lead to "hard-wired" solutions and inflexible control structures. With the black box proprietary structure of a typical PLC, the control of a relatively complex machine tends to be disorganised and communication among machines (via PLCs) can be very limited due to lack of data visibility at the individual machines [Johnson 1987]. A requirement for more flexible and efficient machines in turn gives rise to a pressing need for more advanced control methods. Though the PLC approach is becoming more commonplace (since they can solve the problem of flexibility in a more general fashion as the functionality of PLC's themselves advance), a lack of a standard approach and the fact that PLCs have been built bottom up with ease of industrial use as a prime objective has ultimately limited their effectiveness. Hence PLCs do not represent an optimal control system solution to specific manufacturing problems. Figure 3.4 outlines the PLC type of approach. Despite the drawbacks of contemporary PLCs they can be viewed as a forerunner of highly flexible (i.e. configurable), user-friendly control systems a new generation of which will be generated through top-down design and building on modern process control [Weston 1990].

The modular approach has been widely adopted by machine designers at different stages of design [Tesar and Butler 1989, Karlen et al. 1990]. However, the designer is still expected


Programming Device for Description of Manufacturing Machine Functions
Program Memory
Work Memory
Control Unit
Input Ports
Output Ports
Specific Manufacturing Machine

Advantages:

1. Low level modular approach;
2. Low level of system engineering costs.

Disadvantages:

1. Insufficient functionality for complex manufacturing tasks;
2. Relatively "hard-wired" solutions to manufacturing tasks;
3. Limited configurability.

Figure 3.4 The machine control approach by Programmable Logic Controllers

to design manufacturing machines which are highly constrained both in terms of the physical and control structures. A generalised methodology based on a modular approach to system engineering has yet to be specified [Moore et al. 1990].

In addressing the problems faced and requirements to improve current machine design, a new approach to machine design, referred to as Universal Machine Control (UMC) has been derived by the Modular System (MS) research group at Loughborough which aims to formalise and structure modular machine design methods. The UMC approach aims to provide a software environment for machine design and control [Weston et al. 1989a and Harrison 1991]. It adopts modular methods from machine element design to machine configuration and run-time control. A hierarchical structure is maintained through the use of handlers which ensure conformance of machine building blocks which can be made available from various proprietary sources. In addition data driven methods are used to support highly configurable run-time control. Each of these features are intended to create an "open" reference architecture for general purpose machine design and control.

3.3.2 A description of the UMC approach

The UMC approach represents a step towards compensating for the disadvantages of conventionally controlled machines and in particular the need to realise sufficient flexibility whilst maintaining adequate performance at acceptable cost. It uses a "top-down" approach in creating a high level simulation environment for users to control and co-ordinate various machine control elements, tools, sensors and work piece flow in a consistent and global way [Doyle and Case 1991]. The importance of achieving high level control over a machine with complex tasks can be justified by the following two reasons.

Firstly, complex manufacturing machines normally deal with multi-task, high-speed manipulation and highly efficiency manufacturing, therefore they require high levels of performance when coordinating tasks. For this coordination requirement, there is a considerable communication problem among local machine control devices. The more complex a task is as a general rule the more complex is the coordination required and hence more serious communication problems are. High level control and supervision for these communication and coordination tasks is a natural solution.

Secondly, in terms of modular machine programming, a step up from the local programming and control of devices to more implicit approaches can facilitate faster and more user-friendly control over tasks of a modular machine. Conventionally, the user has to learn controller dependent (such as PLC) type of programming languages. If a low level proprietary programming is utilised, typically highly trained program staff are required: whereas if PLCs type of programming facilities are employed, typically there is a lack of co-ordination among sub-tasks of machine elements.

Bearing these problems in mind, the UMC architecture provides a method whereby machine designers and control engineers can resolve the diverse range of control and manufacturing problems into a consistent approach which can meet the emerging needs of a modular approach to machine building and control. This will eventually eliminate the problems caused by a customised approach; by which machine builders automating similar manufacturing tasks will engineer very different solutions, based largely on limitations imposed by in-house engineering resources and previous knowledge.

3.3.2.1 Universal features of the UMC approach

One of the inherent features of the term UMC is illustrated by the term Universal. Although

UMC is not truly universal (nor could it be) it adopts a device-independent method of standardizing control issues. A software device called a "handler" is normally used to interface with a vendor specific control device and on the other side it provides a consistent interface to high level control (see Figure 3.5). Thus handlers provide the interface between the run-time UMC system and customised device control. It consists of two parts: a device-dependent part and a generic interface to UMC task programs. The generic part of a handler waits for instructions from the task program and on receipt of such instructions it executes the command in question, sending a signal back to the task program to indicate whether or not execution is successful. The hardware specific functions of a handler depend on the hardware. Handlers are normally responsible for sending appropriate signals to local controllers. These signals are Ready-to-Execute, Error and Finish which indicate to the high level controller the protocol format of the proprietary device (such as a Quin motion controller).

3.3.2.2 Inherent Modularity of the UMC approach

The modular concept is explicit in the UMC architecture and can be viewed with respect to three aspects. Firstly the mechanical modules for machine building come from a model (or representation) of the decomposition of many automated manufacturing machines. Typical mechanical modules modelled include one degree of freedom (DOF) motion primitives or modules, two-DOF motion modules, three-DOF motion modules, and standard "mechanical connectors" for binding mechanical modules into a machine. However in the physical realisation of such machines currently only one DOF linear motion modules are used in one UMC demonstration rig, whereas revolute modules are included in a second machine demonstrator. The second modular aspect of UMC concerns the use of proprietary

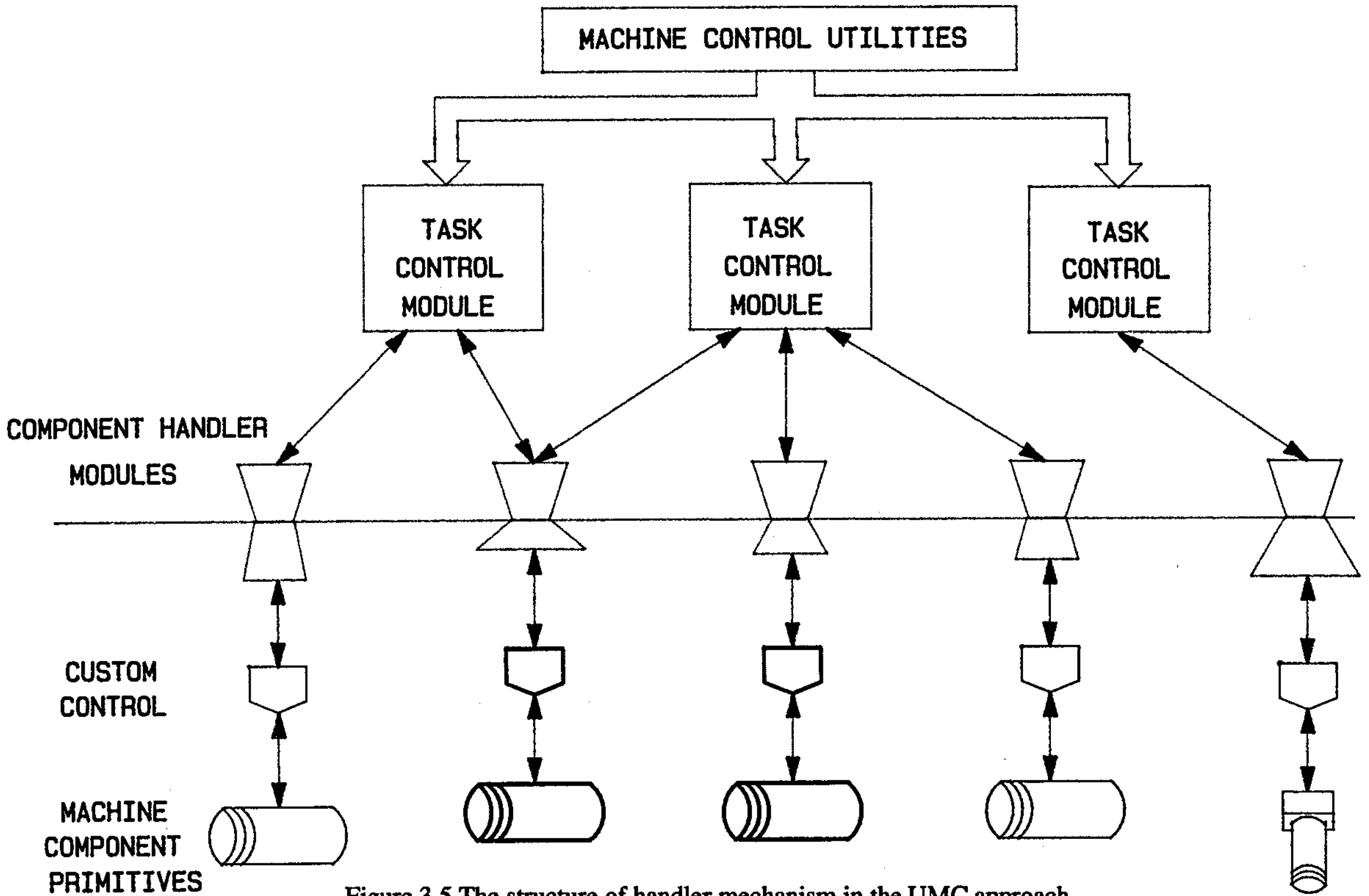

MACHINE CONTROL UTILITIES
TASK CONTROL MODULE
TASK CONTROL MODULE
TASK CONTROL MODULE
COMPONENT HANDLER MODULES
CUSTOM CONTROL
MACHINE COMPONENT PRIMITIVES

Figure 3.5 The structure of handler mechanism in the UMC approach

controller and I/O modules. Various commercially available motion controller modules and I/O processing modules have been purchased and used as control hardware modules. These commercial units are used along with supporting software UMC control modules including handlers to incorporate them within the control architecture. The third aspect of the modular concept is incorporated in the programming and run-time control of the UMC machine. The machine programming and run-time system maintains the coordination of single machine element related sub-tasks through inter-task communication. Therefore these sub-tasks can be stored as tasks modules (if they are newly created) or they can be directly selected from a task module family. The whole system maintains a well established modular structure which ensures that the designed machine has high flexibility in terms of configurability.

3.3.2.3 Hierarchical notion of the UMC approach

A further feature of UMC is its hierarchical nature. Different hierarchies are involved here dependent on the phase of the machine life cycle, i.e. the machines mechanical and control system design phase, the emulation hierarchy and the run-time control hierarchy.

The hierarchical nature of the current version of UMC is mainly typified by its data definition and the relationships between such definitions. In the data representation of the UMC architecture (shown in Figure 3.3), there are clearly five hierarchical levels for data modules. This reflects the hierarchical approach advocated in considering the requirements of modular machine design. It encompasses most advantages of conventional design, being a modular machine at the design stage and reconfigurable machine at the installation stage. Furthermore the use of a hierarchical approach ensures that at the design stage, the solution generated in response to specific manufacturing problems is not that of the usual conventional industrial robot approach, which only provides a sub-optimal solution in most cases. The other advantage is that the solutions obtained using this approach need not be specific to only one process or manufacturing problem, sometimes it is a solution to a number of manufacturing problems facilitating the production processing of a number of products. The modular design approach described allows two possible extreme design routes to be followed, i.e. to design a complete new machine or radically modify an existing machine. In either case, the methodology provides a set of geometrical modelling tools to allow several design solutions to be generated and compared through simulation, thereby achieving a better solution for a specific manufacturing problem. This design method employs a hierarchical approach and machine specifications should be accomplished first. This sequence is shown in Figure 3.6. Iteration is necessary to achieve better solution, therefore the UMC design methodology allows the designer to return to the machine design at various levels in the hierarchy to optimise a design solution.

3.3.2.4 Data-driven features of the UMC approach

In UMC, control and programming is achieved through using data-driven methods. A typical data flow of control parameters is also outlined in Figure 3.3. At the bottom level, the component handler data module passes the control parameters required by a physical component driver. In the case of a motion component for example these data from the handler are treated as commands and parameters, e.g. determining the type of move. Similarly at one level above, the control subroutine module transfers positional information to the component handler module through which the control data are further passed down to the component driver. The control related data modules are normally employed at run time, and task data modules are typically prepared for run-time data modules. This method

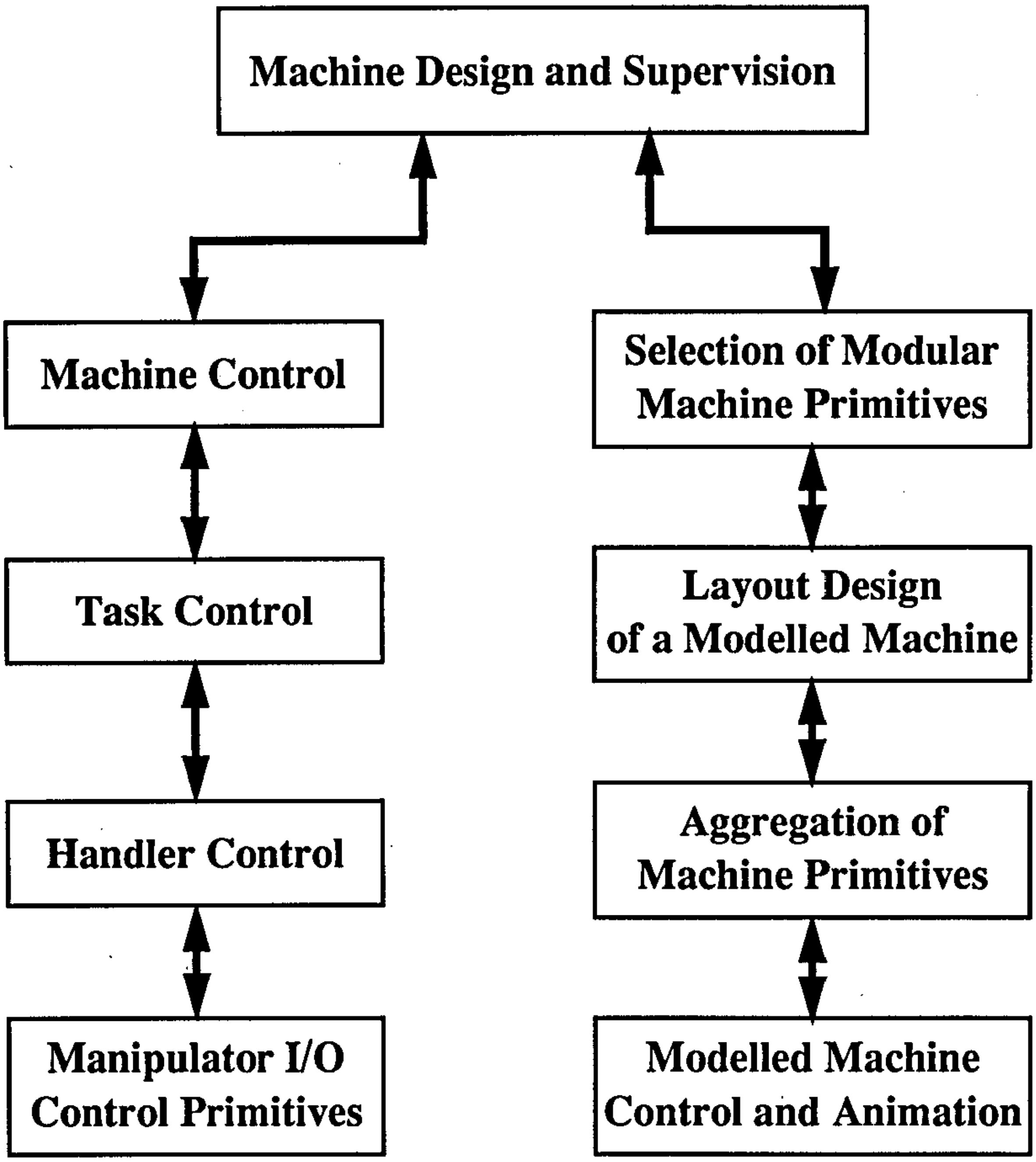

Machine Design and Supervision
Machine Control
Task Control
Handler Control
Manipulator I/O Control Primitives
Selection of Modular Machine Primitives
Layout Design of a Modelled Machine
Aggregation of Machine Primitives
Modelled Machine Control and Animation

Figure 3.6 The Hierarchy of the UMC architecture and its simulation approach

of data driving UMC is also organised so that computation and communication issues associated with real-time control are separated out leading to ready-to-use control parameters which are computed and processed at a high level of the hierarchy. This reduces the computation burden on the local controller and increases the system efficiency and response time. The six data modules used in the UMC architecture are briefly described as follows.

The I/O Component Device Descriptor (CDD) data modules take care of computer hardware dependence at the interface to the mechanical hardware, e.g. port addresses, communication protocols. Thus a 'standard' (or consistent) interface to I/O device descriptors is achieved: i.e. the component handler data module provides a standard UMC interface upwards to component control module and downwards to a hardware dependent component driver [Harrison 1989].

The Physical primitive data modules used in UMC define physical parameters of an associated component, e.g. the dimension and location of an axis of motion. This type of data module is especially useful at the emulation phase. For example the control primitive data module defines kinematic parameters of components, e.g. constraints on the maximum distance, speed and acceleration of an axis of motion. It also contains certain transient data fields which are used and evaluated at run-time.

Positional data modules describe the machine task related positional information for some primitive components. These positions stored in a positional data modules determine the destination position of a defined motion associated with either a single or a group of axes. The positions are stored in the modules in such a way that they can be recalled flexibly via a name associated storage method [Booth 1990].

In terms of kinematic data modules, a user friendly interface is provided which defines at a more abstract level various parameters (e.g. speeds and accelerations) for motions. These data are also stored in a separate data module using a name-velocity associated method. Currently, this data module is not available in a UMC physical machine environment, however it is available in the emulation system of the UMC architecture.

Task data modules within the UMC architecture are not part of the machine description database, and they contain purely transient data. Each task data module supports machine control functions to which particular task calls are made by application task programs. The task data module can be completely created before run-time if the machine data module is complete. When it is created, it contains data which identifies the required component handler module.

The machine data module contains a definition of the physical component primitives which form consistent tools of the complete physical machine for all application tasks to be programmed whilst the machine maintains its current configuration. The machine module is purely a machine description and it is used as a reference at run-time to create all other necessary program and data modules. The machine data module also contains data to define the task configuration.

With the above generalised four types of methodology employed in the UMC approach to modular machine design, UMC provides a general and highly flexible design method as described in Figure 3.6. There is a library of control system modules which emulate control functions required by the physical machine and descriptions of mechanical machine primitives to enable emulation of UMC manipulator systems. The user can select a set of control system modules from the library to configure a specific machine control system. Instances of the modules selected are then bound together by a hierarchical tree data structure, the selection and binding being achieved by using configuration tools. This naturally fits these modules into a hierarchical data model representing the machine and its controller. After the provision of additional data (e.g. a position data module and task description) a machine is fully described and the code and data representing the machine can be pre-processed directly to generate the run-time control software. On completion of the generation of run-time control programs, the data and control program can be mapped to custom control hardware and the physical machine then can be driven through the tasks so defined.

With an understanding of the advantages of the UMC approach, the authors' study has aimed to provide a consistent UMC design and emulation environment. In section 3.5, a general description of the emulation approach so created is outlined.

3.4 The state of the UMC architecture and its limitations

3.4.1 The benefits of the UMC approach

Based on the aforementioned methodologies, UMC modular machine design and control approaches have shown significant promise in various application areas [Harrison et al. 1988]. The designed machine can be reconfigured because of system modularity and its hierarchical data structure and therefore a wide variety of application tasks can be automated efficiently and cost effectively. The logical coordination of different group of axes can also be achieved by high level control. The data driven methodology offers not only considerable flexibility, but also data visibility and extendability, which can enable the machine to be fully integrated within its manufacturing environment.

Inherent properties of extendability and modularity also gives rise to two clearly identifiable advantages. First, very high functionality levels can be achieved (by tackling complex problems in an incremental manner) and second, support and diagnostic tool modules can be incorporated which are designed to serve the specific requirement of users and this naturally extends the machine's available levels of functionality. The inherent properties of machine reconfigurability can much reduce the machine lead time by facilitating re-arrangement into a new machine specific to new product or products. The hierarchical nature of a modular machine architecture provides important advantageous properties when compared with a "flat" architecture. There is a natural hierarchy in manufacturing machines and an increasing need to extend that hierarchy both upwards, to integrate the machine into factory level manufacturing, and downwards, to include low level machine components. The hierarchical approach can meet this requirement and the hierarchical data model also provides visibility since the users are allowed to access the data structure at each level. The obvious advantage of this visibility is that users can program, monitor and maintain the machine at different levels, and therefore the machine has much improved support and reliability. Additional significant benefit will accrue if the system requires modification and enhancement at a future date (a likely requirement as product models and manufacturing process change), resulting primarily from the natural visibility and extendability of the standard approach.

3.4.2 The limitations of current UMC architecture

Although the UMC architecture demonstrates a number of benefits described in the last section, the current UMC implementation suffers various limitations due mainly to there being insufficient manpower available to tackle all major aspects of the problem. In particular, prior to this study there was not a machine design and simulation environment. Current UMC versions do provide configuration tools for run-time re-configuration of an existing machine. However the use is greatly dependent on the designer's experience and there is a pressing need to simulate the modified modular machine and evaluate its performance, e.g. before actually re-configuring an existing machine. Based on such an evaluation and gradual optimization of both the machine layout and task allocation, a designer of a modular machine, using the UMC approach can be more confident that the new configuration of an existing machine is appropriate so that incorrect or unnecessary modifications can be eliminated. Based on parallels with robot evolution, a design and simulation environment for modular machines is an inevitable requirement to extend the designer's ability to design modular machines in a user-friendly and efficient manner.

From the perspective of machine capability, prior to this study the UMC implementations did not tackle the design of complex mechanism configurations. In such cases due to the wide range of possible complex combinations of motion axes, it is impractical to consider building and testing various optional control algorithms and machine mechanisms so that near optimal solutions can be realised. The physical reconfiguration of a UMC machine (or indeed demonstration rig) may mean large investment both on new motion module purchase and on installation. Algorithm testing on a physical machine can lead to lost time and production, judging from experience of corresponding robotic development. Alternatively it may be possible to use control algorithms developed in a simulation environment to control the physical machine thereby avoiding duplication. This can release the debugging burden at device control stage from the UMC machine designers and developers.

Prior to this study UMC machine programming was also limited to the level of a single joint and was typical achieved by using the C programming language. In terms of manufacturing task description, the single joint level is the lowest level of programming: this being equivalent to robot joint level programming. It is often necessary to program at this level to enable local machine task execution, control and programming but it is very often time consuming and results in difficulty for less trained programmers. The provision of only single joint level programming would limit the acceptance of UMC in industry. Programming should therefore be available at least at one level higher. Due to the real time requirement of the UMC physical machine, it is difficult to create a multi-level programming structure at run-time. However there is not as much concern about time constraints in the simulation stage. Thus it was considered to be feasible and desirable to create multi-level programming in the simulation phase.

The above limitations of current the UMC approach are a direct motivation for the work of this thesis. The research study aims to overcome these limitations and hence to enhance the capabilities offered by UMC systems. Essentially UMC versions prior to this study dealt only with the physical machine implementation and its real time control, whereas a simulation environment can fully complement such system by providing design and evaluation supporting tools in order ultimately to achieve high level planning and off-line programming. Before this ultimate goal is fully achieved, and especially the goal of off-line programming, an embryo environment needs to be created of base supporting tools so that necessary enhancements can be made. In the following sections, a general perception of problems faced and approaches taken are illustrated.

3.5 Use of an emulation approach to enhance the capabilities of UMC systems

3.5.1 A generalized emulation approach as an integral part of the UMC methodology

As described in section 3.1 and outlined in Figure 3.7, the UMC methodology encompasses the philosophies of modularity, hierarchical structuring and data or model - driven approaches. It has the advantage of producing flexible modular machines demonstrating high levels of reconfigurability and extendability at low engineering cost. The same philosophies can be employed at the emulation stage leading to a consistency of approach within UMC, i.e. common data models and control logic can then be maintained leading to ease of integration of the system within its host environment. The generalised ideal emulation system with consideration of potential integration with other life-cycle phases of manufacturing through using consistent data model and logic etc. and emulation approach itself are depicted in Figure 3.8 and Figure 3.9 respectively.

This system retains many of the best features of contemporary robot simulation systems including geometric modelling, and work cell layout arrangement. In terms of the chosen emulation approach, the system retains an inherently modular structure which is common in many simulators. Furthermore, the modular approach is adopted to establish the modelled work cell, machine and other objects from a very basic starting point (i.e. low level primitive modules), which implies the adoption of decomposition methods and axioms referred to in section 3.2.1. Here, all machines (including robots) can be modelled from these very basic machine elements, leading to simulation of a host of manipulator systems rather than manipulators which comprise a pre-defined fixed configuration.

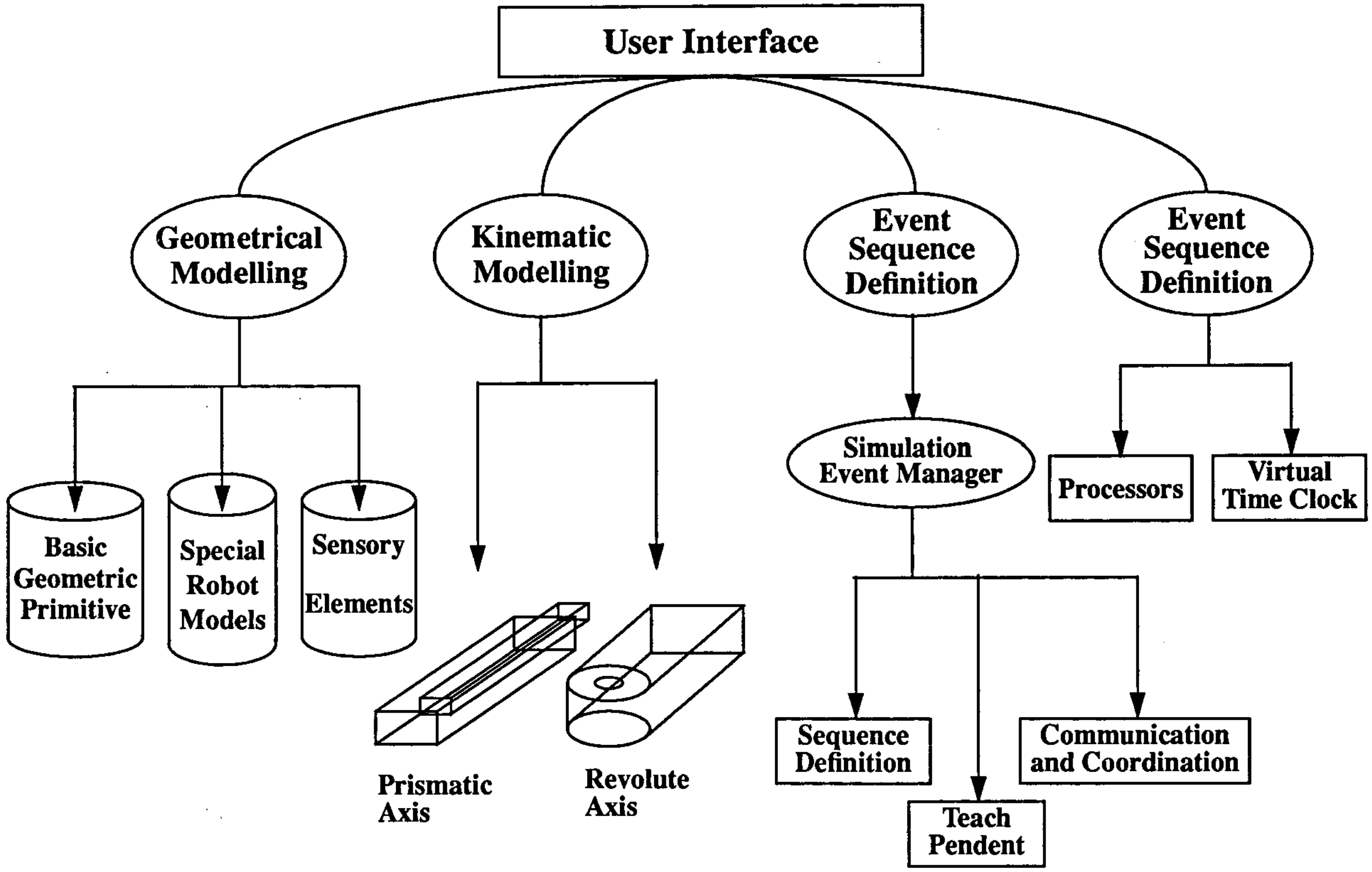


Figure 3.7 A generalised robot simulation system

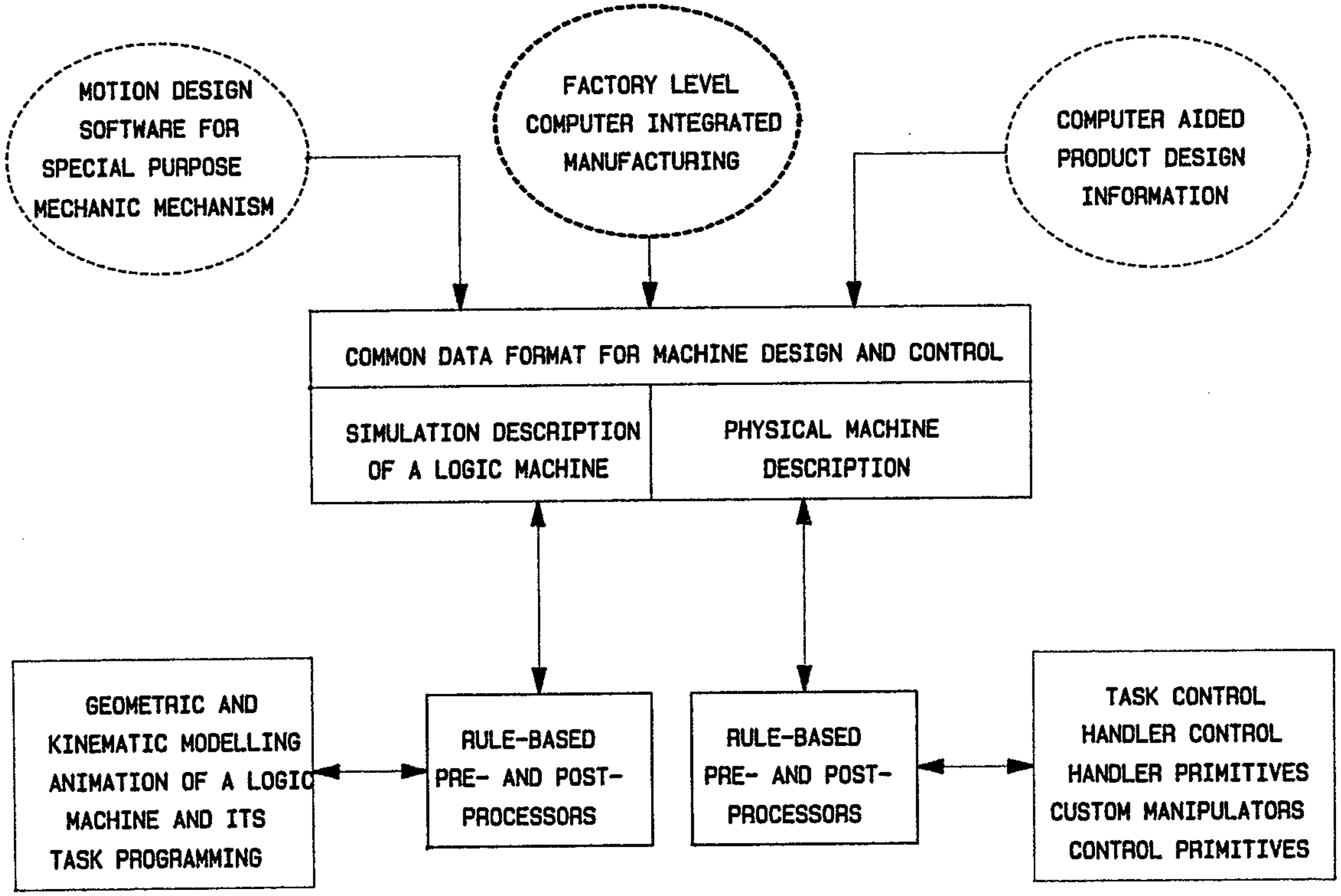

MOTION DESIGN SOFTWARE FOR SPECIAL PURPOSE MECHANIC MECHANISM
FACTORY LEVEL COMPUTER INTEGRATED MANUFACTURING
COMPUTER AIDED PRODUCT DESIGN INFORMATION
COMMON DATA FORMAT FOR MACHINE DESIGN AND CONTROL
SIMULATION DESCRIPTION OF A LOGIC MACHINE
PHYSICAL MACHINE DESCRIPTION
GEOMETRIC AND KINEMATIC MODELLING ANIMATION OF A LOGIC MACHINE AND ITS TASK PROGRAMMING
RULE-BASED PRE- AND POST- PROCESSORS
RULE-BASED PRE- AND POST- PROCESSORS
TASK CONTROL HANDLER CONTROL HANDLER PRIMITIVES CUSTOM MANIPULATORS CONTROL PRIMITIVES

Figure 3.8 A generalised ideal emulation system within CIM environment

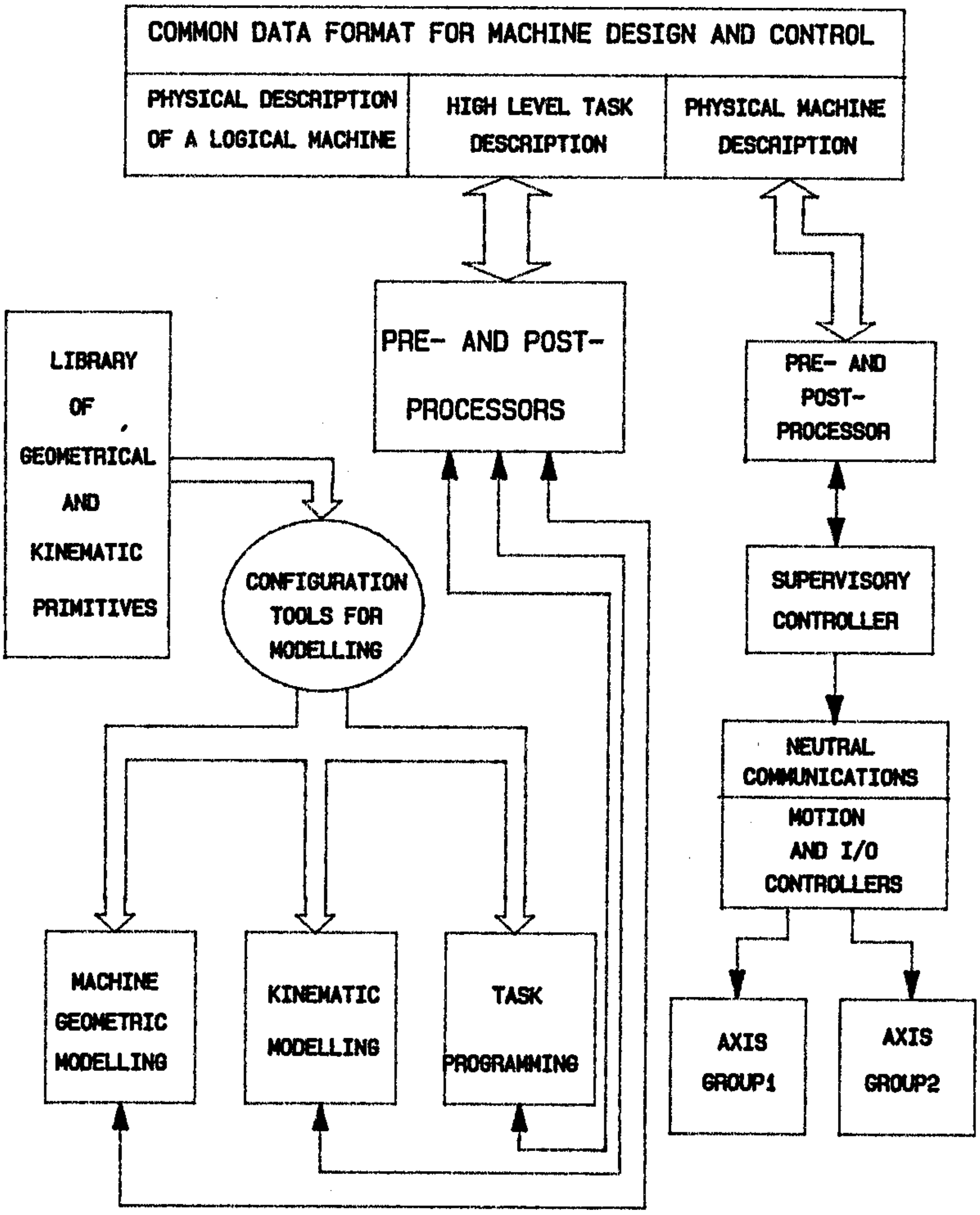

COMMON DATA FORMAT FOR MACHINE DESIGN AND CONTROL
PHYSICAL DESCRIPTION OF A LOGICAL MACHINE
HIGH LEVEL TASK DESCRIPTION
PHYSICAL MACHINE DESCRIPTION
LIBRARY OF GEOMETRICAL AND KINEMATIC PRIMITIVES
PRE- AND POST-PROCESSORS
PRE- AND POST-PROCESSOR
CONFIGURATION TOOLS FOR MODELLING
SUPERVISORY CONTROLLER
NEUTRAL COMMUNICATIONS
MOTION AND I/O CONTROLLERS
MACHINE GEOMETRIC MODELLING
KINEMATIC MODELLING
TASK PROGRAMMING
AXIS GROUP1
AXIS GROUP2

Figure 3.9 The UMC architecture and the author adopted emulation approach

A library is used to accommodate a family of modular machine single elements, ranging from non-powered standard geometric shapes such as cuboids, cylinders, to powered modules including machine single motion elements and motion specification elements. These basic elements contained in the module library will be referred to as modular machine building primitives. Each class or family of machine building primitives will demonstrate a set of variations in one or more its minor features. For example, modules within the same class of prismatic motion primitive may have different dimensions or moving distance constraints. Therefore, for different variations of the same class of machine primitive, different values are assigned to corresponding data fields where this assignment is necessary and easily conducted.

The configuration tools of the simulation system are a set of tools which provide an interface between the machine designer and the emulation system, thereby enabling an appropriate selection of machine building primitives and the aggregation of them into a complete machine within work cell model. For each type of machine building primitive, there is at least one configuration tool to accomplish its basic task. There are other tools for aggregating several joints into a group in a user designed way. This is achieved by using the generalized data structure for motion pairs described in section 4.2. The configuration tools also include editing facilities to modify the workcell of an existing modular machine. These facilities include dimension editing of an object or joints, construction editing of a group of joints, and location editing of an object or joint etc. Collectively these configuration tools facilitate the construction of modular machines during their simulation stage. Theoretically an expert system could be used to advise during the configuration stage by evaluating the suitability of the selected machine primitives for conducting the required tasks. This could be achieved by using a knowledge-based approach to recommend a choice of machine primitives based on user's requirements e.g. in terms of say the required working envelope, accuracy and other kinematic and dynamic features. However the creation and the use of such an expert system was well beyond the research scope of this study, mainly due to time constraints but also to allow the author to concentrate on methodology rather than means of achieving these methods where the means might involve work which defocuses the study.

Object manipulation defines a set of tools for a user to manipulate all modelled objects (from simple geometric representations to elements forming a grouped motion structure). In terms of control related machine primitives (e.g. sensory devices) the manipulation of these primitives should also be achievable to initialise their conditions. For kinematic manipulation, two types of manipulation should be provided, namely forward kinematic manipulation (i.e. drive one or several motion primitives to move a specific distance or turn through a specified angle) and inverse kinematic manipulation (i.e. drive one or several motion primitives to a required position and orientation in work place by moving a prismatic joint through such a distance or turning revolute joints through such an angle which results in desired target position in the cartesian space). Through providing both types of transformation objects can be activated to perform their functions and controlled by the user "manually".

A kinematic modelling facility can provide motion related functions which enable a designer to plan a satisfactory kinematic solution to the user's positional, velocity and acceleration requirements. This requirement can be as simple as a point to point move with constant speed of a single axis, or it can be as complex as a three dimensional curve with various velocity specifications relating to a group joints. The versatility of the kinematic model and completeness of available motion combination will determine the capability of an emulation system and its application areas. There are already many robot simulator systems available, however, being a kinematically fixed and limited structure, they have demonstrated limitations. The kinematic modelling facility is especially important in that sense, being a generalised modelling facility to deal with various motion primitive configurations of different combinations. The kinematic motion planning can be straight forward single path plan, it can also be an "intelligent" path plan which calculates motion's at all passing point and check if there is a collision or impossible move. For some very accurate motions, the kinematic model should provide an error correction facility.

Programming such a modular machine is a very demanding task. Although it is very difficult in one single machine task program to control and coordinate the operation of a modular machine, it is relatively easy to control every machine primitive at its local control level. Due to its modularity and kinematic structure, there is large amount of communication and coordination activity required within the modelled modular machine. At one level above the local primitive control level, task programming should provide efficient functions which cope with these communication and coordination issues. The operations vary with different application areas. An application specific programming facility is bound to limit its utility. Hence generalized and task control based task programming functions are likely to find more wide application areas at this level (i.e. one level above the single primitive or local control level). Some of the basic and common commands at this level can be provided by an emulation system. However, a user or application specific task programming facility can be derived by combining second level from bottom commands or functions into macro commands. Such an open approach is also applicable in the other parts of emulation modelling facilities.

The emulation system also includes an interface between the simulation environment of a modular machine and its physical realisation in the form of an operating machine. A detailed description of this interface and associated integration issues are outlined in Chapter 9.

A data ring and tree method can be used to present both the hierarchical and modular features of a modular machine. Each part of a machine primitive's features can be represented by one or several data blocks, which are a collection of various data types. For example, an event in a task program can be expressed by one data block with certain data fields, which accommodate all event related information in the task. Each of these data blocks can be arranged in various ways. The proposed method for use in the author's UMC emulation is referred to as a data ring and tree method, which arranges the data block in such a way that data blocks at the same level of the hierarchy hang together in a ring form. For these data blocks which are located at one level below in the hierarchy, a new data ring is introduced from one data field at the data block which starts a new data ring. The newly introduced data ring can be seen as a branch in the data hierarchy tree structure, and it is normally called a child data ring. The data block which starts a child ring is called a parent data block. This structure puts the same level objects or their features in a ring which establishes a connected relationship. The tree characteristics can thus determine parent-child relationships: in this way the combined data structure can reflect the hierarchical and modular features of a modular machine. The data block size can vary depending on the complexity of object feature to be represented. However, the same featured object aspects can be represented in a similar manner. Thus if a geometric representation or model of a physical machine can be expressed by such a data ring and tree structure, then the modelled machine can be manipulated in the same way as the real physical machine with the same data structure used at the machine programming stage. Typically a task of a modular machine will comprise several sub-tasks where each sub-task can contain several events or operations. The data ring and tree structure also has the capability to satisfactorily represent this type of task.

3.5.2 GRASP - A robot simulator as a research tool on which to build a UMC emulation system

GRASP [Bonney et al. 1984; Yong et al. 1986; Bonney et al. 1987] stands for Graphical Robot Applications Simulation Package and was derived from a computer aided design tool for ergonomics called SAMMIE (System for Aiding Man-Machine Interface Evaluation [Case et al. 1986 and Porter et al,1986]). Both packages were originally developed in the Department of Production Engineering and Production Management at University of Nottingham. GRASP gradually emerged as a commercial software package and is marketed by a company called BYG Systems Limited (which was formed by some of original GRASP researchers).

GRASP uses a three dimensional body-modelling package, which provides the means for constructing objects from geometric building primitives, e.g. cuboids, regular N-side prisms, etc. The primitives are grouped together into a hierarchical tree structure, therefore several of them can be arranged in appropriate positions and manipulated as a single entity. The model is displayed on a computer graphical terminal in a wire frame form. A high-level statement is incorporated within the GRASP language to define the joint structure, its constraints and other data associated with a robot. Separate entities, modelling the "flesh" of each joint, may then be incorporated into the robot model. Any entity within a model may be nominated as a robot tool and mounted on a robot. These tools are then associated in a special way with the robot and the local axis system of some entity below the tool in the hierarchical tree structure is selected as a tool centre point (TCP) the TCP being a reference for defining a desired tool position at the robot control stage.

A modelled robot then can be manipulated within its workplace which is the owner of the robot or the header of robot hierarchical tree. A robot can be manipulated at two levels. The first of these levels allows the user to move individual joints through a number of degrees or through specified distances, whereas at the second level the desired position of the tool (TCP) is defined so that the GRASP system needs to compute the required increment of each joint to achieve that position. There are alternative methods available to the users for defining a TCP position, including absolute position and relative to a reference object. All the methods ultimately determine the location and orientation of the tool centre point.

In order to automatically manipulate the robot, a robot program needs to be created. In GRASP a robot program is a sequence of discrete robot actions or operations involving the definition of associated positions. Such a sequence in GRASP is termed a track. It is the track which determines the robot operation during simulation. A track can be dumped into a process, which is a robot dependent and time based model since all positions are stored relative to the model objects rather than the robot. GRASP provides path- definition commands so that users can define the path between locations, these mainly being straight or circular paths with time or velocity control. Tracks can also be dumped in one or more processes to simulate (or animate) time dependent motions and robot operations within the workplace. At this stage, cycle time estimates are produced.

The GRASP robot simulator also provides collision detection facilities among modelled objects. Possible interference between objects can also be checked automatically by using the CHECK command, however this is very time-consuming. Off-line programming is also possible if a post-processing facility is used. Currently available the post-processors can convert the GRASP robot program into VAL, VALII and ASEA AR- Languages.

Like most robot simulators (as described in section 3.2.1), GRASP can only be used in modelling a limited family of robots with certain kinematic classes, which in the case of GRASP currently include kinematic configurations of Unimation, Adept, ASEA, OLP system, RCM3-KJKA, CLOOS, and Reflex-CINCINNATI robot. For non-GRASP-standard robot configuration assistance is typically required from BYG (the supplier) in writing a specific Fortran subroutine. The programming and simulation of a robot workcell is also especially designed for serially chained robot type manipulators. However, BYG do provide an open binary version of GRASP, which provides a three dimensional solid modelling facility and a simulation environment for robot work cell layout.The open binary version of GRASP was made available to the author and was used as a basic simulation tool for modular machine design and simulation.

The open binary version of GRASP provides a user interface which allows a user to create a new data model and also to manipulate the GRASP robot model, which are impossible in normal GRASP version. Some basic geometric primitives (e.g. cuboids, cylinders etc.) can be created by calling corresponding subroutines. Supporting tools, such as message manipulation and display, screen layout arrangement, new commands addition, and means of requesting data from user are also provided.

Chapter 4 Mechanism motion synthesis and the establishment of a primitive library

4.1 The notion of a library of primitives

A modular manufacturing machine can be decomposed into sub-systems which are themselves decomposed into machine primitives (section 3.2.3). The machine elements or primitives obtained from a physical machine decomposition can be further analysed and generalized according to their features. In this study, the kinematics of a machine and its primitives are considered to be critical. Thus the decomposition of machine primitives is carried out by considering typical kinematic features of the machines, namely: motion type, primitive shape and compound motion type. A computer graphical technique is employed in this study, and hence a geometric representation of machine primitives is needed to enable graphical machine simulation.

The simulation study focuses on the above two aspects (i.e. kinematic modelling and graphical simulation) and certain idealisations and assumptions are made as follows because generally they either represent trivial influence on physical systems or can be improved by control engineers.

(1) Gravitation and inertial effects are negligible. This implies that the physical drive system of each machine module can develop sufficient drive force (torque) to enable the required load motion to be achieved without error. During simulation no attempt will be made to solve dynamic models describing the behaviour of the load system or systems;

(2) Through assuming the use of a "perfect drive system", this also implies the assumption that the modelled motion primitives can perform their target motions as

accurately as intended, i.e. that there are no steady state errors (including the absence of backlash in manipulator joints, no position overshoot or velocity lag errors). Thus the motion primitives will be assumed to have an infinite acceleration capability;

(3) Also as the dynamic behaviour of simulated machines are not modelled, there will be an inherent absence of transient motion characteristic, i.e. the simulated load and module motions will instantaneously reach their commanded or specified position.

Based on these idealizations of a physical machine and the stated research emphasis, a family of motion primitives can be generalized and modelled (or represented) by a common set of parameters assigned to attributes which characterise the modelled primitives. As an extremely large number of alternative types of motion primitives could be described it is necessary to categorise them into families of a similar class. This can simplify the process of storing (in computer memory) the models and subsequently selecting and building machines from them. Here the notion of storing these parameterised module families in a machine primitive library was employed. The reasons and advantages of building a machine primitive library are described in the following.

There are various industrial machines which have been designed to automate (or semi-automate) a spectrum of manufacturing operations. These machines demonstrate a variety of properties but they exhibit distinct similarities. Hence it is possible to decompose and generalize them into various machine constituents [Benhabib et al. 1990, Tesar and Butler 1989] which individually or collectively can give rise to those properties. For example a typical machine constituent could be a single degree of freedom prismatic or revolute joint. Alternatively, it could be a gear box or cam follower, etc. Conceptually, constituent building blocks of this class can be considered to be low order primitives (i.e, "modules" providing single degree of freedom motion). An alternative decomposition might lead to primitives (or modules) of a higher order, which demonstrate motion in more than one degree of freedom. Parameterisation of these various machine primitives enables not only a class (or family) of them to be stored in a standard form but indeed different classes or families also. In this sense, the establishment of a library of parameterised machine primitives can standardize procedures for building machines and can much reduce the need for repetitive effort. These machine primitives can then be reused by machine builders and new modules added to the library as a need for such modules is identified.

Since parameterised machine primitives are modular building elements of a complete machine model, they provides more flexibility than the normal geometric interfaces commonly found in the solid modelling of conventional pedestal mounted robots. From the library of machine primitives a machine model builder can select machine primitives and aggregate them into a complete machine. They could constitute a machine which already exists, a completely new machine conceived by the designer or a machine design which is somewhere between these extreme cases. Therefore the machine primitive library provides a tool which can facilitate machine design and modelling. In later chapters of this thesis the potential use of machine models, from the perspective of building Computer Integrated Manufacturing (CIM) systems will be considered.

In the context of this PhD study, the library of machine modules can be viewed as a collection of models of (i) geometric primitives of machines (which are an idealised representation of low or high order modular building elements), (ii) motion kinematic primitives of machines (which represent various types of motion), and (iii) non-motion machine building primitives. The library implemented includes examples of each of these three main types of modular machine building element.

The geometric primitives provide the motion elements which collectively will form the moving parts of a machine configuration. The motion (or kinematic) primitives specify the type or types of motion followed by a modular machine and its component parts, i.e. defines characteristics such as spatial path, velocity and acceleration. The non-motion machine primitives are those modular accessory devices required which themselves do not possess any power driven capability (except possibly through gravitational forces). The non-motion primitives are utilised in conjunction with the motion primitives to accomplish certain manufacturing tasks, e.g. storage of components in its magazine. An illustration of this concept can be found in Figure 4.1.

4.2 Building a single degree of freedom geometric primitive by using a generalized data structure

The modelling of a machine for computer simulation requires its description (or representation) in the form of a computer model. Most simulation systems focus on some high level property of a machine's function. Since the interest of this research lies primarily in the kinematic performance of a machine, the simulation model should provide a sufficiently detailed geometric and graphic representation of the motion of configured machines. The user should be able to visualise the machine's construction through providing a display of both static and kinematic properties via animated graphics. Thus methods of representing these degrees of freedom (both individually and collectively) need to be analysed and a full implementation of the necessary geometrical and kinematic information made within a suitable data structure. In order to avoid confusion the term joint

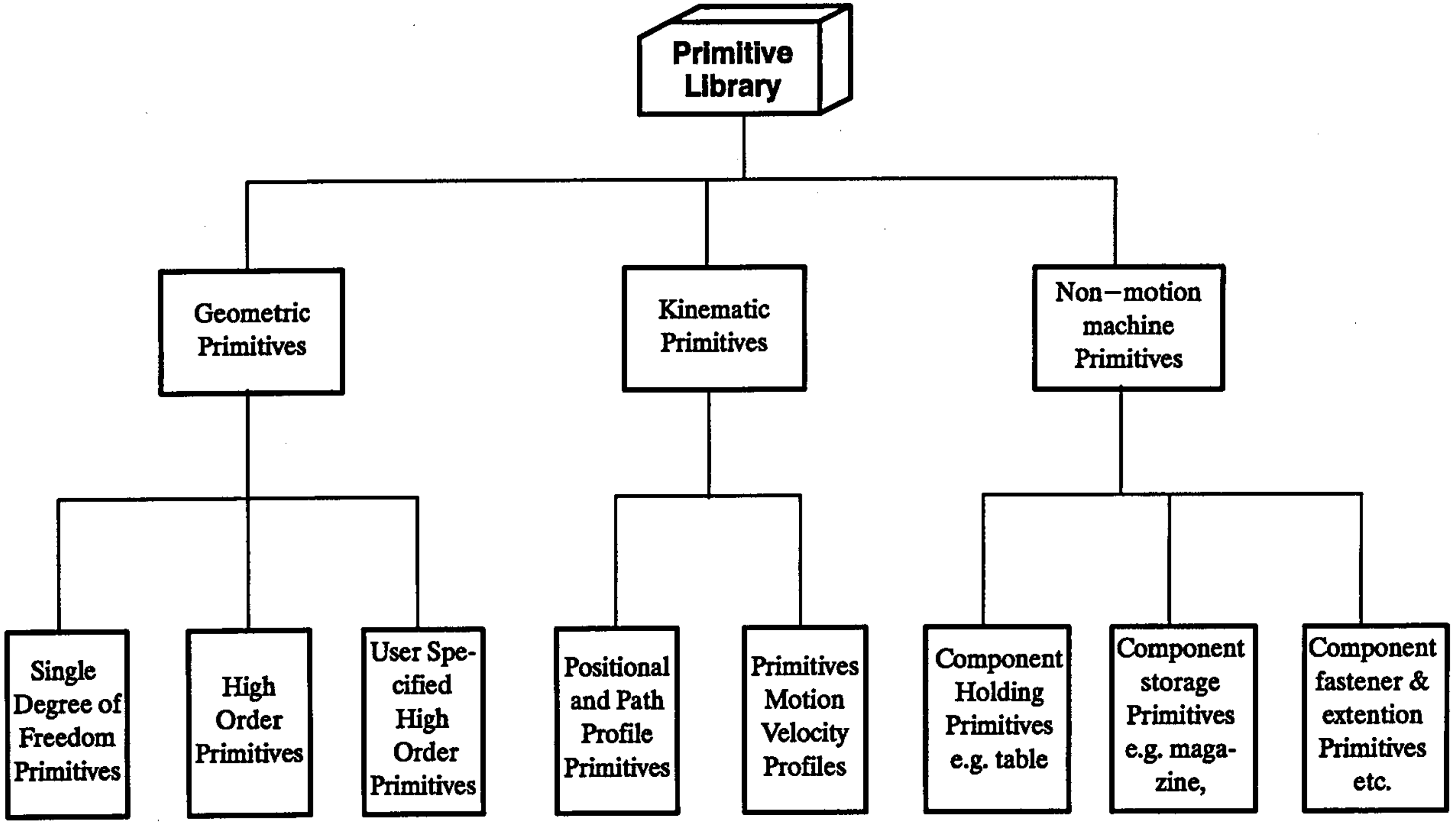

Primitive Library
Geometric Primitives
Kinematic Primitives
Non−motion machine Primitives
Single Degree of Freedom Primitives
High Order Primitives
User Spe-cified High Order Primitives
Positional and Path Profile Primitives
Primitives Motion Velocity Profiles
Component Holding Primitives e.g. table
Component storage Primitives e.g. maga-zine,
Component fastener & extention Primitives etc.

Figure 4.1 The hierarchical structure of Primitive Library

will be used to describe a single motion element of the actual machine, whereas the term axis will be used to define such a single degree of freedom motion element of the modelled machine.

4.2.1 Analysis of axis construction (motion pair structure)

A mechanical mechanism can be viewed as a means of transmitting, controlling or constraining relative movement, and is typically comprised of some combination of joints and links. A joint and the two links it connects are known as a kinematic pair. Each kinematic pair has two connected basic elements allowing relative movement. In terms of kinematic features, a joint then has a moving part which is driven by some type of power unit to achieve an intended target position, and a fixed (or base) part which typically supports the moving part. If two mating elements of a joint are in surface contact the kinematic pair of the joint is called a lower pair; if the contact is in the form of a point or line the pair is known as a higher pair. Lower pairs include translational joints, revolute joints and their combinations, whereas higher pairs are typified by gears and cams [Dimarogonas 1988, Haug 1989, McCloy and Harris 1987]. Some typical joints are now analysed.

4.2.1.1 Synthesis of lower pair motion primitives

In computer modelling, a prismatic joint needs to be characterised by five main items of information, namely: the location and orientation of an axis in a global coordinate frame; the relative position of the base part to the axis; the relative position of the moving part to the axis; dimensional information concerning the moving and base parts; and finally the kinematic constraints and axis manipulation information i.e. currently used kinematic manipulation parameters. In order to accommodate all these aspects of axis information, a coordinate frame system has been designed as shown in Figure 4.2.a. The global coordinate frame system establishes the origin position and orientation of the complete machine model. The axis coordinate frame system defines the origin of an axis position and its orientation relative to the global origin. The origin of the axis moving part specifies the initial position and orientation of the moving part of an axis. The geometrical representations of the two parts of an axis then can be independently attached to these local coordinate frame systems. The coordinate frame of the base part geometry (i.e. the lower front corner if it is a cuboid) can coincide with the axis coordinate frame, but it can also be offset by the machine designer. The same applies to the geometry of the moving part.

This generalized approach provides the flexibility in modelling different varieties of the same type of prismatic joint, i.e, the relative position of two local coordinate frames can be arbitrarily defined. This axis coordinate frame system on the other hand also depicts the structure of an axis, and hence if the geometry to represent one part of the axis is missing, the structure is still maintained. This is very useful simplification feature when the model becomes very complex, and allows the hiding of some trivial geometries.

As implied by a kinematic pair, a joint always has two elements which reflect the relative static and kinematic status of an axis. For revolute joints, the same coordinate frame system should and can be applied. In order to locate the axis origin at an appropriate position and orientation, the axis geometries need to be arranged according to the changes of motion type and the origin of the axis geometry. However, the same general axis structure remains (see Figure 4.2.b). A one degree of freedom rotation around axis Z_1 in the local coordinate frame $O_1X_1Y_1Z_1$ is allowed.

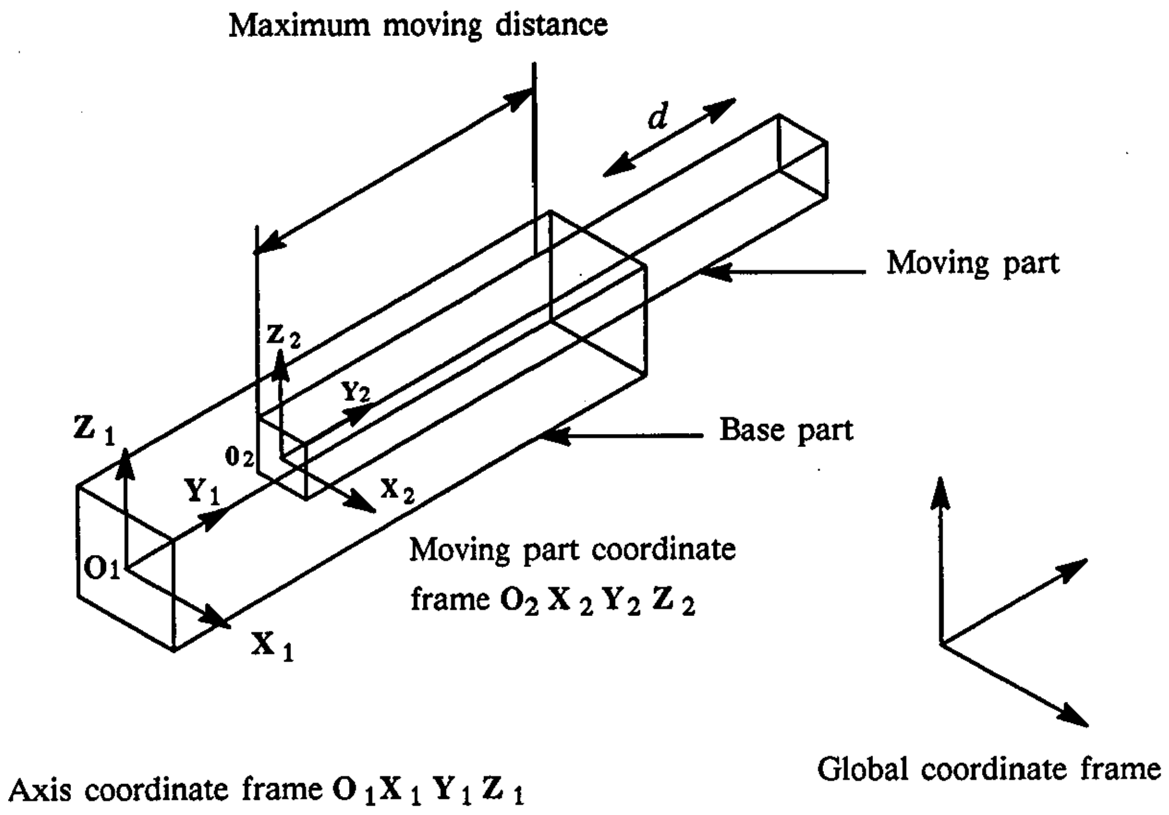


Figure 4.2.a Coordinate frame system establishment of a prismatic axis

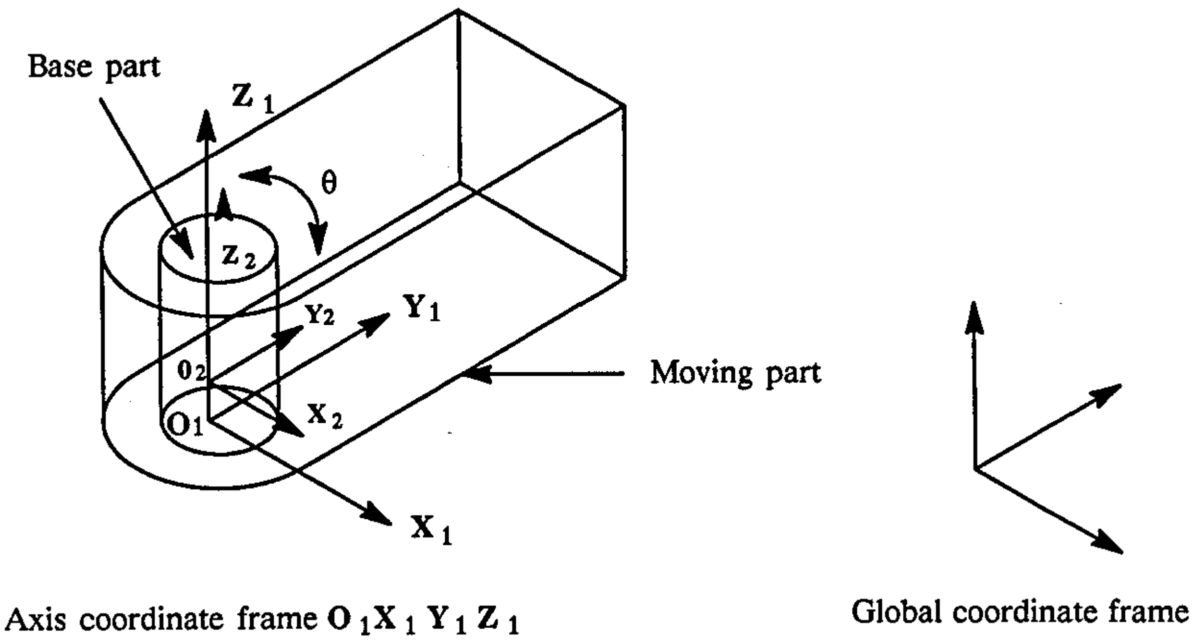


Figure 4.2.b Coordinate frame system establishment of a revolute axis

In terms of lower pairs, for a long time it has been generally accepted that there can be no more than six lower pairs, namely: prismatic; revolute; screw; cylindrical; planar; and spherical. Waldron [1972] has comprehensively proven that there can be no other types of lower pairs.

The main reason is that apart from the basic surface element, general helicoid, there are no other surfaces which can form the elements of lower pairs. All the other surface elements which form lower pairs are in fact special forms of a helicoid, viz: (a) a general surface of revolution, (b) a general cylinder or prism, (c) a circular cylinder, (d) a plane, or (e) a sphere. [Hunt, 1990, 1978]. A close study of these six types of lower pairs results in the conclusion that the constituent freedom of the spherical, planar, cylindrical and screw pairs appear as a combination of the single degree of freedom revolute and prismatic pairs. Thus it is necessary to generalize the degrees of freedom available from a combination of revolute and prismatic pairs and substitute the other four lower pairs with these two elementary motion pairs.

The benefit of this simplification lies in the fact that every single degree of freedom building element can be controlled separately by computer, leading to a simplification of the problems of co-ordinating combined motion of this type. It might be noted that decomposition of a mechanical mechanism and the association of control system (which itself may be a decomposition of a higher order control system) can lead to a module which is sometimes referred to as a mechatronic unit . Since there is less complexity in direct joint control, the substitution of a more than one degree of freedom mechanism by a combination of directly controlled prismatic and revolute joints can provide more freedom and higher precision when controlling mechanisms with several degrees of freedom. On the other hand, despite advances made in regard to communication capability between computers, interprocessor delay may ultimately limit the working of several directly controlled single degree of freedom joints when they need to work in a closely coordinated manner.

In the remainder of this thesis it will be assumed that the functionalities of the other four lower pairs can be replaced by an appropriate combination of prismatic and revolute joints. A suggested proof of the notion that the other four lower pairs may be replaced can be found in [Dimarogonas 1988 and Hunt 1990].

4.2.1.2 Analysis and degeneration of higher pair motion primitives

Though lower pairs have the capability to withstand considerable applied load due to their surface contact (which can be accurately manufactured easily), higher pairs are sometimes indispensable and still find many application areas within traditional machine design [Hunt 1990].

A typical example of a higher pair mechanism is a cam-follower, which is traditionally the simplest means of achieving a complicated displacement profile with respect to some variable (commonly time). However, with advances in computer control, these complicated displacement requirements can be accomplished by the use of a mechanical prismatic joint associated with a flexible controller which stores or computes the required time versus displacement of the prismatic joint. Thus instead of a cam driver, a computer controlled actuator can drive the prismatic joint controlling movement in an appropriate manner to achieve a given time and displacement profile within a machine cycle. Therefore, the higher pair can be replaced by a lower pair both in a real machine as well as in the simulation, and the cam-follow structure can be degenerated into (considered to have been replaced by) an equivalent prismatic joint. This replacement of cam drivers provides the following advantages.

(1) Computer based controllers can change the "cam tour" (i.e the relationship between time and displacement) flexibly and easily i.e. under software control) and this can significantly reduce the cost and lead time to produce a new physical "cam-follower". Furthermore the use of programmable transmission elements of this type can lead to less downtime during product changes on manufacturing machines [Sinha 1990];

(2) With fewer mechanical parts, there can be reduced wear and lubrication problems;

(3) It is easy to maintain the manipulator of a software cam and the software cams are more reliable because there is no line or point contact [Sinha 1990];

(4) Since there is no restriction on the rise and fall profiles, software cams provide wider range of choices even for more complicated transformations.

Despite the above mentioned advantages, computer controlled lower motion pairs on the other hand do suffer the limitation of lower power and limited speed which needs to be overcome in the future through providing better control and drive equipment.

The gear box is the other type of conventional higher pair mechanism, transmitting power to individual drive shafts at various speeds. The use of gear boxes for transmission is based on the assumption that the size, shape or the handling requirements of the product range to be processed by a particular machine throughout its lifetime are known [Hunt 1990 and Sinha 1990]. However, with reducing product life cycles and an increasing pressure to minimise product costs, the use of a gear box type of transmission system can become expensive due to its inflexibility in coping with either faster throughput or an entirely new product. The expectations for new generation of production machines to be able to flexibly adjust to changing requirements have encouraged researchers to derive a family of "intelligent" controllers and drivers [Quin 1989]. In addition to software cams, software gear boxes (or so called programmable transmission systems) can achieve the necessary transformation between a displacement (measured by an encoder, which is a replacement for the input pinion of a gear box) and the position of an output shaft of servomotor. Furthermore the capacity to store different position relationships between the input device and the output shaft can lead to much increased flexibility. Kinematically, this simplifies the gear box into a set of revolute joints rotating in a synchronised and coordinated way. Therefore, the joint structure and coordinate system of a revolute joint can still be applicable in the gear box case. In the next section a solid modelling method is described which is suitable for representing modular machine primitives.

4.2.2 Computer geometric representation

Since all constituents of a machine can have their own physical manifestation, the employment of an appropriate geometry to graphically describe a machine element (or part of it) is a commonly used approach in graphical simulation [Jayaraman and Levas 1988]. In particular, a piece of geometry or compound geometry similar to the shape of a physical machine component can facilitate (in a simulation) the visualization and identification of the component's static and kinematic behaviour. Precise geometric representations of a machine and its working environment may be required to enable evaluation of the machine's performance before it is configured. For example the detection of potential collisions among several components may need to be determined.

As described in section 4.2.1, in terms of machine geometric and kinematic modelling, it is possible and feasible to degenerate various forms of mechanical pairs into prismatic and revolute pairs, thus greatly simplify the complexity of computer modelling.

In simulation technology, the focus of the observation (i.e the type of visual interest in a machine) is critical in designing, and ultimately determining the efficiency of the simulation system. Generally, a joint can be represented by two geometric entities which model the moving and base parts of an axis. Different physical joints have different geometric shapes. Since this study is centred on the kinematic modelling of modular machines, a generalised axis representation is abstracted from the coordinate systems and the construction analysis of an axis described in section 4.2.1.

Two single pieces of geometry can only statically represent a frame of an axis. As a motion pair, an axis encompasses these two pieces of geometry together with a coordinate frame which is established to connect the two geometry items and the axis. In order to associate an axis with its working environment, the complete representation of an axis needs information about the relationship between the local axis frame and the global frame which is a fixed frame in the machine modelling environment.

The completeness of any geometric representation of a machine environment will determine the accuracy with which simulation can be achieved. However it also has an impact on the efficiency of the simulation. These are two contradicting aspects, as the computation power of current computer systems is not limitless. An axis can be simply represented by two single pieces of geometry and five coordinate frames. It can also be displayed on a screen as two compound geometries and related coordinate frames. The compound geometries are the results of addition and subtraction of several primitive geometries. In terms of geometric representation, the base geometry (which is a part of the axis basic geometry) possesses the ownership of other geometries which are added to the base geometry, in the same way as the physical joint constituents can be assembled onto a physical joint base. The extending end of a compound geometry is open and any number of different primitive geometries can be owned by the base of the compound geometry. This will typically result in a low simulation speed, especially when graphic animation is involved.

In order to clarify a model visually, part of an axis geometry can also be dummy (in the sense that it is not displayed and modelled graphically) to facilitate at higher speed the visualisation and understanding of complex configurations (i.e. those involving several interconnected joints). By using a simplified representational model of any group of axes it becomes easier to animate a machine, however this will be at the expense of less accurate modelling. Meanwhile this form of simplification also improves the clarity of the end user's visualisation of the model. A trade-off between the modelling accuracy and efficiency must be made.

4.2.3 Data structure of a single axis for simulation

It is critical to establish a common, inclusive and flexible data representation to computerise the modelling and simulation of a geometric axis. This data structure should be inclusive (in the sense of completeness of joint information) so that it can ensure that the data are informative enough for modelling, evaluation and task programming. A common data structure for an axis should comply with the modular methodology, facilitate the manipulation of axes within a machine modelling environment and simplify the modelling of modular machines. The data structure should also be flexible since the simulation of a machine environment requires large amounts of computation and covers a considerable variety of physical joints. Effective data searching of the data structure of a single axis or a group of axes within a complete machine environment has considerable impact on the flexibility and efficiency of the simulation system.

The information about an axis in a modular machine simulation system can be divided into the following parts:

(1) dimensional information, which specifies the geometric representation of the fixed and moving parts of an axis;

(2) spatial information, which describes the spatial relationships between the local axis frame and geometry coordinate frames of an axis as well as the relationship between the local axis frame and the global frame;

(3) physical information about the kinematic features of a joint, such as the maximum position, velocity and acceleration of the joint;

(4) the dynamics of a single joint and interacting forces amongst various machine elements. However, with respect to this research study dynamic information is only included in the data structure to enable future study and is not currently used for simulation purposes.

Based on the above classification, three relevant data types are created by subroutines, namely: geometric primitives; spatial entity data blocks; and a general data blocks. For further details on these subroutines, see Glib manual by BYG [Glib 1989].

For ease of manipulation all data blocks created by the above subroutines are always associated with a name. For clarity of the name representation, the following defaults are introduced as a suffixes:

i) ***name_M***: the name of the axis moving part;

ii) ***name_B***: the name of the base part of an axis;

iii) ***name_A***: the local coordinate frame, corresponding to the name of the axis manipulation data block.

With the above defaults and classification of axis information, three sets of functional subroutines are derived and the respective data blocks can be created and filled with the required information. One data block of the dimensional information representing a cuboid is illustrated in table 4.1. The data block has a number of words and can be divided into two major parts, viz: a common part and a part which is data block specific.

In the common part, the first word of the block is reserved for an encoded specification of the block length and type. The second word is the address of this block. Since the ring (the entity data blocks at the same level of the hierarchy are formed into a ring) and tree (all entity data blocks are arranged in a hierarchical tree) type of data structure is employed in this research, two integer spaces are reserved for ring continuation pointers. The Principle Ring Pointer (PRP) is normally used for forward data block searching and the SRP (Secondary Ring Pointer) is usually used to search the data block of another ring of data blocks with the same features. The fifth word is reserved for the name block pointer of this block. The data type specific part of a data block varies in terms of its length and content depending on its requirements. For the cuboid it has enough words to describe its dimensions in X, Y and Z of the local coordinate frame (this having its origin at the cuboid

Word No	Data Contents	Comments
Word 0	Data type and length	Defines the data block type and its word length in binary bit form;
Word 1	Principal Ring Pointer	Data Ring Continuation pointer for forward ring searching;
Word 2	Secondary Ring Pointer	Data Ring Continuation pointer for Secondary Ring Pointer searching;
Word 3	Name Block Pointer	The address of this data block's name data block;
Word 4	Display Status Flag	The status of current graphical display ;
Word 5	General Block Pointer	Data Ring Continuation pointer for other special data ring searching;
Word 6	X Value of a Cuboid	Dimensional specification of a cuboid in X direction;
Word 7	Y Value of a Cuboid	Dimensional specification of a cuboid in Y direction;
Word 8	Z Value of a Cuboid	Dimensional specification of a cuboid in Z direction;
Word 9	Empty	Can be used for other purpose;
Word 10	Empty	Can be used for other purpose;
Word 11	Empty	Can be used for other purpose;

Table 4.1 The data contents of a cuboid data block

corner).

The spatial information data block has the same common part as that of a cuboid. Since this block is used to describe the spatial relationship between a geometry local coordinate frame and another frame, a word to associate the block with the geometry block address is introduced. In a spatial layout sense of geometry, it is usually convenient to establish an ownership between the spatial data block and the geometry. Thus when the spatial positions and orientation vary, the data block applies the change through the ownership to update the geometry's spatial position and orientation. For details, see section 6.1. The other part of the spatial information data block is used to store the position and orientation of the geometry relative to another frame. For details of the data block see table 4.2.

Kinematic information about an axis is stored in an axis kinematic feature data block. It also has a common part and a block specific part which describes a kinematic feature of an axis. For the details of the block, see table 4.3.

Based on an analysis of axis structure and data representation, a physical joint can be described in the computer data structure as an axis composed of seven basic data blocks (as shown in Figure 4.3).

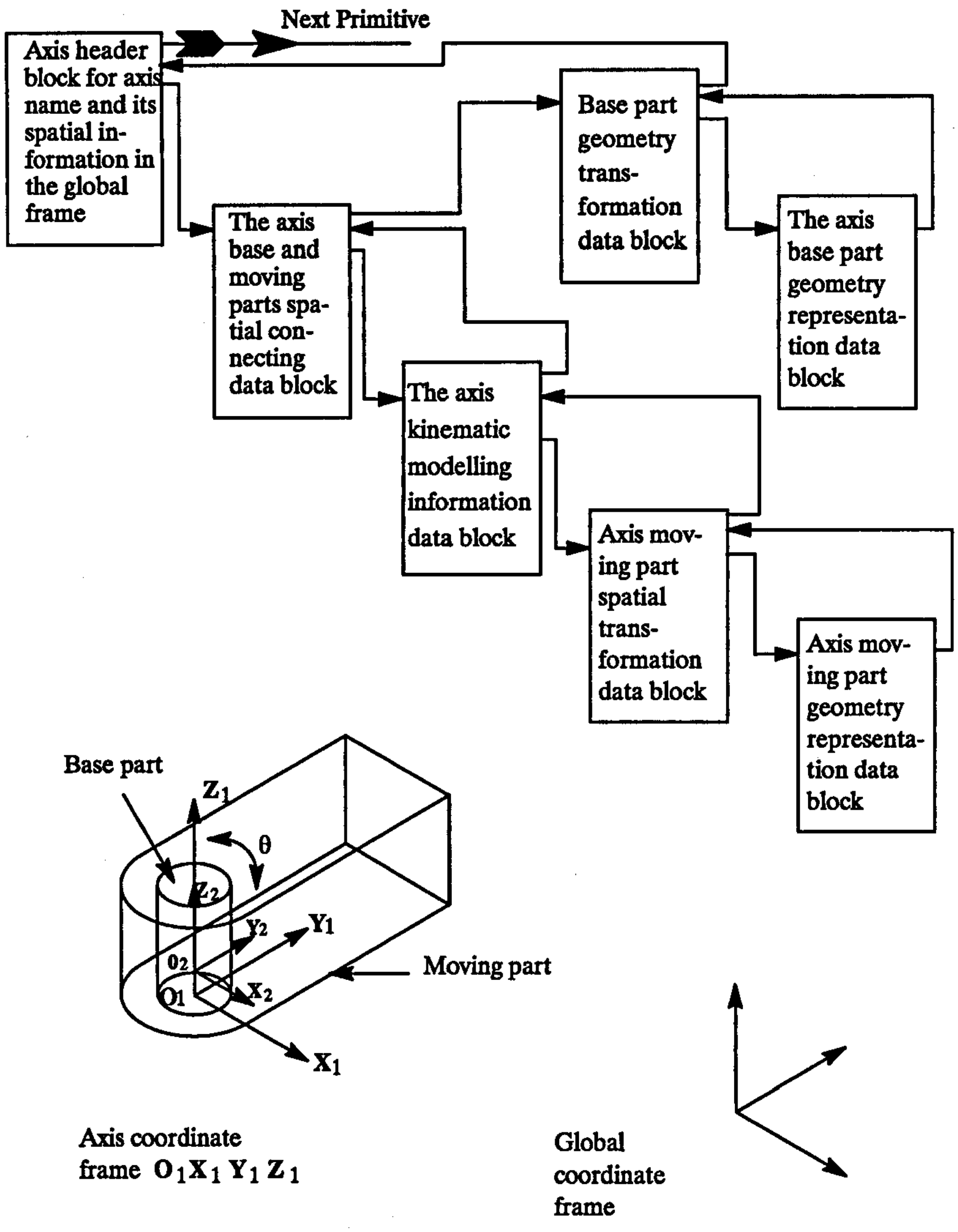
The moving part geometry and base part geometry data blocks are at the bottom of the tree branches and they define the axis' dimension. Since it is possible to have dummy geometry included (concerning either moving or base parts) two separate data blocks are required to improve flexibility. The corresponding spatial data blocks for the base part specify the positions and orientations of the geometries relative to the axis' origin; the spatial data block of a moving part specifies the spatial relationship between the local axis moving geometry and moving part origin, since they are normally offset and there is a need to describe this offset in the kinematic simulation. Due to variation in the relative location of the two parts, they can be coaxial or offset. These two separate data blocks then enable the various possible combinations the two parts of the joint to be modelled in the axis data structure.

Word No	Data Contents	Comments
Word 0	Type & length	Data type and length
Word 1	P R P	Principal Ring Pointer
Word 2	S R P	Secondary Ring Pointer
Word 3	N B P	Name Block Pointer
Word 4	D S F	Display Status Flag
Word 5	G B P	General Block Pointer
Word 6	E R S	Entity Ring Start Pointer
Word 7	P S N	Picture Segment Number
Word 8		From word 8 to word 16, nine vari-
Word 9		ables about the orientation of an ob-
Word 10	Part of 4*4	ject are stored. Since a 4*4 homoge-
Word 11	Homogeneous	neous transformation matrix always
⋮	Transformation	has 0 0 0 1 as its last row, the row
⋮	About orienta-	can be omitted without losing any
Word 14	tion	useful information. The translational
Word 15		information is kept in word 17,
Word 16		word 18 and word 19
Word 17	Value along X	Linear translation along X
Word 18	Value along Y	Linear translation along Y
Word 19	Value along Z	Linear translation along Z
Word 20	Empty	Empty for special purpose
Word 21	Empty	⋮
⋮	⋮	⋮
⋮	⋮	Empty for other use
Word 29	⋮	⋮
Word 30	Empty	⋮
Word 31	Empty	Empty for special use

Table 4.2 The data block contents of transformation

Word No	Data Contents	Comments
Word 0	Type & length	Data type and length
Word 1	P R P	Principal Ring Pointer
Word 2	S R P	Secondary Ring Pointer
Word 3	N B P	Name Block Pointer
Word 4	D S F	Display Status Flag
Word 5	G B P	General Block Pointer
Word 6	E R S	Entity Ring Start Pointer
Word 7	P S N	Picture Segment Number
Word 8		From word 8 to word 16, nine vari-
Word 9		ables about the orientation of an ob-
Word 10	Reservation for	ject are stored. Since a 4*4 homoge-
⋮	axis manipula-	neous transformation matrix always
⋮	tion of rotation	has 0 0 0 1 as its last row, the row
Word 16	or translation	can be omitted without losing any
Word 17	around or	useful information. The translational
Word 18	along Z axis of	information along X, Y and Z is
Word 19	local frame	kept in word 17, word 18 and word
⋮		19. Therefore only twelve words are
⋮		used to store all spatial information.
Word 24	Home position	The home position of an axis
Word 25	Mini. position	The maximum negative position
Word 26	Maxi. position-	The maximum positive position
Word 27	Maxi. velocity	The maximum allowable velocity
Word 28	Maxi. accelera.	The maximum allowable acceleration
Word 29	Empty	Empty for future use
Word 30	Empty	Empty for special use
Word 31	Empty	Empty for other uses

Table 4.3 The data block contents for kinematic manipulation



Next Primitive
Axis header block for axis name and its spatial information in the global frame
The axis base and moving parts spatial connecting data block
Base part geometry transformation data block
The axis base part geometry representation data block
The axis kinematic modelling information data block
Axis moving part spatial transformation data block
Axis moving part geometry representation data block
Base part
Z1
θ
Z2
Y2
Y1
O2
O1
X2
X1
Moving part
Axis coordinate frame O1X1Y1Z1
Global coordinate frame

Figure 4.3 A primitive axis and its generalised axis common data structure

The axis kinematic data block retains the joint kinematic information and the constraints for the axis kinematic simulation which are used in section 6.3. The spatial data block, above the kinematic data block, keeps the spatial information relating the moving part origin and the axis origin. The last block is about the spatial information of axis' origin and the axis owner's origin. This is the head of the axis data structure and the axis name is stored in this data block. The data blocks are formed into a data ring and tree structure. It complies with the natural hierarchical structure of a physical joint and its spatial information decomposition. The moving part spatial information can be easily searched and manipulated from the axis head data block.

The above describes representations of the five information entities required to model an axis. The data structures so chosen by the author offer a flexible way of modelling modular machines.

4.3 The derivation of library of geometric primitives

Based on the use of the above data structure a family of single degree of freedom mechanical modules was derived in this study as follows.

4.3.1 Prismatic axes

Coaxial prismatic axes have been included in the library and have the following basic default shapes: either two cuboids; two cylinders; or a combination of one cuboid and one cylinder (see Figure 4.4.a). Though a simple axis shape (in terms of the axis two parts) is provided here, pointers are provided is left for users to add any sub-shape geometries onto these two basic pieces of geometry.

Offset prismatic axes have been parameterised in the library. They can be used as a mechanical slide module or a carriage, and are modelled as cuboids (see Figure 4.4.b).

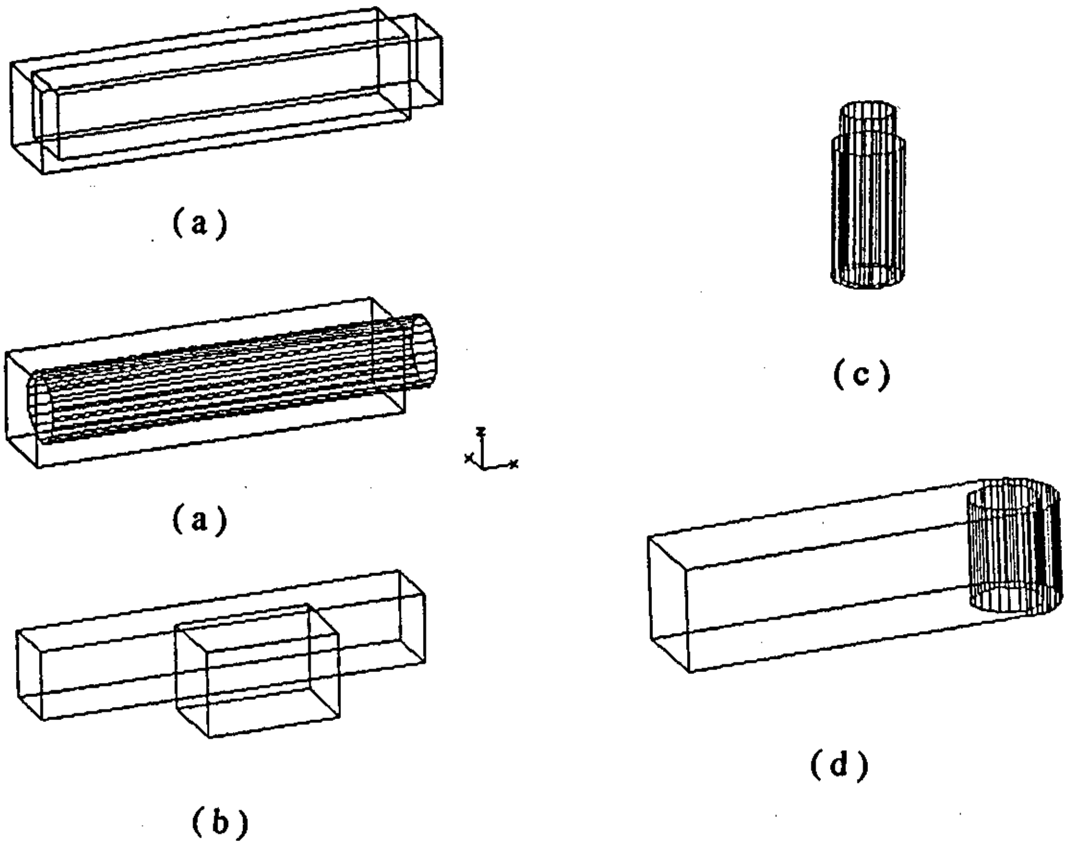


Figure 4.4 A family of single degree of freedom axes modules

4.3.2 Revolute axes

Coaxial revolute axes have been provided in the library as show in Figure 4.4.c. A commonly used swing type of revolute axis is also included in the library (see Figure 4.4.d).

4.3.3 Screw type axis

As an exceptional case, the screw type axis is also included in the library as single degree of freedom modular unit. Graphically it is represented by a revolute axis (see Figure 4.4.c and 4.4.d). However a special driver is needed to drive it should linear or rotation motion be a requirement. The features of such a driver are described in section 7.8.4.

4.4 Grouping of up to three axes to create higher order library primitives

In the majority of manufacturing application areas there is a requirement for motion in three dimensional space. Thus seldom will a a single degree of freedom unit be employed on its own. However at the other extreme, multi-degree of freedom mechanisms (such as conventional serially chained robots) will often include redundant motion capability for a specific set of requirements. Thus although a multi-degree of freedom mechanism may represent a feasible kinematic solution, it will often not represent the best solution because (i) the machine may be unnecessarily costly, (ii) it may demonstrate relatively poor accuracy (and repeatability) and (iii) it may involve relatively long cycle times (e.g. limited effective power to weight ratio). One approach to machine design is to specifically design a complex machine tailored to manufacturing certain types of component. However, the alternative approach of designing a distributed machine will gain in popularity with the increased availability of modular building elements [Ranky and Ho 1985]. This in turn will lead to cheaper solutions, with improved levels of accuracy and repeatability when compared with conventional industrial robots

4.4.1 Reasons for limiting the number of serially chained single degree of freedom primitives

An advantage of limiting the number of joints in a serial chain is that it much reduces modelling problems, i.e. it can lead to a reduction in complex modelling approximation errors and decreases the computation time to derive a kinematic solution. It can also simplify and facilitate modularization of simulation modelling of a complex machine and provide the possibility to study the mechanically distributed machine configurations. In addition it enables the possibility to derive at least one kinematic solution for every configuration considered here.

When considering the major axes of motion of contemporary pedestal mounted industrial robots they are dominated by four types, namely: cartesian (PPP); cylindrical (PPR); spherical or polar coordinate (PRR); and revolute or articulated (RRR) configurations [Wolovich 1987, McCloy and Harris 1986] although SCARA (Selective Compliance Arm for Robotic Assembly) configured robots have also become widely used, particularly for light assembly applications. Here P denotes a Prismatic axis and R denotes a Revolute one. Three serially chained low level machine primitives (with their axes mutually perpendicular) can easily reach any position within three dimensional working envelope (note the four types of conventional robot configuration employ three joints to locate the robot gripper at a spacial position). The other three orientation related joints can be decomposed from three position related joints [Tourassis et al. 1989]. This practically implies the need and feasibility of a three degree of freedom mechanism to reach a possible location.

Another reason why it is possible to limit to three the number of articulated joints in a group is that it is possible to drive a number of groups (of up to three joints) in a distributed way. In such a machine system the practical restrictions (such as computing power of a controller and the complexity of kinematic algorithms) imposing a limit the maximum size of any serial chain (and hence on the range of kinematic solutions) is removed [Harrison 1989]. If more joints are required another group of axes can be created and logically (rather than physically) connected to the previous groups.

In addition, due to the modular approach adopted, tasks requiring motion can be achieved via the concurrent operation of several simple sub-tasks (and associated sub-motions). This results in parallelism which can lead to shorter cycle times.

Due to the above reasons, it is desirable (particularly in the context of a proof of concept PhD study) and indeed feasible to limit the joint number in any given group to three. The mechanical parallelism (i.e. use of concurrently operating groups) can then be applied in the configuring motion mechanisms and in the modelling of a modular machine.

4.4.2 Possible combinations of prismatic and revolute axes within the limit of three

The configurations possibly come from two approaches: (i) articulated or serially chained configurations of up to three axes and (ii) distributed or physically decoupled configurations.

4.4.2.1 Articulated configuration of two axes

This is a common configuration used when building simple manipulators and is often used in industrial robot configurations. The advantage of this type of configuration is that the manipulator has better reach capability (improved dexterity) than other two degree of freedom mechanisms. However, since the base joint has to carry a second (or chained) joint, the moving mass will adversely effect the accuracy (through link defection etc.) and the speed of response (i.e. the power to weight ratio will be reduced as will the maximum acceleration of the end effector). All possible combinations of two axes groups and thus all of their configurations and working envelopes are analysed in Figure 4.5. Since this involves two axes, it can only produce a two degree of freedom surface envelope.

4.4.2.2 Articulated configuration of three axes

Milenkovic and Huang [1983] have analysed the major combinations of three joint linkages. They only considered simple chains, which they defined as open linkages, involving the use of revolute and prismatic joints with the joint axes either perpendicular or parallel to each other. Closed linkages were not included in their study. Amongst the 36 possible combinations of these three joints, there are essentially 12 classes of combination available after discounting redundant configurations and eliminating others through a process of degeneration of degrees of freedom (16 of 36).

However, even for these twelve simple chains, as earlier described only four of them have found wide-spread use in industrial robots. In establishing methods of designing machines from serial chaining it is useful to fully study the characteristics of all twelve configurations and to provide corresponding supporting tools to facilitate design processes. In attempting a quantitative appraisal of these configurations the author chose to use the following criteria:

(a) Inherent accuracy, this being the theoretical accuracy with which a specific configuration can be modelled and controlled;

(b) Ease of control which will depend upon the complexity of the kinematic solution for different configurations;

(c) Working envelope which specifies the working volume of the configuration;

(d) Speed of a movement of the end point of a configuration, which will characterise the speed with which a configuration can reach its target position when compared with

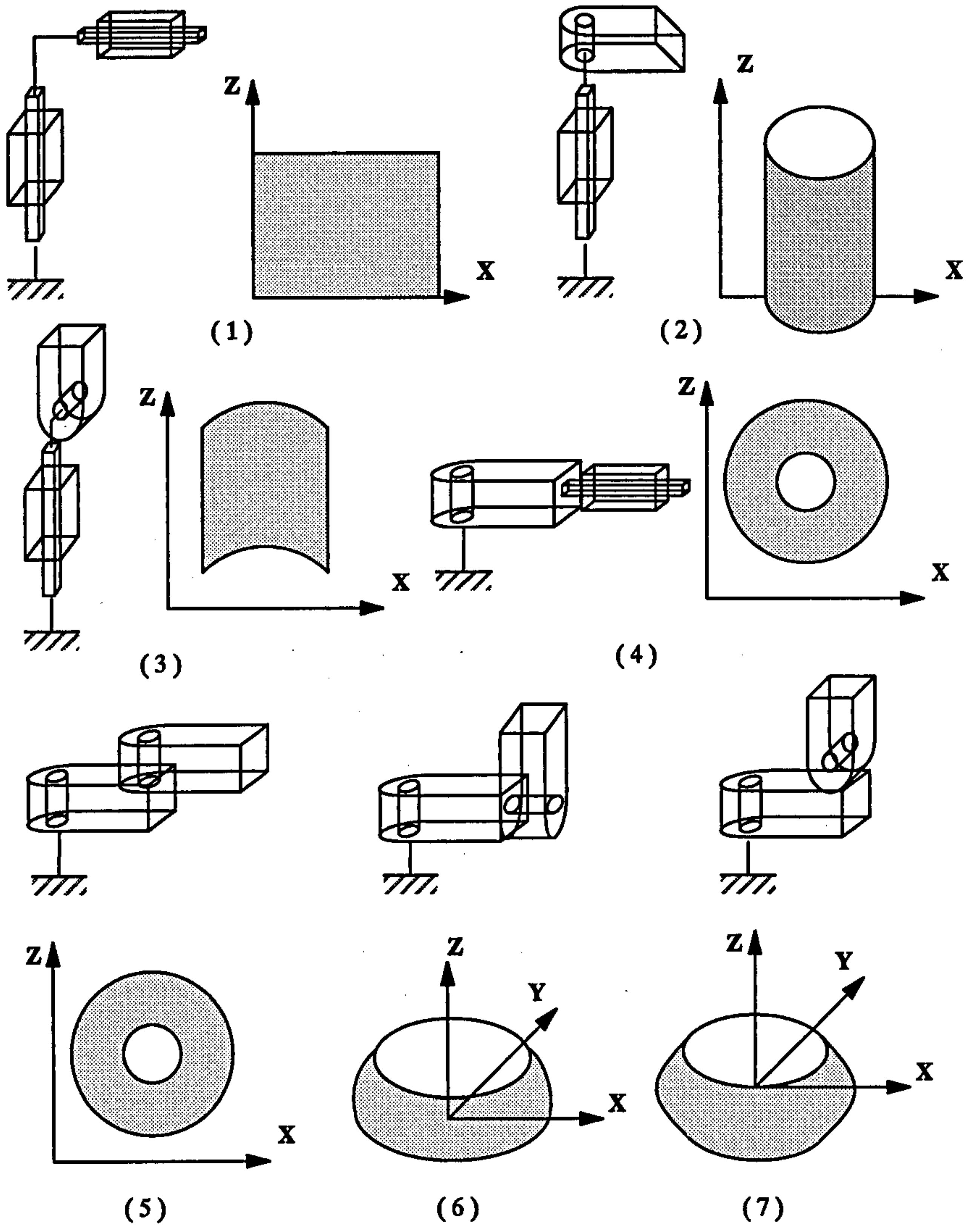


Figure 4.5 Analysis of seven possible two joint configurations and their working envelopes

other configurations.

Prismatic joints will demonstrate the same order of inherent accuracies as the resolutions of their feedback devices for each axis (which is essentially determined by the resolution with which position measurement in practice is achieved). Any configuration which employs one or several prismatic joints can maintain a high level of inherent accuracy in the degree of freedom in which the prismatic joint is used. Since the resolution of a revolute joint is an angular one, the actual inherent accuracy is essentially determined by the product of the joint length and angular resolution. The joint length is always greater than 1 millimetre which is the usual measurement unit of a positional accuracy, therefore the length actually magnifies the resolution by the joint length times, and a poor inherent accuracy in that degree of freedom appears.

Since a revolute joint introduces trigonometric functions in the forward kinematic computation, the control of such joints is more complex than for translational joint particularly when a configuration has a revolute joint as its first joint or there are more than one revolute joints in succession. If a prismatic joint is located in the parallel direction of a cartesian coordinate frame axes, it dramatically simplifies the kinematic computation.

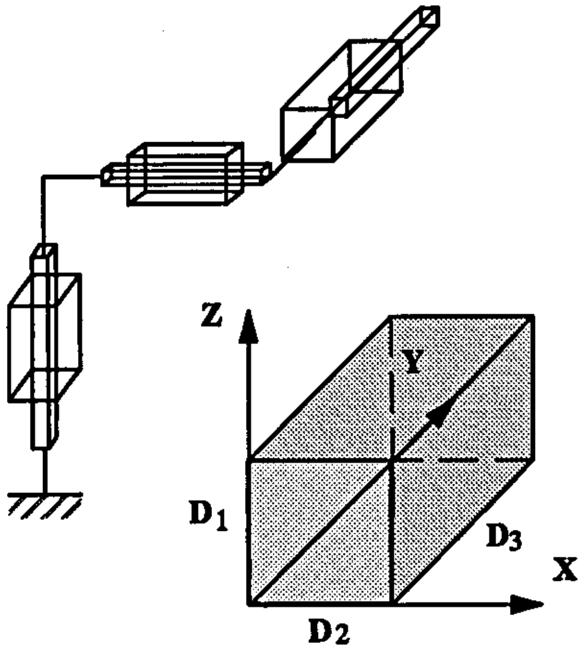
A revolute joint rotates about its axis, Hence configurations which include rotations are inherently more flexible than prismatic ones in the sense that they can usually rotate within a relatively large envelope. On the other hand, the length of a prismatic joint is usually limited because it has a linear mating surface and the mass of the joint needs to be restricted. Also the use of a leadscrew (in electric motor driven machines) can limited their maximum velocity. Thus, configurations with revolute joints typically have advantages of large working envelopes and relatively high speed of movement of an arm's tip.

Figure 4.6 illustrates the twelve valid configurations, their working envelopes and inherent accuracy in each direction. All of these configurations can be included in the library as multi-axes primitives.

4.4.3 Distributed configurations

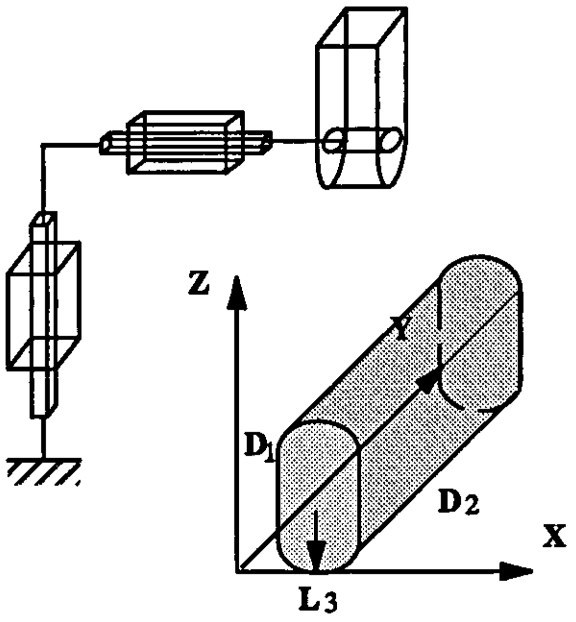
Articulated configurations are used in automating many manufacturing machines because of their ability to facilitate three dimensional motion. This is particularly true in robot configurations. However, potentially distributed but logically coupled mechanisms can be even more widely applied as they can decompose a complex task into several sub-tasks, possibly accomplishing the whole more simply and quickly. In the past, due to lack of suitable complementary distributed control system capabilities, this potential has not been widely industrially realised, nor indeed very widely studied in academic circles.

On considering possible distributed configurations, each individual device can be a single axis, an axis group or some other form of compound mechanism (e.g. proprietary devices). However coordination and synchronization of distributed configurations, to achieve some group (or global) functional goal, will need to be established by appropriate control of each device both separately and collectively. A distributed configuration can have its devices arbitrarily placed within its working environment, with electronic or logical coupling between the individual motions. Thus a distributed configuration does not suffer from the same spatial restrictions as serially chained manipulator systems. The logical relationships will determine the global properties of a distributed system. Such properties are discussed in section 7.6.



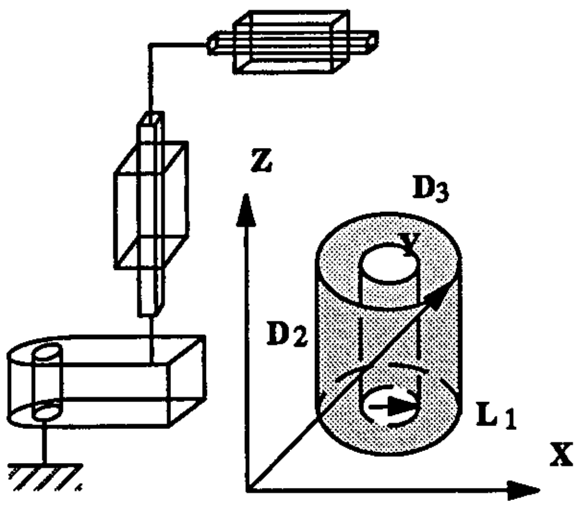
IAOR: R1, R2 and R3 in X, Y and Z axes

(1)



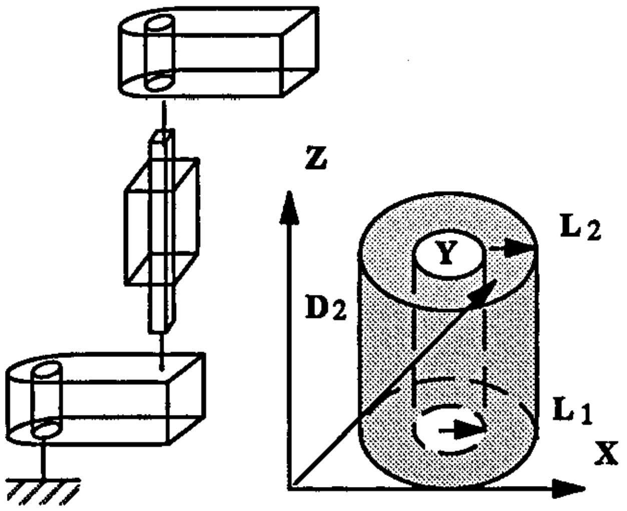
IAOR: R1 and R2 in X, Z and R3*L3 in Y

(2)



IAOR: R2 in Z and R3 to R3*(L1+L3) in X and Y

(3)

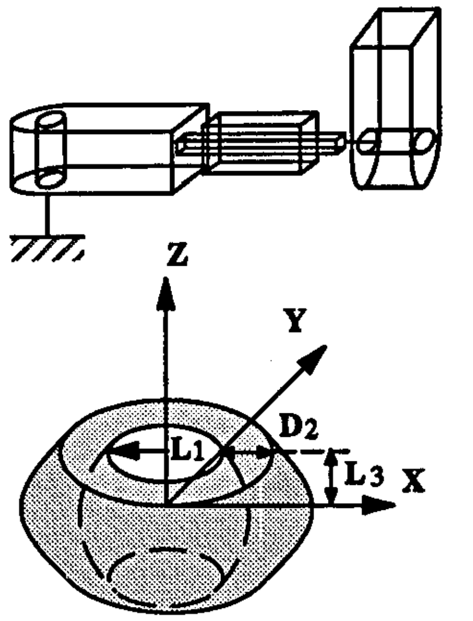


IAOR: R2 in Z and R1*L1 to R1*L1+R3*L3 in X and Y

(4)

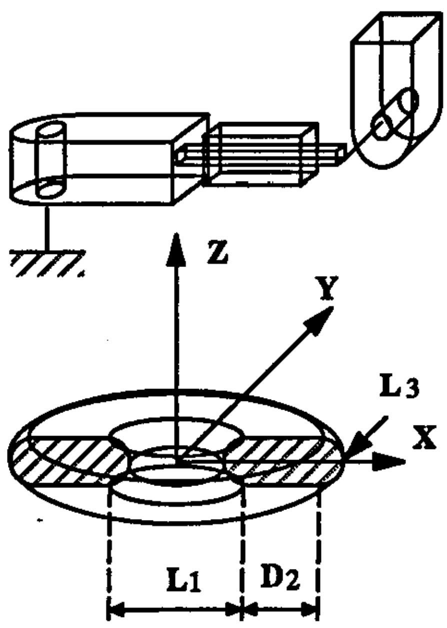
Note: D_i stands for the maximum distance the *i* th axis can move;
L_i for the orthogonal length between the *i* th axis and the *i+1* th axis;
R_i is the resolution of the *i* th axis;
IAOR stands for the inherent accuracy of resolution of a manipulator configuration.

Figure 4.6 An analysis of twelve possible three joint configurations, their working envelopes and the inherent accuracy of resolution (to be continued)



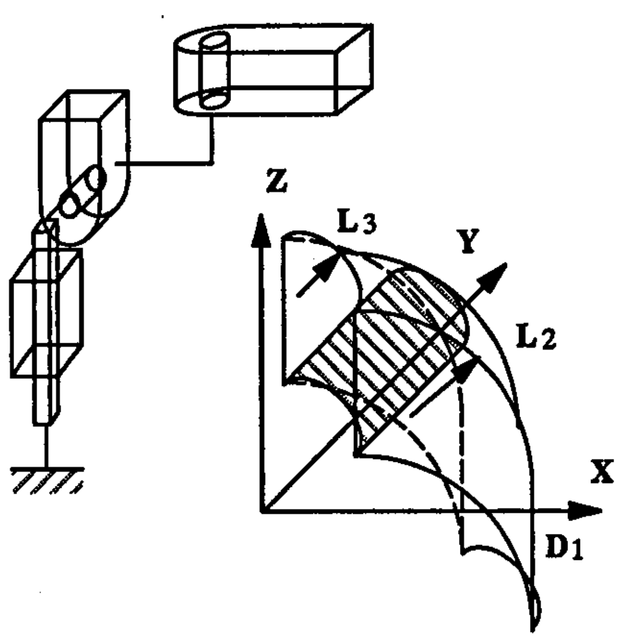
IAOR: R2 to R1*(L1+L2) +R3*L3 in X and Y; R3*L3 in Z axes

(5)



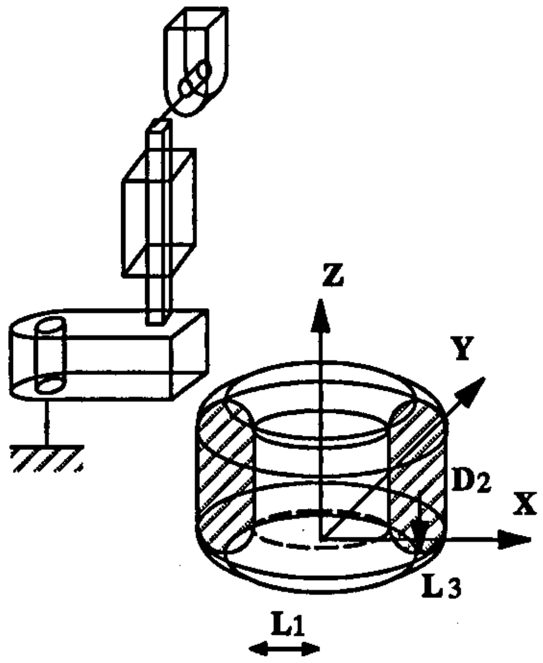
IAOR: R2 to$\sqrt{2}/2$*R1*(L1+L2)+ $\sqrt{2}/2$*R3*L3 in X and Y; R3*L3 in Z axes

(6)



IAOR: R2*L2+R3*L3 in X and R3*L3 in Y ; R1 to the bigger value of R1+R2*L2 and R1+R3*L3 in Z

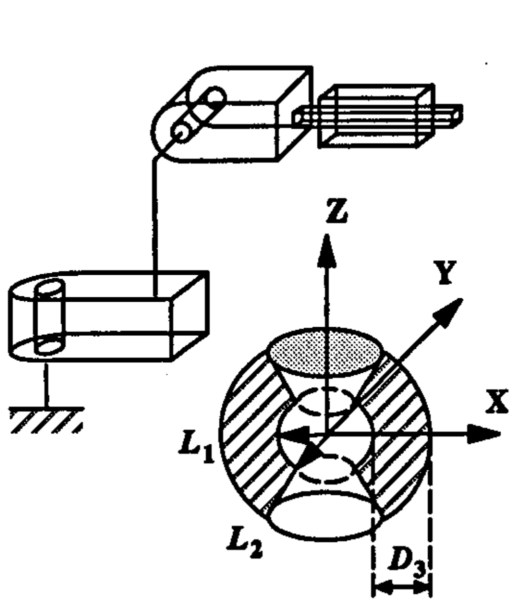
(7)



IAOR: R1*L1 in X and Y; R2 to R1+R3*L3 in Z

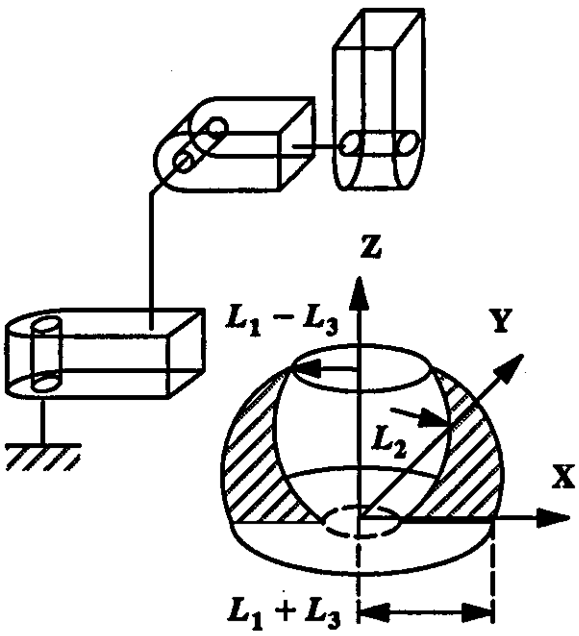
(8)

Figure 4.6 (continued) An analysis of twelve possible three joint configurations, their working envelopes and the inherent accuracy of resolution (to be continued)



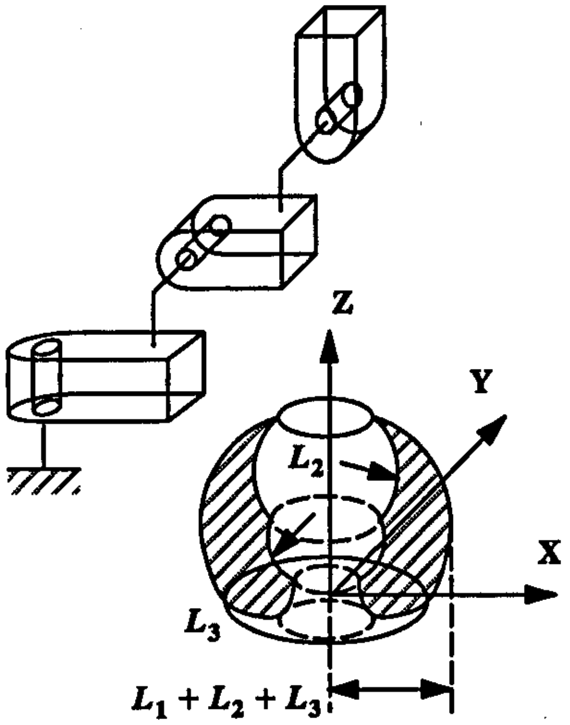
IAOR: R3 to R1*L1 or R2*L2 in X and Y; R3 to R2*(L2+L3) in Z axes

(9)



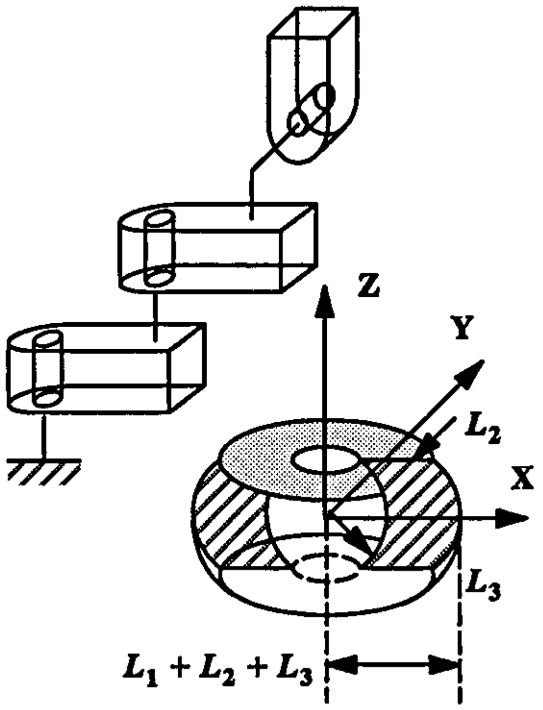
IAOR: R3*L3 to R1*L1+R3*L3 in X and Y; R2*L2+R3*L3 in Z axes

(10)



IAOR: R1*L1 to R2*L2 + R3*L3 in X and Y; R2*(L2+L3) to R2*L2+R3*L3 in Z

(11)



IAOR: R3*L3 to R1*L1+R2*L2 in X and Y; R3*L3 in Z

(12)

Figure 4.6 (continued) An analysis of twelve possible three joint configurations, their working envelopes and the inherent accuracy of resolution

4.4.4 Methods of aggregating (or building) articulated axis groups

The methods chosen by the author for building an articulated axis group include two distinct operations, viz: graphical configuration and the establishment of data structure relationships.

Graphical configuration (or aggregation) can be defined as the process of locating each constituent axis at the right position and orientation. This requires computer assistance or configuration tools which in this project have been built on either graphical manipulation tools or textual/language based (translational and rotational) commands. Constituent axes are aggregated to form an axis group. A single axis can be easily created by selecting an appropriate family of primitives from the library and defining parameters of the axis selected. Scaling of the graphical representation depends on the values input as axis parameters. Graphical manipulation at this stage is at the level of whole axis rather than of two separate graphical representations of the two axis parts. The manipulation covers the scaling of an axis, deleting or adding of an axis, establishing the position and orientation of an axis and viewing an axis from different points of interest. The appropriately prepared set of axis primitives then can be bound together as a whole.

In terms of changes in data relationships, the aggregation of a set of axes leads to the establishment of a set of relationships between data representations of individual axes. The data relationships created in this way are of a parent-child nature, where a typical data arrangement of an axis group is illustrated by Figure 4.7.

Since a child axis is always attached to a moving part of its parent, it is essential to create and maintain the parent-child relationships. The pointers in the axis data blocks are used to

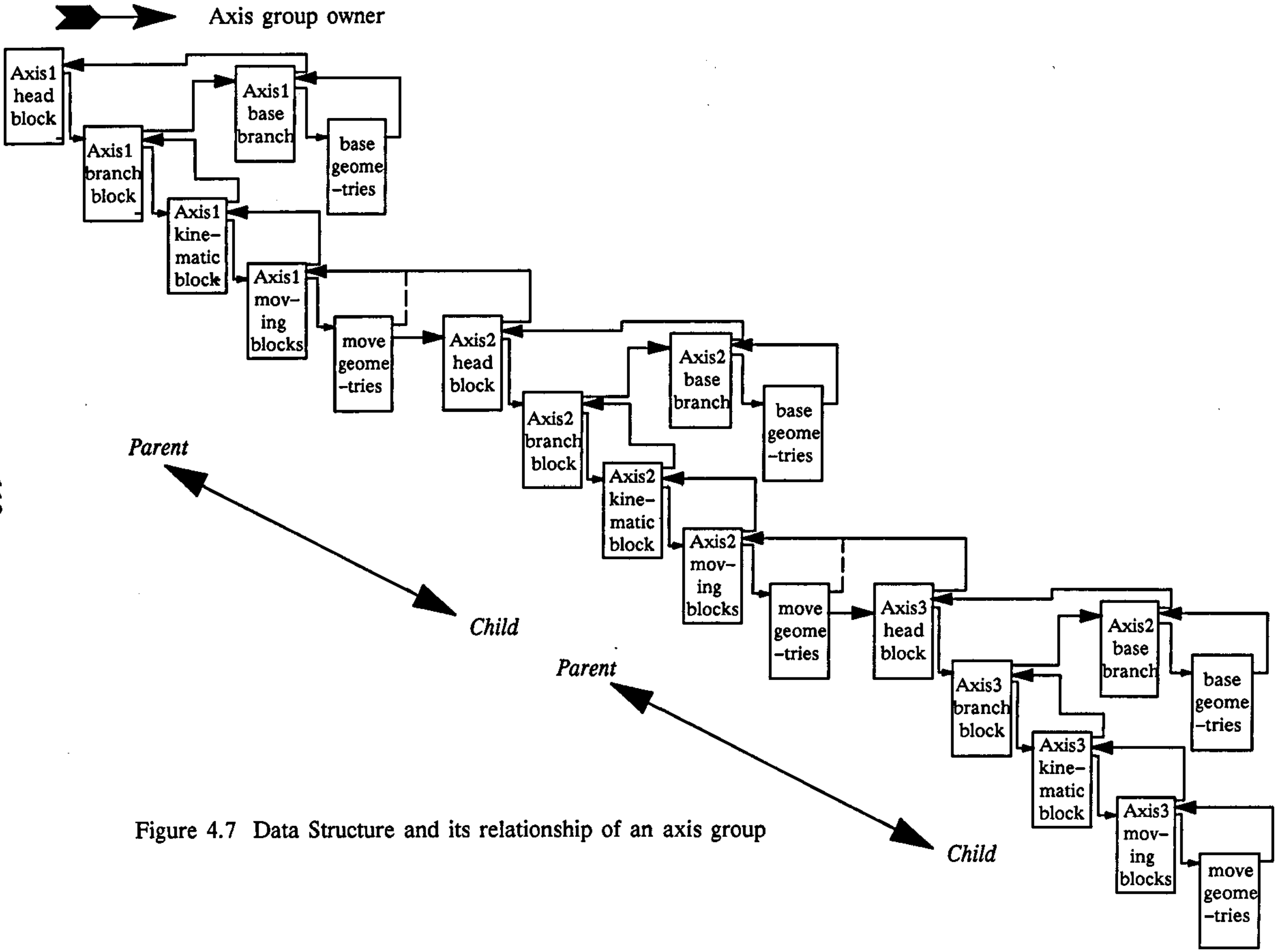


Figure 4.7 Data Structure and its relationship of an axis group

link the child axis to its owner, i.e. the moving geometry of the parent axis.

4.5 Building an end-effector - an example of a higher order mechanism primitive

Serving as an example of building a higher order (more than one axis) library primitive device (by aggregating previously described axis primitives) an analysis of industrial robot end-effectors is outlined and the associated data representations are detailed as follows.

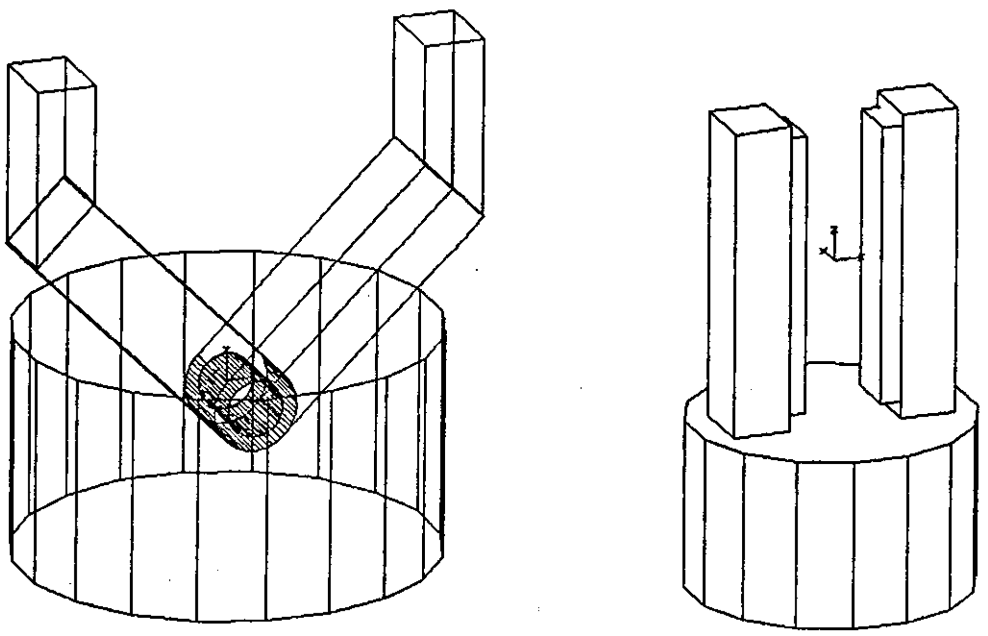
4.5.1 An analysis of industrial robot end-effectors

For a long time, industrial robot end-effector research has been oriented towards the design of replacements for typical human operator hand functions. Consequently most end-effectors take the shape of a two fingered parallel-jaw. This type of end-effector possesses the capability of grasping objects in either one or two-dimensions of the three translational degrees of freedom in the object's space [Kato 1980]. Another research direction has been towards the creation of industrial robot hands and particular effort recently has been aimed at creating dexterous multi-finger robot hands [Li and Sastry 1988].

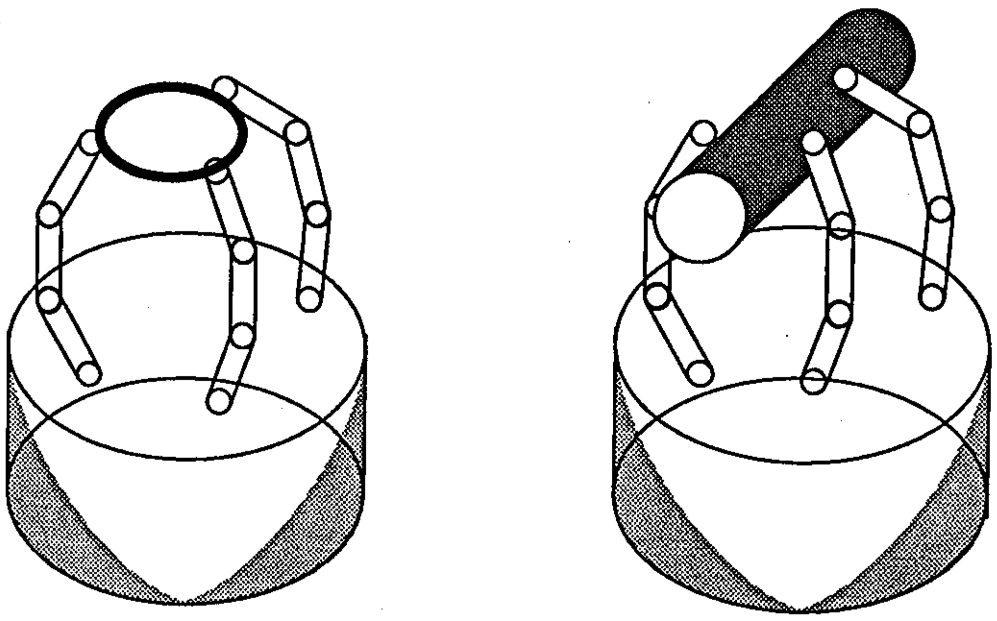
Amongst the two fingered gripper class, typical configurations are dominated by one of the two structures: (i) rotation type and (ii) translational type, as shown in Figure 4.8(a). For the robot hand, some would consider the ultimate universal gripper to be the human hand. However, at present such structures are too complex for industrial use. However, for many applications a three fingered hand should provide sufficient dexterity [McChey and Harris 1986, Ito 1980]. A three fingered hand is illustrated in Figure 4.8(b).

4.5.2 Modelling of industrial end-effectors

The rotation type of Figure 4.8(a), consists of two revolute joints though they are controlled



(a)



(b)

Figure 4.8 Modelled industrial robot grippers (a) and hands (b)

to rotate in a coordinated way. The translational type of gripper is composed of two prismatic joints. Both types of gripper can be treated as two distributed but logically coupled axis groups in respect of modelling. Graphically, two fingers can be modelled by using either two revolute axis primitives or two prismatic ones and typically they belong to the gripper base or wrist. In terms of the data structure, both fingers are children of the base and they form a child data ring for the base. A control relationship between the two fingers should be established and a possible arrangement is discussed in section 7.8.1.

A three fingered hand is much more complex, but by aggregating modules from the library, it is possible to build up such a hand. The three fingered hand can be composed of three open serial chains and a wrist. Each finger consists of three revolute axes. In terms of the hand configuration, three finger chains are distributed on the wrist. Therefore, such a hand can be modelled by using three, 3 degree of freedom serially chained primitives and configuring them into appropriate positions. Similar to the two fingered gripper, the wrist owns three child fingers and a child data ring is formed for the wrist in terms of an internal data structure. However, each finger has a three serial axis chain and each digit or axis can be controlled individually or collectively. Section 6.5 discusses the control issues relating to hands. The graphical models are shown in Figure 4.8(a),(b) separately for a two fingered gripper and three fingered hands respectively.

4.5.3 Working Centre Point (WCP) definition

A working centre point of a machine is generally defined as the centre point of the machine tool or gripper which is normally precisely located in the workspace by the manipulator system to achieve machine and application dependent tasks. Since typical precise motions are required in the computer control of machines, a WCP definition is required, which can enable convenient computation both in terms of modelling and control. For example a tool offset often needs to be accounted for.

A WCP and the associated coordinate frame for the gripper are illustrated in Figure 4.8(a) and (b). The WCP is defined as the point in the centre point of all fingers thus every finger can spend least time to grasp an object. This definition can also simplify the control of an individual finger by using the same control procedure for all three groups of serially chained fingers.

4.6 Library primitives and their management

The geometric primitives included in the library can be classified into the following types:

i) single degree of freedom primitives;

ii) high order manipulator primitives;

iii) user specified high order primitives;

iv) non-motion accessory primitives.

A more detailed categorisation is shown in table 4.4.

The same types of basic axes with different geometric shapes can be used to distinguish various physical joints. Users are also provided with supporting tools to define their own type of axis, should they require some variation from the basic axis representation. As to the functional simulation of associated sensors, approaches are described in section 7.5, the sensory primitives are only listed here as graphical symbols of sensory primitives in the library.

Primitive type	Device type	
Single Degree of Freedom axes	Prismatic axis	Axis moving part in cuboid shape and base in cuboid. Axis moving part in cylinder shape and base in cuboid. Axis moving part in cylinder shape and base in cylinder.
	Revolute axis	Axis rotating part in cylinder shape and base in cylinder. Axis rotating part in compound shape of half cylinder and a cuboid and base in cylinder.
High order manipulation primitives	Two axis group	Axis group in the form of two perpendicular prismatic axes connection. Axis group in the form of vertical prismatic and vertical revolute axes connection. Axis group in the form of vertical prismatic and horizontal revolute axes connection. Axis group in the form of vertical revolute and horizontal prismatic axes connection. Axis group in the form of vertical revolute and vertical revolute axes connection. Axis group in the form of vertical revolute and horizontal revolute axes connection with rotating axes parallel. Axis group in the form of vertical revolute and horizontal revolute axes connection with rotating axes perpendicular.
	Articulated three axis group	There are twelve combinations of three articulated axis chain group. For the axis group constituents description see Figure 4.5.
	Distributed up to three axis group	Axis can be any axis in the category of single degree of freedom axes.
User specific high order primitives	Mechanical grippers and hands	Two fingered grippers with prismatic axes. Two fingered grippers with revolute axes. Three fingered hand with three revolute axes on each finger.
	Component feeder and conveyor	Combination use of machine building primitives described above which are arranged in the same way of mechanical feeders and conveyors .
Sensory primitives	Contact sensors and positional sensors	Graphically a sensor is represented by a 1*1*1 cuboid and its function is associated with a sensory processor to achieve the simulating of the physical sensor. The position of an component in the simulation model can be detected.
	Distance sensors	The representation of this type of sensors is same as last group, however a distance sensor can detect the distance between the sensor and a component in a model.

Table 4.4 The classification of machine building primitive library

Because of the relative complexity of the data structures used to describe individual and combinations of library primitives, a library manager is required to ensure the correct and efficient use of these primitives. Firstly, the manager should automatically create all primitive constituent data blocks. Secondly, all data blocks need to be linked in a specific way within the data structure, thereby describing a primitive geometrically and the assignment of coordinate frames to each geometry forms the central issue of primitive creation. Thirdly, the manager has to ensure that the correct data are assigned to primitives. Since every library primitive has been parameterised, only required meaningful data may be input. It is the manager's task to check the data type and possible value range, provide another chance for input if mistakes are made and finally to assign parameter values when correct input has been made with appropriate dimensioning applied. As the primitive's data structure is a subordinate of its owner, the manager should call graphical display functions to enable visualization of the primitive on a screen (see Figure 4.9).

Since the implementation of the author's work is based on the open version of GRASP, the manager is working between the processed screen layout windows and functional subroutine calls. The association of a multi-window environment to the primitive library makes the design of modular machines easier and more flexible. In the next chapter, a strategy is illustrated to show how modular machine building can be realised.

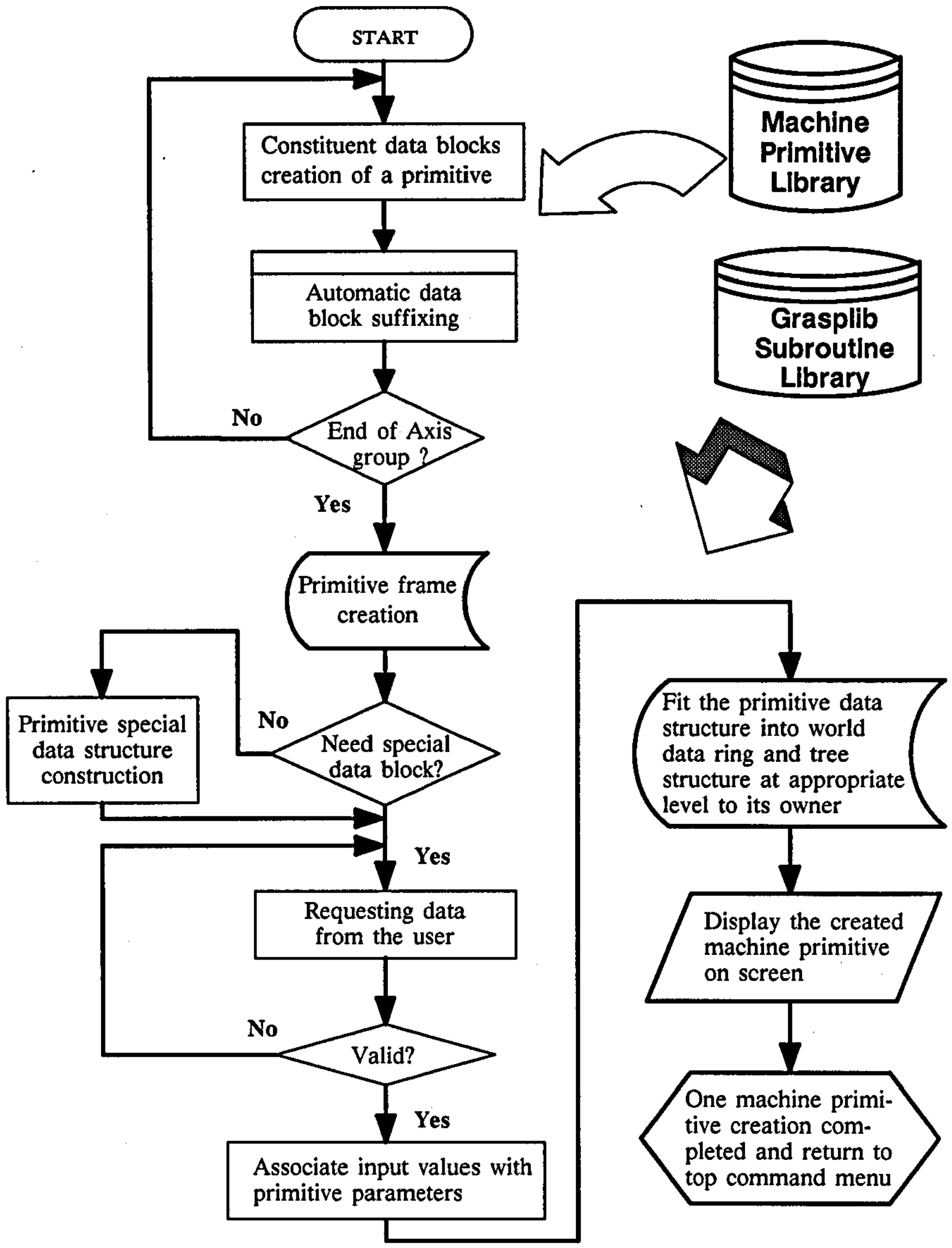

START
Constituent data blocks creation of a primitive
Machine Primitive Library
Automatic data block suffixing
Grasplib Subroutine Library
No
End of Axis group ?
Yes
Primitive frame creation
Primitive special data structure construction
No
Need special data block?
Yes
Fit the primitive data structure into world data ring and tree structure at appropriate level to its owner
Requesting data from the user
Display the created machine primitive on screen
No
Valid?
Yes
One machine primitive creation completed and return to top command menu
Associate input values with primitive parameters

Figure 4.9 The flow chart of a machine primitive creation

Chapter 5 Configuration tools for building modular machines

5.1 Introduction

This chapter develops further the discussion of applications of library primitives and considers the need for computer assistance in the configuration of modular machines. It describes the features of a set of configuration tools for machine aggregation and machine element modification. An interactive user environment is also described. Finally, an open approach to modular machine configuration is outlined to demonstrate inherent capability and limitations.

In this context the term "configuration tools" is used to denote a set of software functional subroutines which are easy to use, are flexible in operation and provide assistance to the machine designer when he/she is building a machine model from primitives selected from the machine library. For each type of primitive, corresponding tools are available to include a chosen library primitive within the machine's simulation environment. The establishment of relationships between individual primitives within the machine's simulation environment is also accomplished by using configuration tools thereby enabling the aggregated primitives to be manipulated as single entity.

A multi-window interactive environment enables the user to communicate in a flexible manner with the modelled machine. The need for manoeuvrability (which is defined in this study as a capability for an end user to manipulate a simulation model), of a simulation model and its environment is a well-established requirement of simulation systems [Chan 1989, Siegler et al. 1987]. Several vendors of graphical simulation systems, particularly those who market kinematic mechanism simulators, claim that their simulators are user friendly and can be mastered within two to four weeks; suggesting that operators need no previous simulation experience [Yong 1989 and GRASP 1988]. However, in practice the lack of flexibility found when modelling mechanical mechanisms in a simulation environment has greatly limited their use, as indeed has their lack of user friendliness, particularly in the case of modelling complex modular machines. With the modular methodology, and the author's implementation of an enhanced user interface to easily manoeuvre machine models, opportunities exist for flexibly building models of modular machines which can be created from an appropriate construction of articulated and distributed devices. This is achieved by the provision of an enhanced user environment involving the parameterization of library primitives.

5.2 Methods used in configuring a modular machine

Currently, many graphic simulators (particularly robot simulation systems) use a machine dependent configuration method as an integral part of simulation tools offered to the designer. Consequently most simulation systems are structured so that a model in the workcell can be extended for a new application in the form of geometric and other functional model building elements (e.g. a robot in a robot simulator, and symbolic machines in a general simulation system). However a single machine (or functional piece of equipment) is structurally fixed once it is created. This means that a user does not have any manoeuvrability in controlling the configuration of the machine or equipment. In this study an open approach to machine configuration is adopted which configures a machine from basic building elements contained within a library. The adoption of such an approach gives the following advantages when designing a modular machine:

i) the capability of building a new machine in terms of either distribution or articulation. In other words, a modelled machine can be enhanced by adding (as required) combinations of motion primitives to achieve a more complex task. The spatial arrangement of these new primitives can be based on various performance criteria (such as minimum working distance, provided that they do not cause any obstruction to other elements of the machine);

ii) the capability of reconfiguring and rearranging an existing modelled machine. In certain industrial situations the need for rapid change-over of products implies that manufacturing machines need a capability for rapid re-configuration. One way of achieving this is to build a modular machine and to logically reconfigure its functional properties to satisfy the requirements of the product change. In the simulation phase, the inherent capability of being able to reconfigure a modular machine can reduce the machine build-time, ultimately eliminate possible errors in reconfiguration and provide an evaluation of the new machine configuration.

In the rest of this chapter, the configuration tools, user interface and the open approach are described.

5.3 Configuration tools for modular machine building

Since when simulating modular machines, the machine's geometry is graphically displayed and its kinematic motions can be visualized and evaluated, two aspects are of great importance in terms of configuration tools. The first is that the simulation model should clearly define the machine's constitution and its graphic and spatial construction. The second is that a model should specify the logical relationships between the constituent devices which form the machine. Generally, a modular machine is composed of several functional devices, i.e. some necessary (or required) number of axis primitives or sensory devices. The primitives will be added into the simulation environment as geometric entities. It is the role of the configuration tools to arrange them in a desired manner and to associate appropriate functional attributes to each device. The author's implementation of these concepts, based on the open binary version of GRASP, is described as follows.

5.3.1 Graphical definition tools

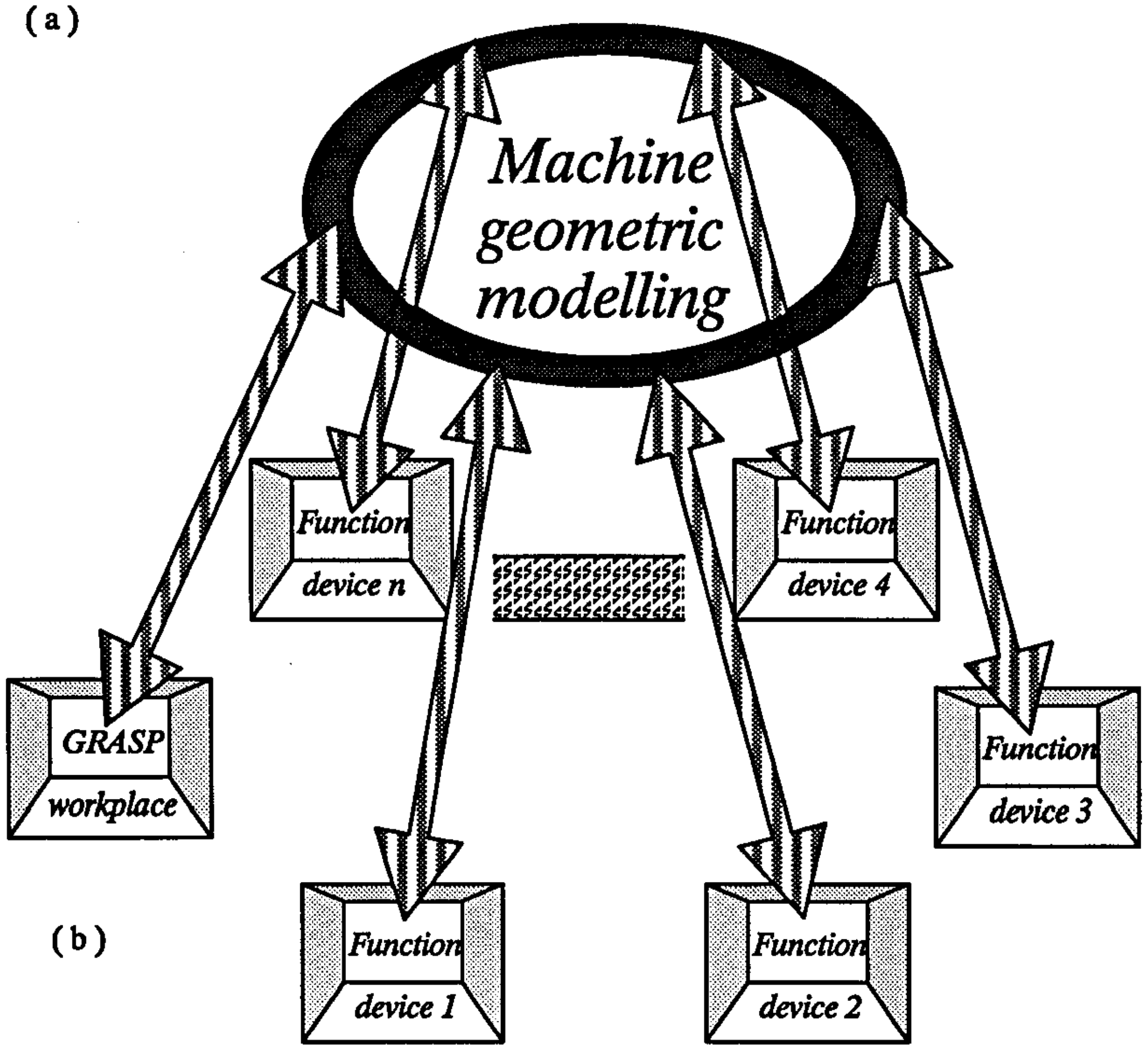
5.3.1.1 Machine definition tools

The machine definition tools created during this study provide a set of software services which describe a modular machine model and define the data block ring and tree structure. This capability has been implemented at the highest possible level of the simulation hierarchy. An example modular machine definition file and its data structure are illustrated in Figure 5.1. The machine definition tools implemented in this study specifically achieve the following:

- supply information about the modular machine's identification name, the machine's functional description, the machine's dimensions, and the number, type and name of its constituent devices;
- create the machine definition data block, this comprising the same number of data blocks as that of the number of devices. The data blocks store machine and device information;
- form the data ring representing the modular machine, starting from the machine definition data block followed by a description of all machine constituent devices (one by one). Here the last device data block points back to the machine definition data block. These device descriptions reside at a common level in machine's hierarchy which is one level below the data block ring formed by machine definition

Modular Machine Definition File	
Machine ID name:	points to machine name block.
Function description:	points to machine function block.
Machine dimension in X:	Real in millimeter.
Machine dimension in Y:	Real in millimeter.
Machine dimension in Z:	Real in millimeter.
Machining parts description:	pointer to machine part description block.
Parts maximum dimension in X:	Real in millimeter.
Parts maximum dimension in Y:	Real in millimeter.
Parts maximum dimension in Z:	Real in millimeter.
Device number:	Integer.
Device *name* 1 pointer:	Integer points to device 1.
⋮ ⋮	
Device *name* n pointer:	Integer points to device n.
Reservation of some data words	Empty.

(a)



(b)

Figure 5.1 Modular machine definition file (a) and its modelling data structure (b)

data blocks and its same level data blocks (e.g. task description data blocks) (see Figure 5. 1(b));

- edit the machine data block including deleting functional descriptions, and generally in recreating and modifying the contents of data blocks;
- edit machine data ring functions, e.g. deleting a data block for a specific device from the ring, adding new data to the ring to introduce a new device.

5.3.1.2 Primitive layout tools

Machine primitives can be created by filling in parameters of a primitive through the library manager. In GRASP, a created primitive geometry usually belongs to the *workplace* which is the head entity at the highest level of the GRASP hierarchy and is located in coincidence with the origin of GRASP global coordinate frame. Whereas the graphic primitive is located at the origin of the *workplace*. The machine primitive can use the *workplace* as a buffer before finally being affiliated to a device. Having chosen a primitive type from the library, the primitive can be created within the simulation environment and can then be located in a desired position by using primitive layout tools. These tools enable the user to achieve translational positioning of the primitive along the X, Y and Z axes of the global coordinate frame (*workplace*), this being an extension of the original GRASP functions. The relational orientation of a primitive was also enabled in this study. These layout tools (in the current implementation) provide the builder of modular machine with a flexible means of achieving machine layout.

5.3.1.3 Aggregation tools for modular machine building

Serially chained mechanisms and distributed mechanisms provide the main focus of this research, and therefore in this context the design of the machine configuration tools, required to aggregate primitives into these two types of mechanisms, are of great importance.

For the serially chained type of mechanism, the aggregation of primitives involves:

- scaling an axis;
- re-dimensioning of axis moving and base geometries;
- adding an axis to a device;
- deleting an axis from a device;
- forming a device of articulated or distributed building elements;
- changing logical relationships between motion primitives within a device or returning a primitive to the *workplace*.

Each of these operations ensures that the primitives are correctly created and a device is "well constructed". In fact, unlike some other kinematic and robot simulators [Yong et al. 1988] the dimensions of each primitive axis can be modified easily and this improves the flexibility of the man-machine interface.

Distributed mechanisms can involve a wide variety of combinations of different library primitives. The constituents of a distributed system can be a single degree of freedom primitive, a higher order primitive, or even a functional device. There is no restriction on the number of primitives, but the complexity of control of these elements will generally grow with the number of these elements.

Distributed mechanisms can be further sub-divided into:

i) devices with up to three distributed (i.e. physically separated) single degree of freedom modules, which are the simple case of distributed mechanisms;

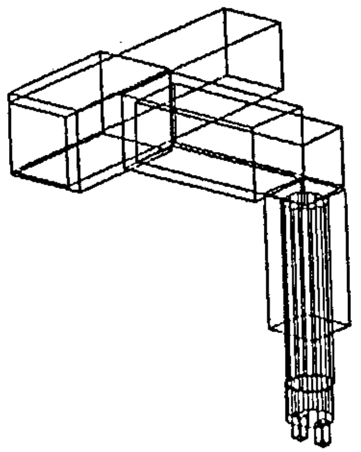
ii) complex devices which could include multiple instances of higher order primitives and multiple instances of (i).

The configuration service supports the creation of distributed mechanisms which fall within both of these classes, thereby providing the user with more choices and significant flexibility. A user can organise his machine in his own way, with the system providing the supporting tools. It is important to re-emphasise that different machine layouts and organizational structures can accomplish the same manufacturing task and conversely that the same machine can be reorganised to achieve different tasks. Thus comprehensive and highly flexible tools are important in bridging the gap between machine users and designers. It can provide flexibility to the machine users allowing them to choose an appropriate machine rather than being driven towards a single machine type or at least having a heavily constrained choice.

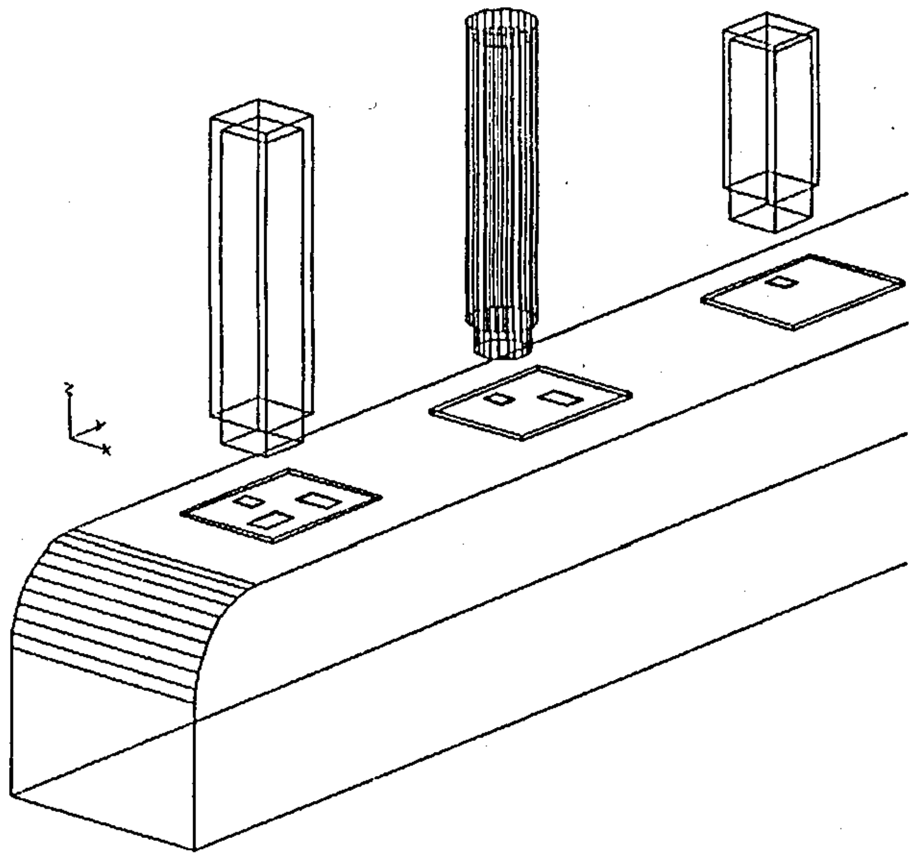
Two of the aggregation tools in the serial chained case can also be used to build distributed manipulators. These are the tools for (a) scaling an axis operation and (b) re-dimensioning an axis. In addition the following services were included to enable configuration of distributed machines:

- adding an axis into a distributed device;
- removing an axis from a distributed device;
- forming a distributed device, by selecting motion primitives from the primitive library;
- re-assigning ownership of an axis from a device to the *workplace*.

An example of both a serially chained mechanism and a distributed device (both being created using the configuration tools) is illustrated in Figure 5.2. In the case (a) of the figure a serially chained mechanism can achieve "pick and place" assembly operation of different



(a)



(b)

Figure 5.2 An articulated axis group (a) and a distributed axis group (b)

components from their feeders; whereas the same operation can be alternatively achieved by a distributed mechanism through decomposing the task into several serial sub-tasks.

5.3.2 Configuration tools for defining logical relationships between devices

The operation of a modelled device also needs to be specified in terms of temporal relationships and motion performance. Here logical relationship definitions, which define the kinematic operation of a device need to be determined. To facilitate such requirements configuration tools were created and their use demonstrated. Based on a classification of motion requirements, three types of logical relationship were seen as being important so that their use was enabled in this study.

5.3.2.1 Sequential logic relationships

Probably the most important requirement in performing a particular operation is that the various elements of a machine or device need to accomplish some pre-defined sequence of sub-operations. In this study therefore it was necessary to specify (or program) such sequences so that the operation of a device can be animated. The transportation of a printed circuit board onto a conveyor and the subsequent insertion of an electronic component is a typical example of an operational sequence. To enable simulation of such sequences an appropriate data block was assigned which can be modified easily. The configuration tools implemented to enable the definition of sequential logical relationships were as follow:

- a sequence creation function, which defines the order of execution. Here a data block is used to associate all primitives in the device with the sequence data block;
- a sequence modification function, which re-establishes the operational sequence of the various primitives of a device through editing the contents of the sequence data block to meet the needs of a new situation;

- a sequence deletion function which removes one operation of a defined sequence and all operations of a defined sequence can also be deleted by removing its data block.

The sequence data block belongs to the device data block and was introduced from word 5 (General Block Pointer) of the device data block. The relationship between data blocks and their main data contents are illustrated in Figure 5.3.

5.3.2.2 Relationships between loosely coupled motion primitives

The motion of distributed primitives need to be co-ordinated, i.e. two or more primitives moving together (but not necessarily starting and finishing at the same time) may need to move concurrently but independently. A simulation of this type of motion is necessary to facilitate the modelling of a modular machine. It provides the means of simulating parallel operation, establishing a loose coordination among motion primitives within a device. Here a device was considered to be composed of two or more distributed motions with loose kinematic relationships (such as precedence relationships) represented either by equations within a data block. specifying a positional relationships. Here the device can be either a mechanically serial chained one or a distributed device. A coupled motion relationship is defined as a library primitive when the primitive is created. The motion relationship in other complex situations is defined at device creation stage in the form of either equations or tables. All information is stored in data blocks which are associated with appropriate motion primitive data representation blocks.

In this study the configuration tools created to define relationships between loosely coupled motion primitives allow:

-loosely coupled primitive motion relationship data block creation, which sets up the one to one relationship between positions of motion primitives. At this stage of the

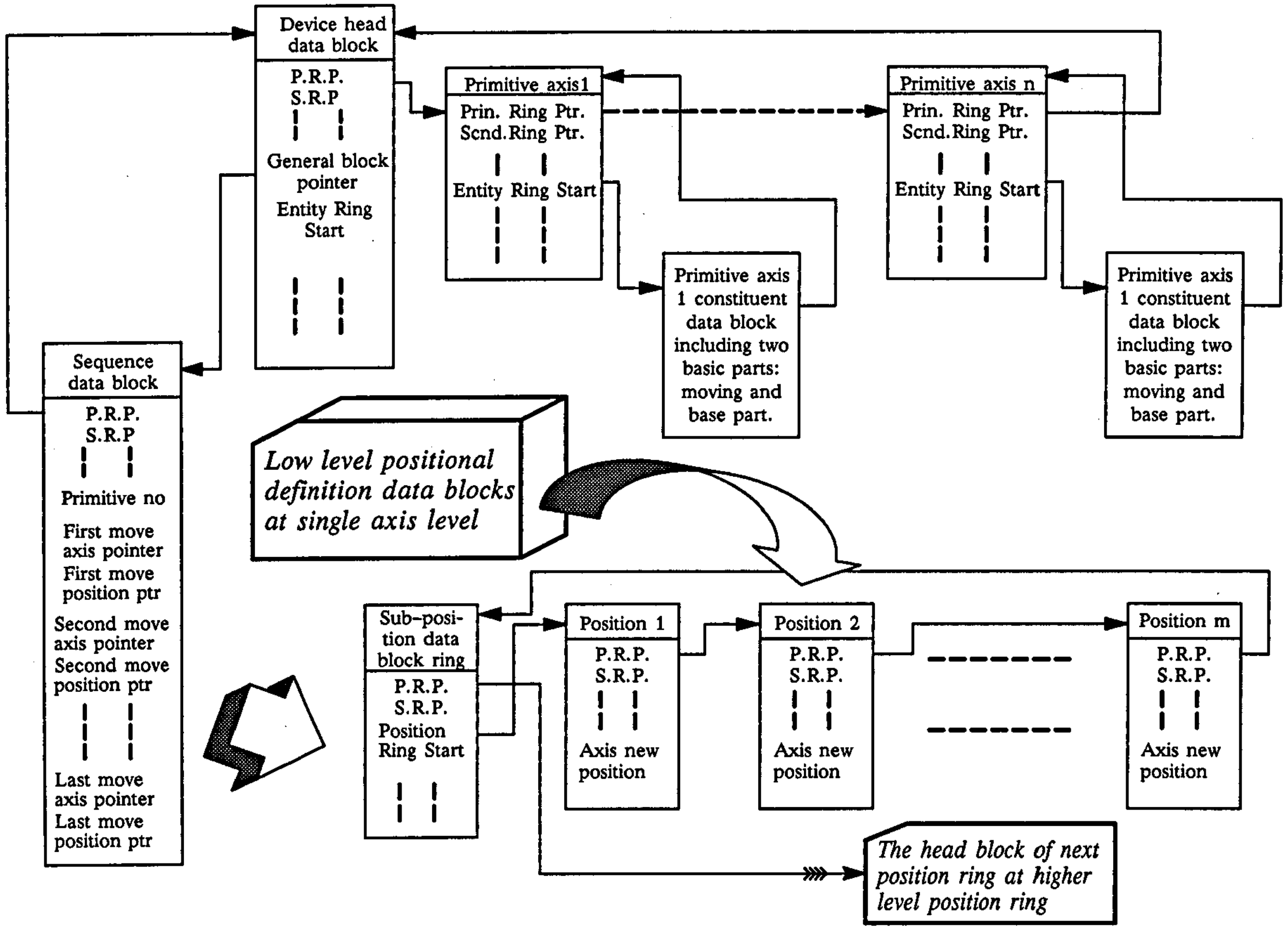


Figure 5.3 The data structure for sequential operations of primitive axes within a device

research, only a one to one relationship is allowed and the one to many relationship is left for future research. This relationship can be exemplified by an operation of metal removing using a lathe with multi-machine-tools where efficiency can be improved by such a multi-tool operation;

- an interface for establishing the one to one relationship in position;
- tools to create data blocks which store the required logical relationships and arrange them in a convenient way for retrieval;
- modification tools to change the content of the co-ordination data blocks;
- deletion functions, which delete unsatisfactory relationships before recreation or permanent device decoupling;

The data block arrangement was designed in a similar way to that for sequential devices, except that positional relationships can occupy several data blocks.

5.3.2.3 Synchronization of primitive motions within a device

Another form of motion co-ordination is required to cater for closely coupled concurrent motions - i.e. synchronization of distributed motion primitives. This type of motion differs from the previous class in as much that several physically coupled or decoupled primitives need to move as if they were a single motion with common start and finish times, whereas loosely coupled primitive motions do not necessarily ensure all motions start and finish at the same time. Thus synchronization facilities were included to facilitate the simulation of closely coupled motion. Here the motion of several primitives can be modelled with common starting and finishing time. The number of motion primitives is not restricted as long as they belong to one device. The device can be either a distributed or serially chained type. A mechanically coupled device where there are two motion primitives can be the simplest case of a software cam - a higher order library primitive of this type is described in section 7.6. In terms of data representation, an additional data block needs to be attached to the device data block. Section 7.6 will describe how all primitives in a device were synchronized at the same time. It was considered that the following tools were needed to establish synchronization, these being implemented in this study.

- a synchronization data block creation facility, which defines the synchronous relationship among device primitives and associates all synchronized primitives with the data block;
- modification tools to update or modify the contents of the synchronization data block;
- means of specifying a "master" primitive, this serving as the reference primitive in terms of synchronization time duration;
- means of deleting synchronization relationships which separating such relationships for establishing other type of relationships.

The data block arrangement employed is similar to that for the sequential logic relationship. However, the data block is distinguished by the suffix '_S' in the data block name, whereas the sequential blocks are characterized by the suffix '_Q' and those of the loosely coupled motion relationship by the suffix '_L'.

5.4 The construction of the simulation model data structure

With the machine configuration tools described in section 5.3, a modular machine simulation model can be gradually built up in the form of a data ring and tree structure. It is important to stress that the use of a hierarchical data ring (at the same level) and a tree structure (between different data levels) has been shown to perform admirably in creating simulation models for modular machine.

Figure 5.4 outlines the implementation of a possible ring and tree structure for comprehensively modelling modular machines. At the top of the hierarchy, a global ring acts as the root ring and heads the second level rings. The global ring usually starts from the geometric modelling ring and is initiated by a general data block. The Principle Ring Pointer (of the data block) points at the next data block on the global ring and the address of the first device is stored in word 5 of the initialization general data block. The initialization block of the geometric model is created automatically by the system, and at the same time, some other initialization data blocks are created for kinematic modelling, e.g. reference coordinate frames, path creation and task programming.

At the geometric modelling level, devices are characterised by general data blocks which are created when several primitives are formed into a functional device. These devices can be created by using the configuration tools described in section 5.3. The other branch trees and global ring are created in a similar way and they can also possess other rings and sub-trees. More details can be found in chapter 6 and 7.

5.5 A User friendly interface for the creation of a machine model

In the open binary version of GRASP (which is the basic GRASP plus some subroutines which enable users to manipulate certain data blocks), a computer screen layout arrangement facility is provided and this gives the research the possibility to evaluate various methods of creating a user interface. Currently, GRASP itself only provides one means of defining a six degrees of freedom robot structure, this being via the use of the proprietary GRASP textual language. Once the robot is defined in GRASP syntax the user will not have any interactive way of modifying the robot in terms of either its dimensions

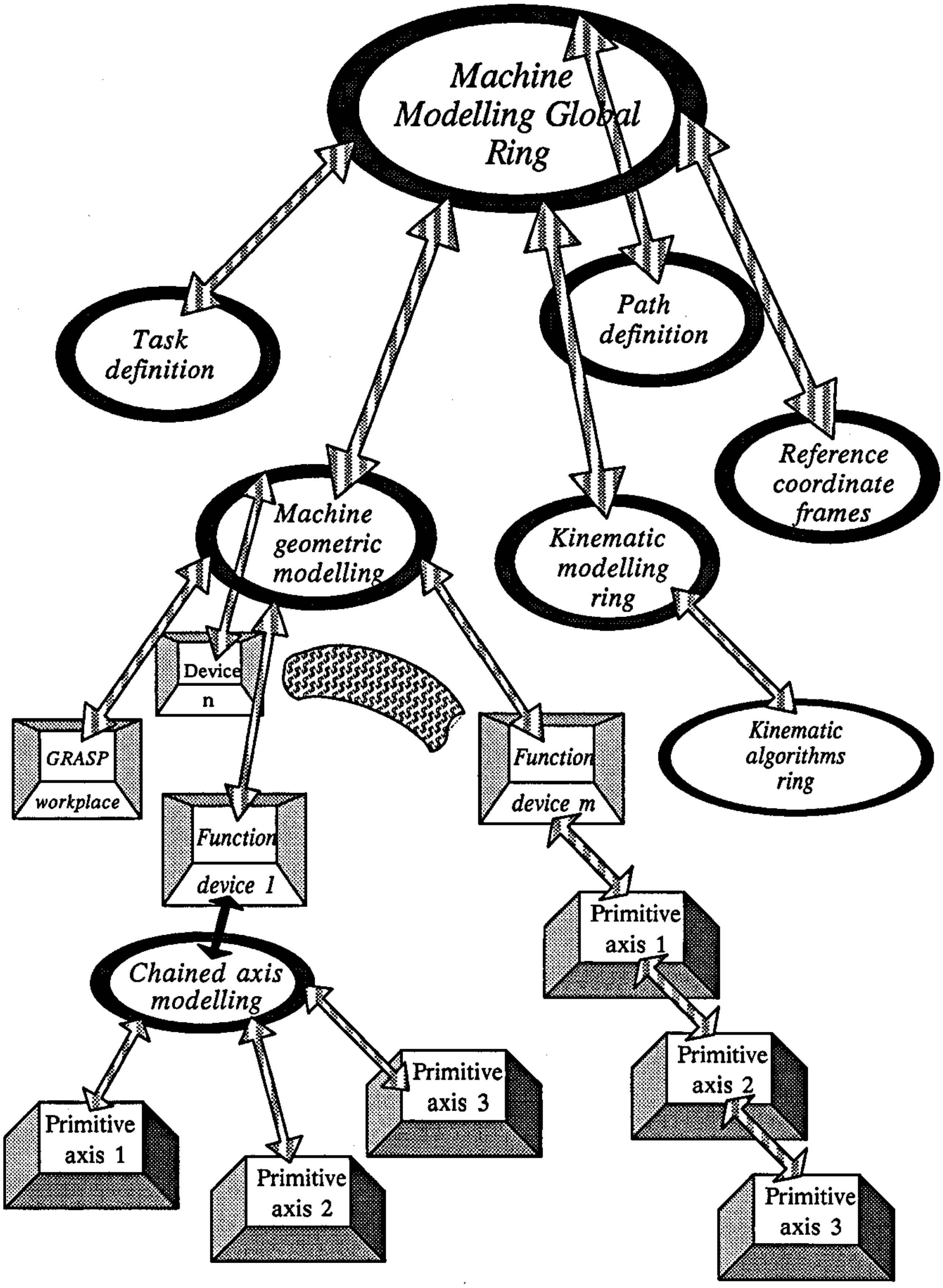


Figure 5.4 An example of a modular machine data ring and tree structure

or its structure. This approach is feasible for robot simulation. However, it is disadvantageous for modular machine modelling due to the wide range of possible motion primitives. The lack of a flexible and user friendly interface makes it difficult and time-consuming to use the system. An attempt has been made in this research to study a comprehensive user friendly interface for modular machine simulation.

5.5.1 Creating machine elements by using default parameters

As described in section 4.6, the library primitives created for modular machine building can be used in many situations as shown in table 4.4. The parameterisation of these primitives enables the system to provide default parameters which can simplify and speed up the process of selecting and aggregating primitives.

There are three types of information required to characterise a machine primitive in this study as discussed in the last chapter, viz: dimensional; spatial; and physical information about a primitive. Default values are provided for all these three types, and a set of tools are also available to modify them in this implementation. This provides a very convenient way of creating machine primitives, i.e. by creating a common primitive then modifying it. The creation of single primitive in terms of data representation can be at the level immediately under that at which device creation occurs, if such primitive is required only once in the machine, or at the top of the machine geometry modelling hierarchy if several same type of primitives are required in several situation. In the latter case, the same primitives can be copied as many times as the user wishes and modified to serve a different purpose in different devices. Copy and paste tools are very important for improving the user friendliness of the system. To begin with the primitive can be copied at the level at which the original primitive was created. Subsequently, the copied primitive can be moved to an appropriate level and a new ownership can be established. At the correct hierarchical level, the primitive can then be modified to meet a given set of simulation requirements.

5.5.2 Creating machine elements based on user specifications

An alternative method of creating a new machine primitive is to call an appropriate primitive creation function from the primitive library and require the user to supply the parameters. The user specifies primitive parameters by using a syntax simulation language complying with a pre-defined language syntax. The implemented system provides an "interpreter" to check the syntax errors and then translate the text following into appropriate function calls which create a primitive. This option is designed for advanced users who can correctly provide the primitive parameter values and further speed up creation. However this option was only partially implemented and currently only axis and device creation is allowed.

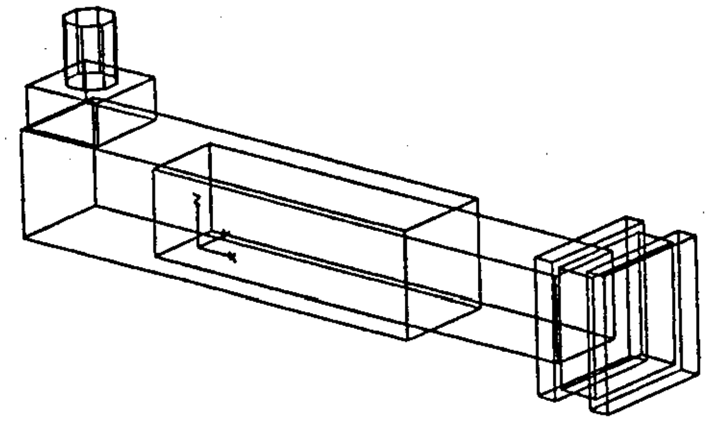
5.5.3 The creation of complex machine elements

Usually, machine elements created through the primitive library possess a simple geometric shape. However for cases where more accurate modelling is needed, a detailed geometric description is required within the model. One approach to this problem is to create some basic geometry which can represent the detailed shapes of the machine elements. This basic geometry can then be "assembled" into a compound geometry which can satisfy the detailed simulation requirements. In terms of data structure, these geometric elements can be arranged in the form of a local hierarchy which can be later owned by a library primitive to allow the precise modelling of a physical machine and its geometry. A possible example of such modelling approach is illustrated in Figure 5.5.

An alternative method could have been to use the solid creation functions provided within GRASP. First a two dimensional face (or outline) is created in the X-Y plane. Then the face can be given depth or rotated about a coordinate frame axis to create a solid. If this solid is attached to the moving part or base part of a primitive, then a detailed representation of that part of a primitive is achieved. Another possibility for primitive creation is to create only an axis frame (i.e. without flesh or with empty geometry). Hence, an axis frame creation function is required, ideally along with related configuration tools to associate flesh geometry to an axis frame. In the implementation of this study, a means of creating an empty axis frame was included and the tools for associating the solids created from a face were also provided. This gives a user another flexible option of creating a primitive, and this facility was achieved by a simple extension of functions for creating solids in GRASP.

5.5.4 Inputting primitive parameters through SunView formatted windows

Currently, GRASP and other simulation packages use a query man-machine interface - i.e. one question for one required parameter method. The advantage of this query method, compared with a text file, is that the user can see the modification interactively. However, the clarity of related primitive information is still quite poor and the efficiency of data input sometimes is extremely low. A mistake at the last step of answering a series of questions means that the user has to do it all again. The related information of a model feature, e.g. the dimension of an axis base part or the constituents of a compound geometry, is beyond the user easy access at one time. There is a need to investigate an efficient and convenient user interface.



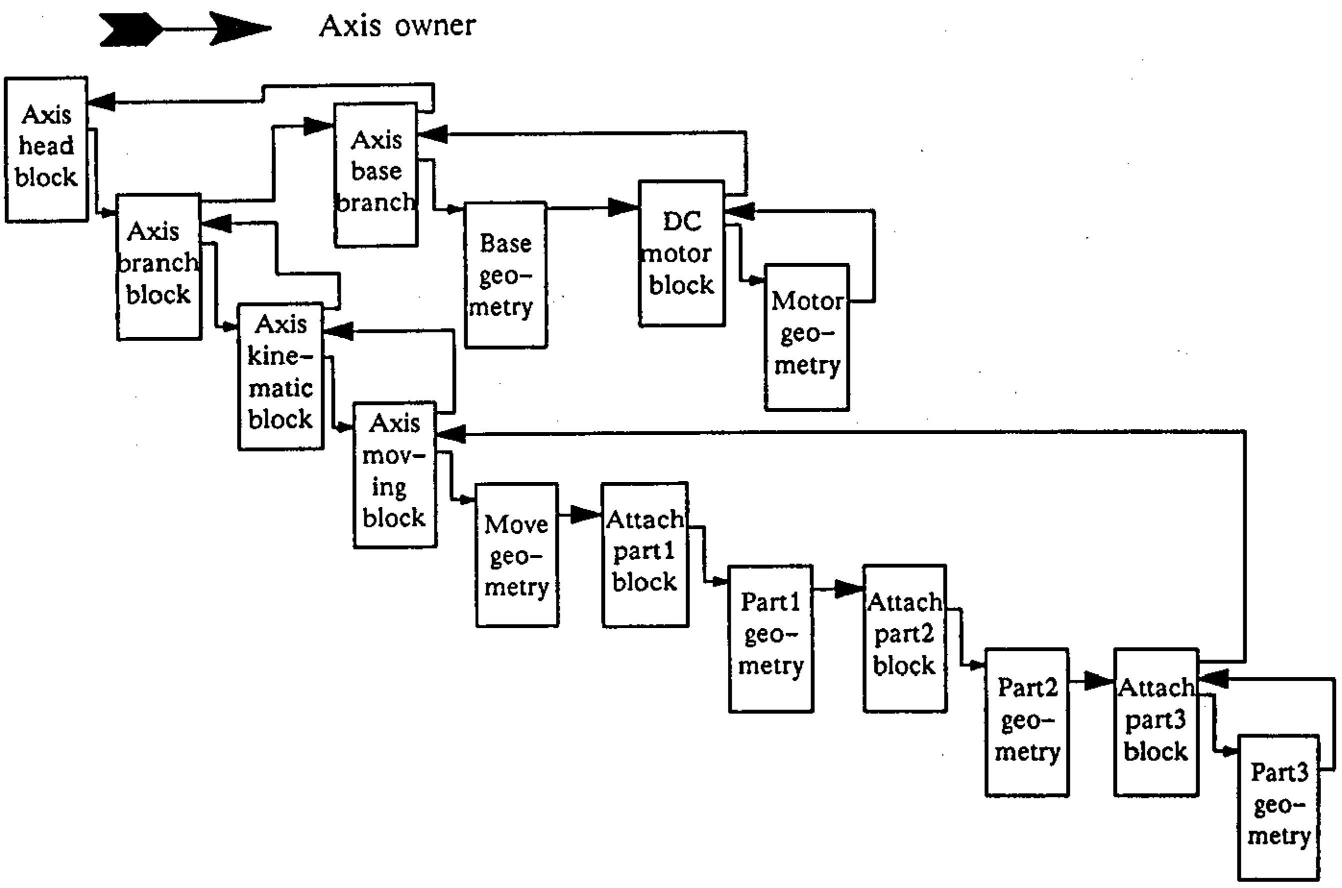


Figure 5.5 An example of a detailed axis primitive and its modelling data structure

The SunView (Sun Visual/Integrated Environment for Workstations) window facility provides possible tools to build an informative, efficient and flexible user interface. It enables users to achieve increased functionality in two major areas: i.e. a run-time system for managing input; and building blocks to support interactive application running. Since SunView is a notification-based system and an event-driven mechanism - Notifier is used in this window form of interface environment, thus the complex management of various events (or inputs) does not rely on the application program. The Notifier reads UNIX input from the kernel and notifies a procedure to perform the formatted high-level events group task. The procedure which is called out or notified has previously been registered with the Notifier. In this application, the Notifier sits between the user's input environment and application objects and related procedures, i.e. primitive creation subroutines, reads UNIX events, formats UNIX input into SunView events, and passes each event to the event procedure through the appropriate window. With this Notifier mechanism, each component or procedure of the user interface program receives only the SunView events the user has directed towards it. The burden of managing a complex, event-driven environment is shouldered by the Notifier. Figure 5.6 describes the user input, the Notifier, SunView objects and user interface application procedures.

Among four building blocks of SunView, which include *canvases* for drawing, *text* sub-windows for editing text file, *tty* sub-windows for running programs and *panels* for user input interface creation, panels are created to build an input form which can be opened if a user wants to modify or create a library primitive. The primitive related information can be obtained and modified in one page of its property form rather than one question for one parameter and so on. The procedure of creating a library primitive property form and

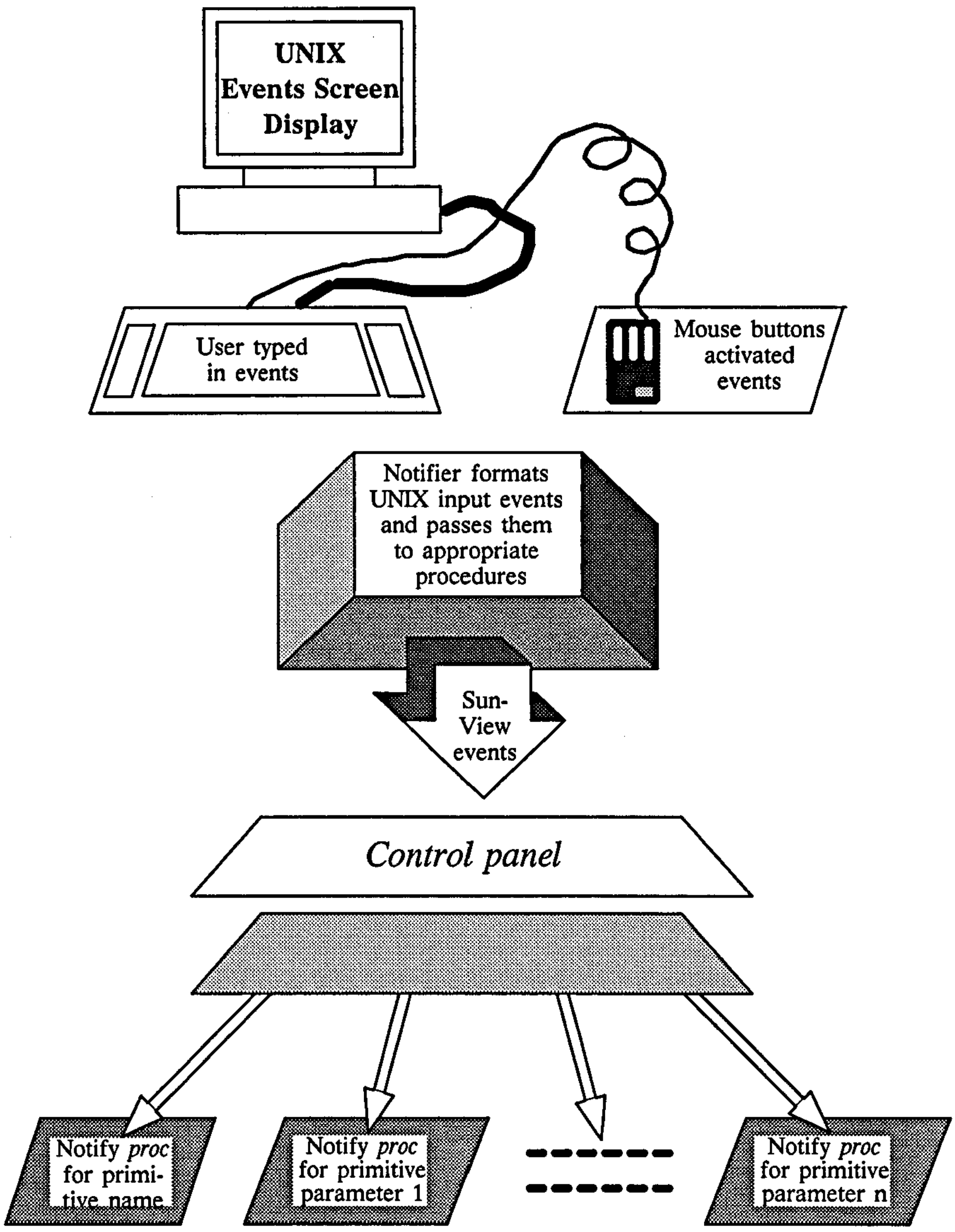


Figure 5.6 Interface of SunView and modular machine simulation system

interfacing the form with the simulation of the modular machine is outlined in Figure 5.7. In this research SunView was chosen as a window facility because it was available to the author. Since it is a proprietary windowing system, standard ones such as Motif and X-Windows can be used to provide the same facility.

The whole operation of primitive parameter input or modification can be illustrated as three sub-operations, as follows.

(1) The user specifies the specific machine primitive and the aspect of creation and modification. The simulation system then searches for the corresponding parameter values based on the user's specification. A SunView window with the appropriate primitive property format is then opened and the current parameter group values are displayed in corresponding panels as parameter default values. If the primitive is not an existing one, zero values are provided for new primitive creation and the whole set of primitive property forms is provided;

(2) After the primitive property form is created on top of the machine graphic simulation windows, the user can input and modify the displayed parameters. Only valid type of parameter values is accepted by the SunView panels, and if not valid then a partial repetition of the modification is necessary;

(3) The input values through SunView primitive property forms can be searched through pointing to the physical addresses of these parameters and transferred to the simulation model to update the new information on structure and dimension. A newly created or modified model can be viewed at this stage.

5.5.5 Modification tools for a machine and its primitives

With the SunView primitive property form, it is also easy to build up machine primitive

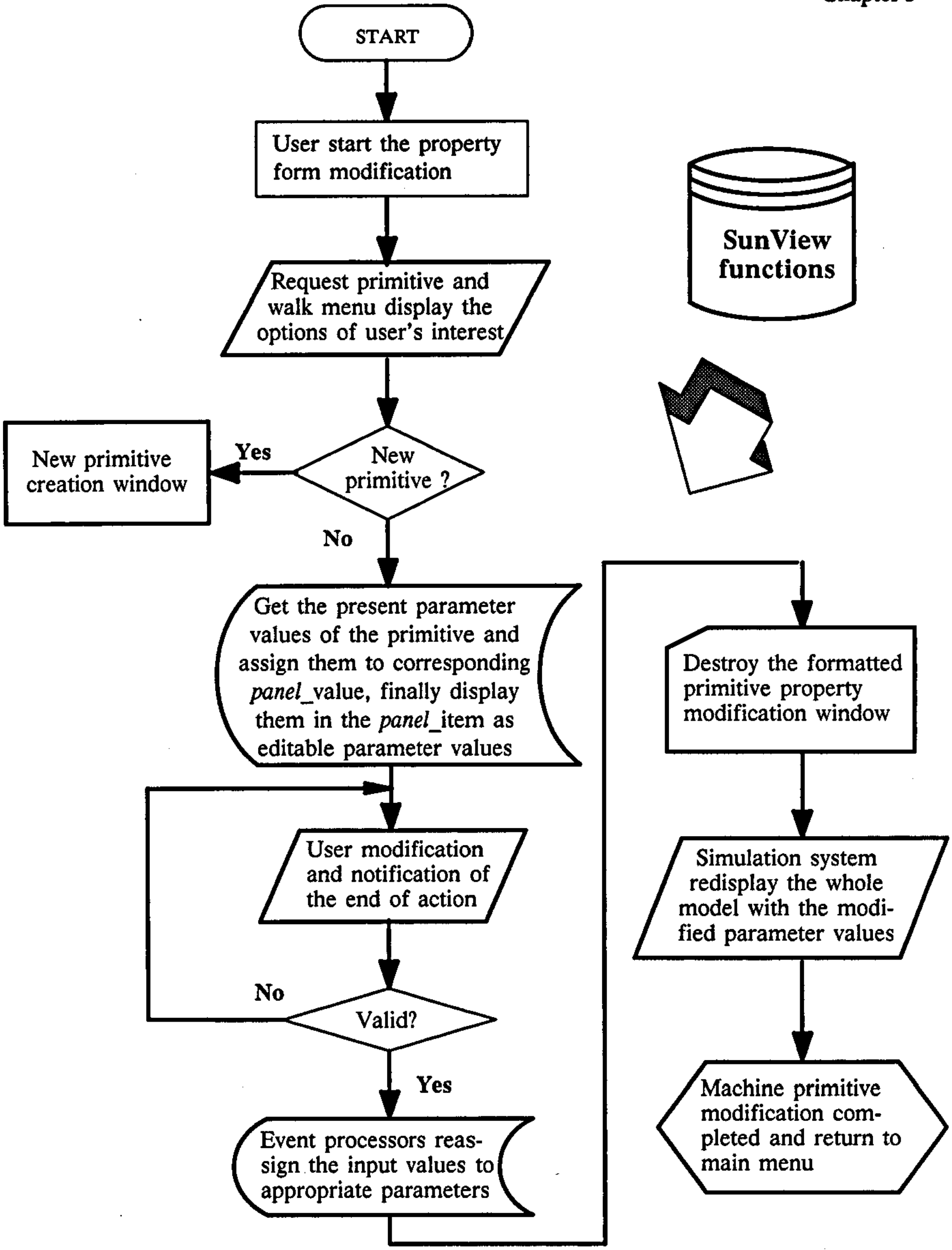

START
User start the property form modification
Request primitive and walk menu display the options of user's interest
New primitive ?
Yes
New primitive creation window
No
Get the present parameter values of the primitive and assign them to corresponding *panel*_value, finally display them in the *panel*_item as editable parameter values
User modification and notification of the end of action
Valid?
No
Yes
Event processors reassign the input values to appropriate parameters
SunView functions
Destroy the formatted primitive property modification window
Simulation system redisplay the whole model with the modified parameter values
Machine primitive modification completed and return to main menu

Figure 5.7 The schematic of using primitive property form for machine primitive modification

parameter modification tools and window forms. All primitive parameters can be easily modified by re-inputting new parameter values and re-creating the primitive. At the primitive level, the following modification tools and window forms were partially implemented.

(1) Primitive geometry modification tools, which enable the user to search for a correct geometry. These provide the current geometric parameters in an editable form, retrieve the new parameter values from the SunView window and put them into the data block representing that geometry;

(2) Position and orientation modification tools, which again obtain the current translation and orientation information along and around the local direction of the frame, display them in each panel item in an editable form, modify the value and assign them into a geometry entity block containing the new position and orientation information of the geometry within global model;

(3) Kinematic constraints modification tools which can be used to get current primitive kinematic values, renew the constraints and store them in the kinematic data block of the primitive.

For the machine level appropriate modification tools have already been described (in section 5.3.1.3). However a detailed "snatching" operation is illustrated in Figure 5.8. The system has to know which axis primitive is going to be removed from its current owner. After the initial query to the user, the modification function starts (from the machine geometric modelling ring) searching for the device which owns the axis primitive. Once the primitive has been found, its block address and its owner block address will be known. The axis primitive can then be removed from the device and the owner primitive. Thus the other

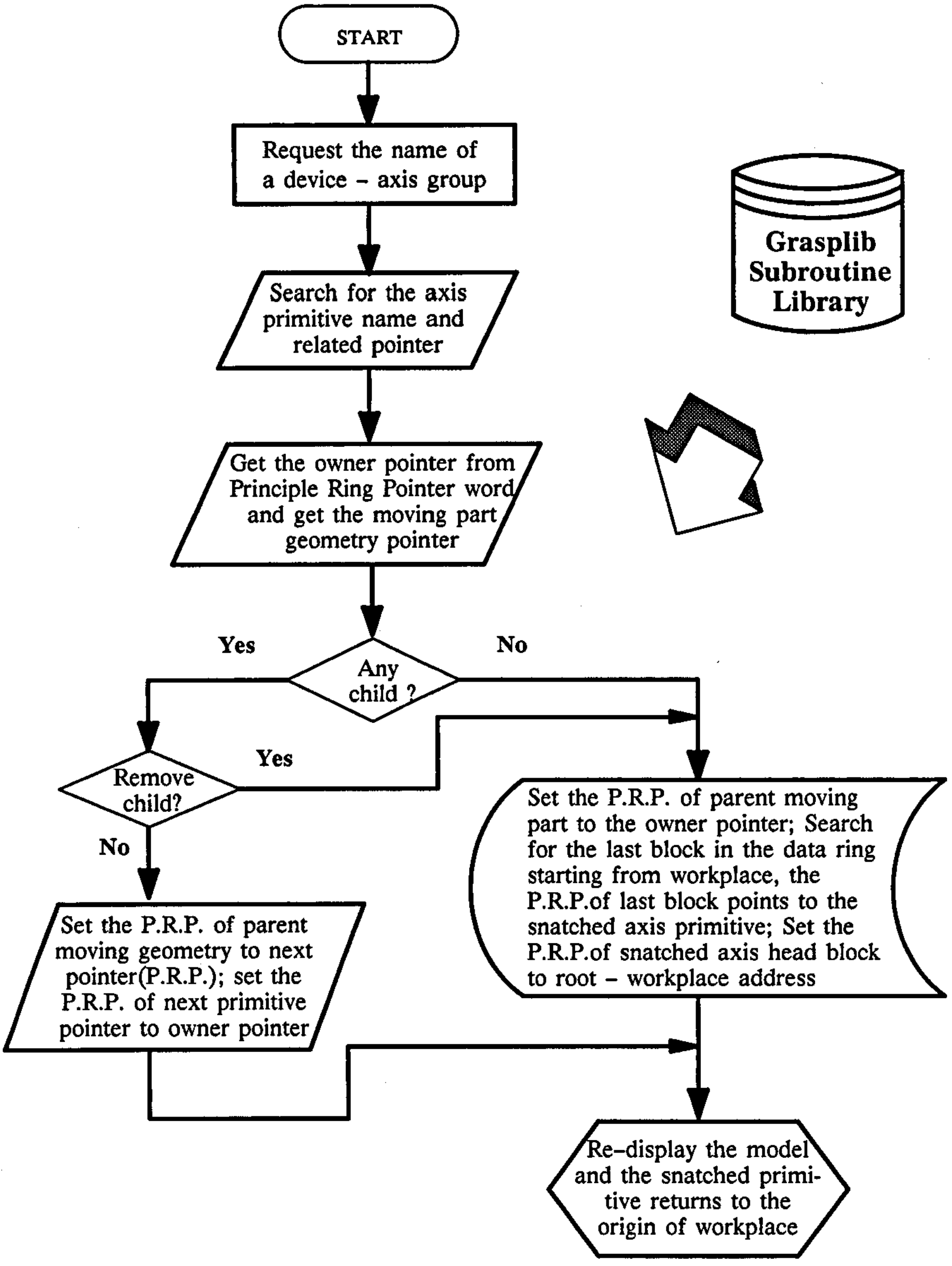

START
Request the name of a device - axis group
Search for the axis primitive name and related pointer
Get the owner pointer from Principle Ring Pointer word and get the moving part geometry pointer
Yes
No
Any child ?
Yes
Remove child?
No
Set the P.R.P. of parent moving geometry to next pointer(P.R.P.); set the P.R.P. of next primitive pointer to owner pointer
Set the P.R.P. of parent moving part to the owner pointer; Search for the last block in the data ring starting from workplace, the P.R.P.of last block points to the snatched axis primitive; Set the P.R.P.of snatched axis head block to root - workplace address
Re-display the model and the snatched primitive returns to the origin of workplace
Grasplib Subroutine Library

Figure 5.8 The schematic of snatching a primitive operation

device constituents can be rearranged into the form of a new local data ring and tree structure. If the primitive which is to be removed from the device owns another axis primitive, the user has to inform the modification tool if that primitive is to be removed as well. If the owned primitive is also to be removed, then all removed primitives are returned to the geometric data ring and the first removed primitive will point to its new owner rather than its original child. After the searching operation, the removed primitive returns to the root owner - typically the *workplace* in this implementation in terms of both ownership and position.

5.6 Name and data block address based searching methods

Due to the large amount of data manipulation required during machine simulation (including: data modification; primitive addition or deletion; and machine reconfiguration), an efficient way of finding the correct data block and word is of great importance. This efficiency of searching method will be dependent on the data structure of the machine model. Once again it has been shown that the use of a multi-layered ring and tree data structure to represent modular machines provides an easy to use and clear data structure which can allow a searching mechanism to find the required data block quickly.

Data block searching in this study is based on an identification of a data tree branch, from the top level of the hierarchy according to the data manipulation category. Once the correct branch is found, the second level data ring address can be obtained from the branch data block. The searching advances to the second level of the hierarchy and this is automatically done by the simulation system. From second level downwards, the name of a manipulated object is used in the data ring search at that same level. Since all objects within the model are identified uniquely by their names, the search of an object name therefore is the only way of locating the data block address of that object. Therefore, at this stage an object name is required by the simulation system for the object data block address. If a further downwards search is necessary to find the third level primitives, then the primitives address can be found from the searched second level data block contents. Once the lower level data block address is found, data manipulation of the block can be easily carried out.

Using the conventional GRASP modelling system the modelling origin *workplace* is treated as a pseudo functional device in the modular machine simulation system. Under the *workplace* various machine primitives are usually created and re-located to their target positions. The data manipulation of these primitives should be carried out under the *workplace* sub-tree. Since the configuration tools enable the user to build up this type of data ring and tree structure, the user should always realize this structure in this particular implementation and this will enable an easy manipulation of the data ring and tree structure. However, a warning system is provided to prevent unstructured data manipulation.

5.7 An open approach towards machine elements creation and configuration

5.7.1 The needs to create an open structure for simulation of modular machines

In the assembly industry, common sub-operations of assembly tasks include: pick a component from a device (usually a feeder, conveyor or component magazine), assemble the component in a desired way (often involving accurate placement and orientation, followed by some fixing operation, some form of inspection operation, withdrawal of the assembly (or sub-assembly) to some output devices (e.g. pallet, conveyor, etc.). A very common operation in assembly requires a so called "pick and place" motion which is generally achieved by a mechanism through picking a component from its holder and placing it into its other mating part. Among these non-processing operations, the most important task is to relocate a specific component at another target position with a defined orientation. The flexibility and efficiency in achieving these transportation related tasks are of great importance to machine designers and users. The various device arrangements can lead to a better solution to a modular machine design. Therefore there is a need to create an open structure which enables the user to build up complex devices and operations in order to find optional designs. The current single primitives and their configuration tools ensure that the user can configure their own machine in a simulation model.

Although modular machines are more usually found in the assembly application area, the concepts can be extended to the other manufacturing application areas. Hence the simulation should be similarly extendable. Since the system adopts the library concept together with modular methods, the machine user can utilise library primitives and configuration tools to construct his special primitives. Once the primitive is parameterised, it can be stored in the library for future repetitive uses.

5.7.2 Extendability of the open approach

A simulation system for modular machines should be versatile in terms of being able to cope with various applications and complex operations. The extendability of the modelling approach enables enough flexibility to deal with application variations. It is the adoption of an open approach towards machine building (which is inherited from the data ring and tree structure) which makes the modelled machine highly extendible. Though at each level the ring of data blocks is in the closed form, the configuration tools enable the user to add new data blocks or primitives in a user friendly way. Such extension tools are available at different hierarchical levels.

Although the configuration tools are described in section 5.2, a detailed description is given here to illustrate the ease with which device addition can be achieved (see Figure 5.9). A device should first be created by forming a primitive group under *workplace*. The user then has to locate this group at some desired position and orientation. The head device block of the ring to which the device is to be added needs to be found via a search, this being the responsibility of the configuration tools. Once the address of the head device is found the device description can be added to the end of the ring.

The systems extendability comes not only from the inherent nature of the modelling data structures but also from the concept of creating a primitive library. At the top level of the hierarchy of the implemented simulation system, five main categories for assembly associated manufacturing were created. However, arising simulation categories can be added into this ring in order to simulate other manufacturing applications. This should be easily carried out using the tools provided. Once the new categories are added, the association of the new class and its branch with the other parts of the system is again via a data block address. It is possible for the user to use some system subroutine tools to write application dependant programs.

5.7.3 Open structure for machine element creation and machine configuration

The foregoing discussion illustrates that the data structures used for modelling in the

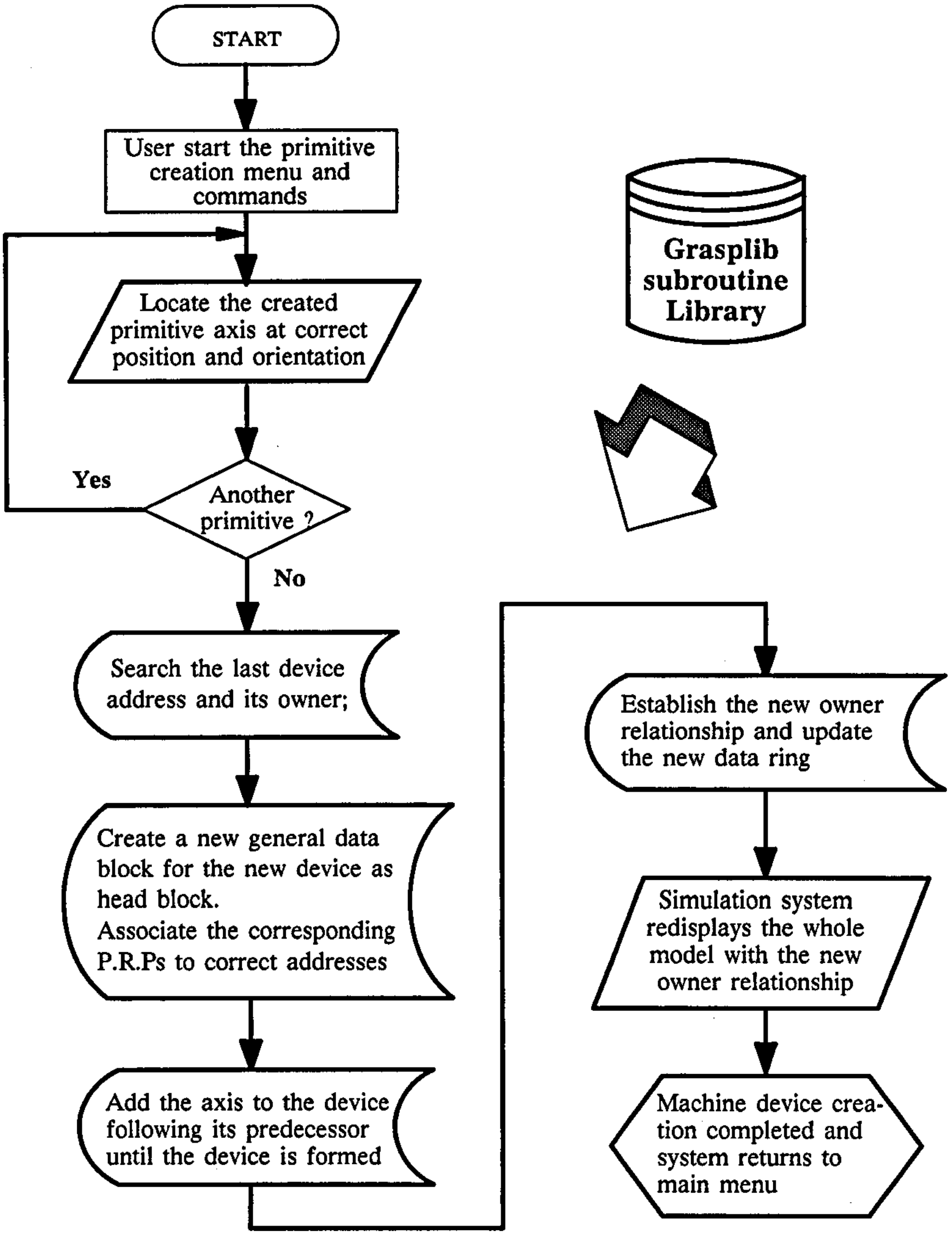

START
User start the primitive creation menu and commands
Locate the created primitive axis at correct position and orientation
Yes
Another primitive ?
No
Search the last device address and its owner;
Create a new general data block for the new device as head block.
Associate the corresponding P.R.Ps to correct addresses
Add the axis to the device following its predecessor until the device is formed
Grasplib subroutine Library
Establish the new owner relationship and update the new data ring
Simulation system redisplays the whole model with the new owner relationship
Machine device creation completed and system returns to main menu

Figure 5.9 The schematic of forming primitives into a device

simulation system have a further extremely useful feature - namely their openness in enabling machine construction. Most graphically based machine simulation packages provide efficiency in the respect that they can create, simulate and evaluate a model with a fixed structure quickly. However their inherent flexibility when building up a model very much depends on restrictions imposed which limit the possible machine configurations, typically to a class of manipulator system. In some graphic simulation packages, the built-in machine models often make workcell design user-friendly provided that existing machine models or models of a similar class are involved. However it is very difficult even impossible to simulate machines which do not correspond to the supported class of models.

In this study of modular machine simulation, single degree of freedom machine building elements are standardized and parameterised so that they can be flexibly arranged, in terms of either the spatial relationships or the shapes of a moving part and a base part. As a demonstration, Figure 5.10 illustrates some more common machine primitives used in the field, which have been created and associated using the simulation system. At the lowest level of solid modelling (i.e. the GRASP basic geometry primitive level) the user can use the GRASP geometry primitives to construct any non-powered machine device. Most of these types of device are mainly concerned with the dimensional shape of the physical devices so that accurate dimensional modelling provides the useful information to govern powered motion, e.g. as planning and collision detection position data which may be of vital importance for model evaluation.

The openness of the approach also lies in the fact that the machine primitives and their configuration are open. Most robot simulation systems model a robot by creating it as a fixed entity, so that any addition or deletion of an axis or other primitives is very difficult.

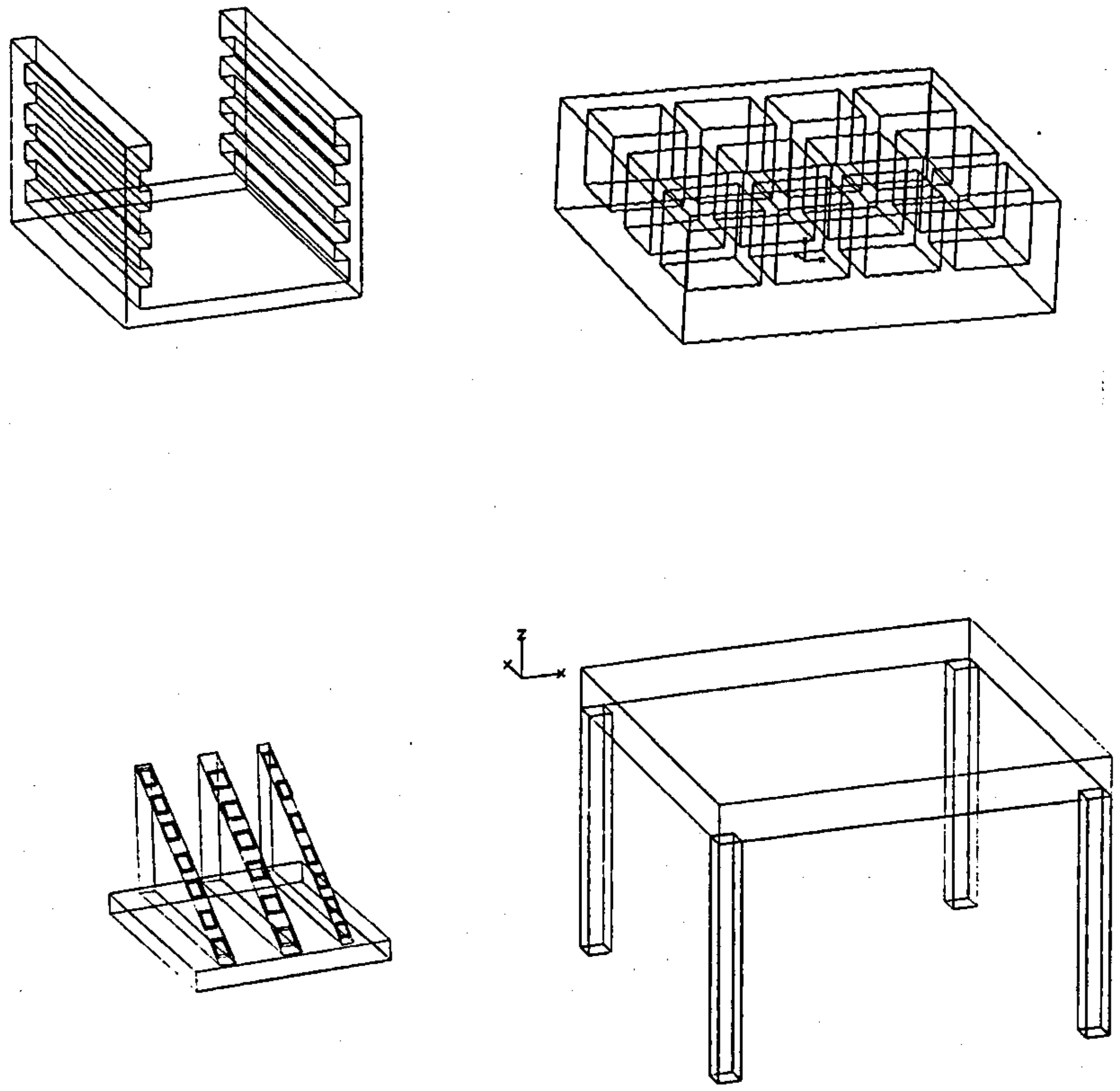


Figure 5.10 Some of non-power driven machine primitives

GRASP is a very typical example. It only allows a user to create a robot by defining the structure and its flesh using a GRASP text creation form. Some aspects of structural and shape modification prove impossible at an interactive level. However the modular machine simulator enables the user to create machine primitives and modify them flexibly via the user friendly interface which is established specifically to support modularity.

These features are crucial when attempting to create a machine design, simulation and evaluation system. They potentially enable the machine users to design a machine based on information concerning their products and their manufacturing operation requirements. Potentially it can provide very strong support tools for machine end users and could bridge the gap between machine designers and users. The lack of communication between and understanding of design as opposed to manufacturing problems may disappear. The second benefit of this type of design and simulation is that the machine user can reconfigure an existing machine to accomplish a new manufacturing task. As basic machine primitives, for instance transporting primitives, are commonly used in various machines, they can be easily converted into a part for another machine primitive. A simulation exercise centred on the feasibility of such a conversion can provide useful information for machine builders. The third benefit is that the flexibility of machine configurability is well maintained due to the re-use of the library primitives and configuration tools. The availability of a comprehensive set of configuration tools ensures that the user has access to the machine at any level of the model's hierarchical data structure. The last advantage is that the model has a simple and clear data structure and hence it is easy to maintain and manipulate.

Chapter 6 Machine kinematic modelling

6.1 Introduction

A further key issue in modelling a modular machine concerns kinematic modelling aspects, which are required to define spatial relationships as a function of time. Simulation of a modular machine (which includes machine graphical modelling, machine configuration, machine kinematic modelling and task programming based on use of the machine model) can lead towards an integration of machine design and evaluation activities.

Powerful computer graphic facilities provide very useful tools not only to graphically model the modular machine but also to animate motion within the machine system. Animation, using computer graphics, enables the user to visualize the kinematic performance of a machine much more easily [Yong 1990]. Simulation systems, which incorporate animation capabilities, have been used in the design of various robotic applications and have proved very useful tools for robot work cell design and evaluation [Miller,1985, Yong and Bennaton 1988].

The realization and control of animated motion requires a specification of the machine's kinematics. The kinematics of a modular machine system can be characterised by its position versus time information, relating all moving elements in the system, together with information concerning the velocity and acceleration of each motion. In manufacturing engineering applications, it is quite common to require a machine tool point to traverse a specified three dimensional continuous curve, and thus it is necessary not only to study discrete position versus time relationships but also to include a description of the continuous tool point path in the kinematic study. In this chapter, a study of the kinematic properties of modular machine systems is carried out.

6.2 Kinematic representations of machine primitives

6.2.1 Single degree of freedom motion primitives

In this study two types of single degree of freedom motion primitive are included in the machine primitive library, viz: prismatic and revolute axis or single degree of freedom machine building elements (see Figure 6.1). They represent the simplest cases of kinematic (or motion) primitives of machines. In the kinematic representation of a mechanism, two methods are usually used to describe its motion, involving respectively the use of coordinate equations (describing typically the position of the end-point) or the use of homogeneous transformations. For example, the relationship between the end-point of a revolute axis in its original frame{1} to that in its rotated frame{2} can be expressed as follows: where rotation is through θ around the Z-axis,

$$X_1 = X_2 \cos(\theta) - Y_2 \sin(\theta)$$

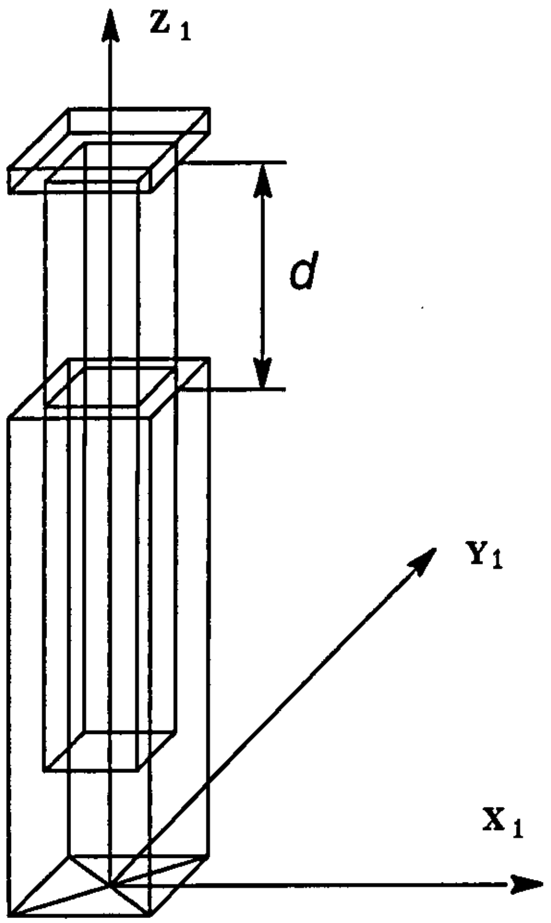
$$Y_1 = X_2 \sin(\theta) + Y_2 \cos(\theta)$$

$$Z_1 = Z_2 \quad .$$

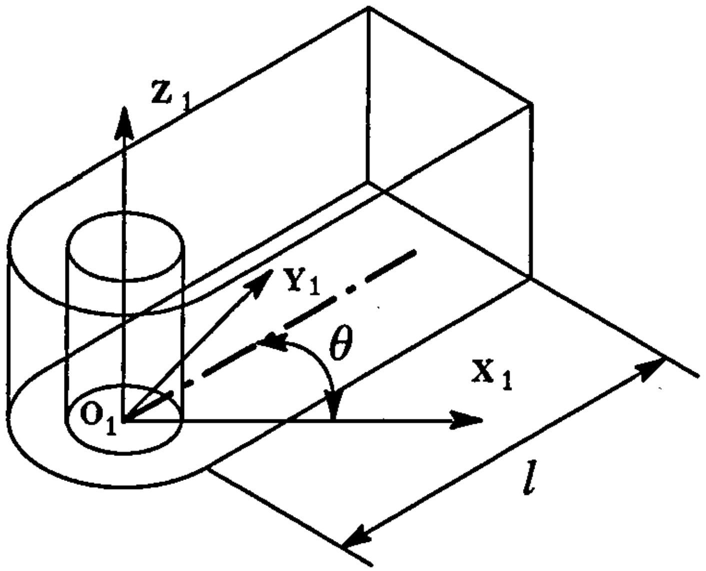
This same relationship can also be described by using a homogeneous transformation as follows:

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} \quad .$$

Usually the homogeneous transformation matrix for rotation about the Z axis through θ is



Axis coordinate frame $O_1\ X_1\ Y_1\ Z_1$
and the axis displacement d .



Axis coordinate frame $O_1\ X_1\ Y_1\ Z_1$
and the axis rotating angle θ .

Figure 6.1 Two basic types of motion primitives and their kinematic parameter representation

denoted by Rot(Z, θ), i.e.

$$Rot(Z, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Since the homogeneous transformation matrix is a much more convenient form than explicit equations, especially when mechanisms become complex, this method has been commonly used in the kinematic representation of machine primitives [Paul 1981, Fu et al. 1987].

Similarly, the homogeneous transformation for positive rotation around X is

$$Rot(X, \alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and for a positive rotation about the Y axis

$$Rot(Y, \phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

If the axis frame origin is translated through distance *a* in the positive X-direction of coordinate frame, *b* in a positive Y-direction and *c* in a positive Z-direction, the homogeneous translation transformation matrix becoming

$$Trans(X, Y, Z) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Since the homogeneous transformation matrices are used to describe the transformation (rotation and translation respectively) between the original frame and the new coordinate frame of motion axis, they can be further generalised as

$$Transformation\,(X,Y,Z) = \begin{bmatrix} t_{11} & t_{12} & t_{13} & d_x \\ t_{21} & t_{22} & t_{23} & d_y \\ t_{31} & t_{32} & t_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Rot & Trans \\ 0\,0\,0 & 1 \end{bmatrix} ,$$

where, *Rot* denotes the 3×3 rotational component of new frame$\{2\}$ with respect to the original frame$\{1\}$ and *Trans* is a 3-element column vector pointing from the origin of frame$\{1\}$ to that of frame$\{2\}$. The simplified homogeneous transformation matrix can be stored in a transformation data block - entity block in the form of Rot(3×3) and Trans(3). Four data words are saved due to this generalisation, this being a useful simplification which has been utilised in this study.

Since single degree of freedom motion primitives can only move along, or rotate around, one direction of the local coordinate frame, there is only one variable in the single motion primitive transformation matrix. For example, the translational motion along the Z-direction of the local frame can be expressed as

$$Trans\,(X,Y,Z) = Trans\,(0,0,z) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} ,$$

where a and b are the fixed displacement of the motion axis along X and Y directions and z is a variable within its working constraint. In order to simplify the creation of an axis primitive, the Z direction of a local coordinate frame is always chosen as the initial direction of rotation or translation. For the rotation, the homogeneous transformation matrix is then

$$Rot(Z, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & a \\ \sin(\theta) & \cos(\theta) & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} ,$$

where a, b and c are the fixed initial displacements along X, Y and Z directions of the local frame, and θ is variable of the rotation.

6.2.2 General coordinate frame establishment

The Denavit-Hartenberg [1955] representation is employed in this study to represent the articulated axis groups (high order primitives) because it has been proven to be a very effective method of modelling articulated robots. A brief description of this approach is given below as it forms a mathematical base of this modelling study. A coordinate system attached to each link of the primitive must be established in order to produce a 4×4 homogeneous transformation matrix; this represents the coordinate system of each link at the connecting joint with respect to the previous link's coordinate system. Through a sequential concatenation of transformations from the first to the last axis, the primitive end-point can be transformed and its position obtained in terms of the local coordinate frame of a high order primitive. For each primitive constituent (e.g. axis of motion) an orthogonal Cartesian coordinate system (O_i, X_i, Y_i, Z_i) is established and attached to each axis of motion at its connecting joint, where i = 1, 2, 3 (frame{0} is for the local coordinate frame).

The convention for establishing these orthogonal coordinate frames is based on the following rules:

(1) The Z_i direction of a local frame of an axis primitive is always chosen to be in the

direction of translation if the axis is a prismatic one or a rotation in the revolute case;

(2) The X_i coordinate direction will be normal to the Z_i and Z_{i-1} direction, pointing away from the Z_{i-1} axis if Z_i and Z_{i-1} are not in a common plane;

(3) The Y_i coordinate direction complies with the right-hand coordinate system convention;

(4) The coordinate system establishment starts from the base part of axis1 and finishes at the end-point of the moving part on axis2 in this case (obviously this would be axis3 in the case of three serially chained single degree of freedom primitives).

Based on this convention, a rigid link geometry can be expressed by four geometric parameters (see Figure 6.2). The definitions of each of these four parameters are given below:

d_i: the distance from the origin of the (i-1)th coordinate frame to the intersection of the Z_{i-1} direction with the X_i along the Z_{i-1} direction;

θ_i: the joint angle from the X_{i-1} coordinate axis to the X_i coordinate axis about the Z_{i-1};

a_i: the normal distance from the intersection of the Z_{i-1} coordinate axis with the X_i coordinate axis to the origin of the *i*th frame along the X_i direction;

α_i: the offset angle from the Z_{i-1} direction to Z_i direction about the X_i.

With these four parameters, the characteristics of a joint in an articulated motion primitive can be defined in terms of its position and orientation relative to its previously connected axis. Since each low level motion primitive contained within the modular machine library has only one degree of freedom, there will be one variable amongst the four parameters. For a revolute axis, d_i, a_i, and α_i remain constant for the joint, whereas θ_i is a variable which will change when the motion axis rotates with respect to its preceding connection. For a prismatic axis of motion, θ_i, a_i and α_i are constant, where d_i is the axis variable.

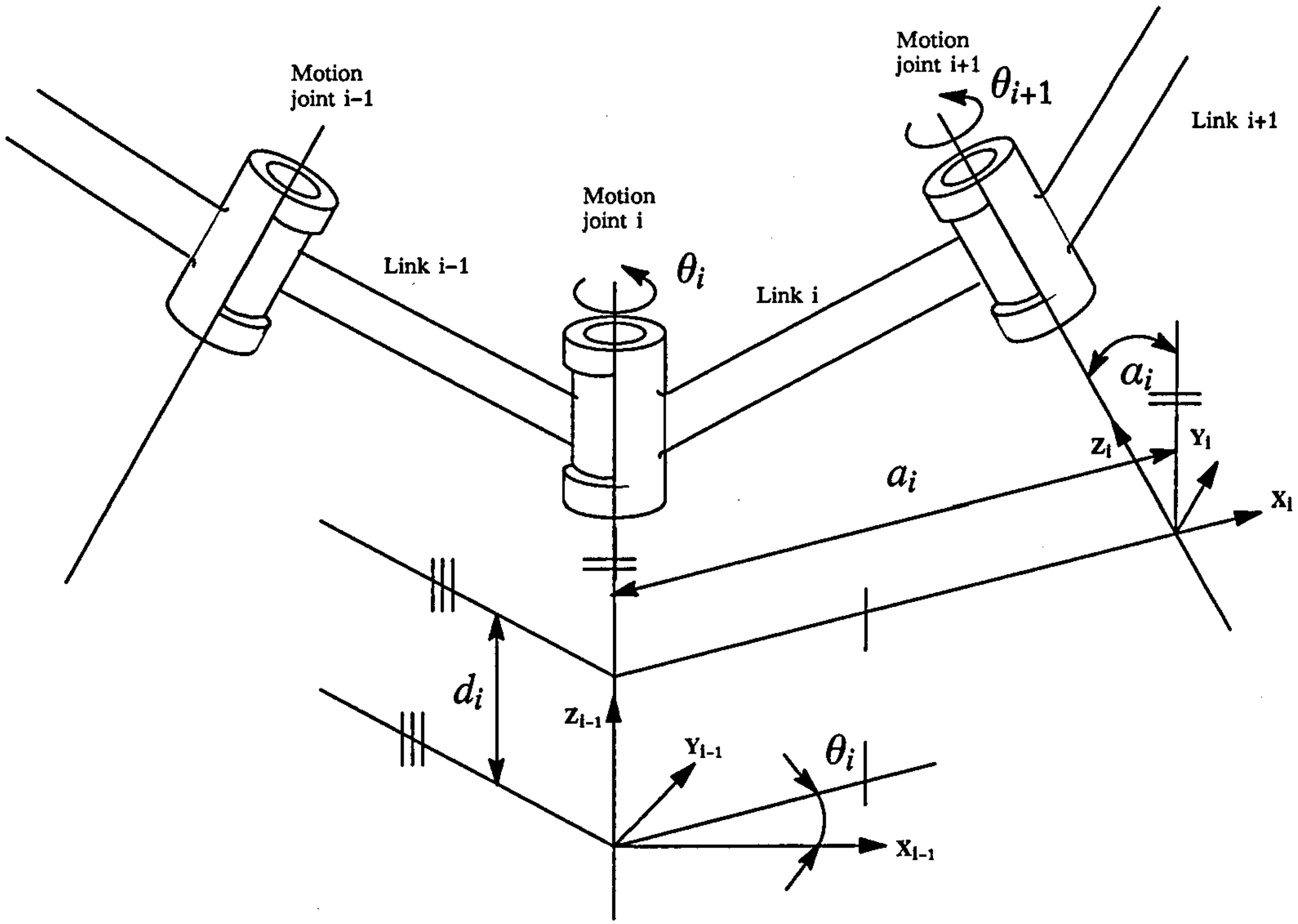


Figure 6.2 Motion axis coordinate system and its parameters

The homogeneous transformation matrix can then be developed to describe the relationship between the joint coordinate frame{i-1} and {i}, this being known as the D-H transformation matrix for adjacent coordinate frames [Fu et al. 1987, Craig 1989 and Paul 1981]. It can be obtained by the following transformations:

(1) rotating frame{i-1} through the angle of θ_i around the Z_{i-1} coordinate axis in order to align the X_{i-1} coordinate axis with X_i;

(2) translating along the Z_{i-1} direction a distance of d_i to bring X_{i-1} and X_i into coincidence;

(3) translating along the X_i coordinate axis by a_i to coincide the two origins X_i and X_{i-1};

(4) rotating through an angle of α_i around the X_i coordinate axis in order to bring the two coordinate systems into coincidence.

Therefore

$$T(i-1,i) = Rot(Z_{i-1},\theta_i)Trans(Z_{i-1},d_i)Trans(X_i,a_i)Rot(X_i,\alpha_i)$$

$$= \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Based on this equation, the forward kinematic transformations of two or three serially modular machine primitives can be easily solved.

6.2.3 Two degree of freedom articulated axis groups - higher order motion primitive

6.2.3.1 Forward kinematics of two degree of freedom (DOF) articulated axis group

For higher order articulated machine primitives constructed from chaining together two single DOF motion primitives, a further degree of freedom is introduced requiring a second variable to describe the new degree of freedom. The forward kinematic problem is thus to obtain the end-point (or Working Centre Point) position in the global Cartesian coordinate frame, assuming the displacement or rotation of each axis primitive is known. To solve this problem transformation through the local coordinate frames of each constituent single DOF motion can be used and compounded. Thus the end-point position of a higher order two-axis primitive can be obtained as the result of coordinate frame transformation from the second axis frame to the end-point after consideration of the transformation from the first axis frame to second axis frame. All possible seven two axis higher order primitives are listed in Figure 4.5, and the forward kinematic transformation of the first of these primitives is described below. The primitive is composed of two prismatic motions, therefore the end-point (WCP) position is given as follows: this being based on the Denavit and Hartenburg [1955] convention

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = Trans(Z, d_1) \times Rot(Y, 90) \times Trans(Z, d_2)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & d_2 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

or
$$X_{wcp} = d_2$$
$$Y_{wcp} = 0$$
$$Z_{wcp} = d_1 \quad ,$$

where (X_{wcp}, Y_{wcp}, Z_{wcp}) is the end-point position relative to the local frame of the higher order primitive. A coordinate frame rotation is included since the Z-direction of the local frame of an axis primitive is always chosen to be in the direction of the translation or rotation for the convenience of machine primitive solid modelling [Paul 1981]. The same convention is used throughout for kinematic representations. By using the D-H transformation matrix, the same equations mentioned above can be obtained and the D-H method is illustrated in the next higher order primitive.

For the primitive (2) in Figure 4.5, the link parameter table can be summarised as follows:

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	d_1	0	0	0	d_1	1	0
2	θ_2	θ_2	-90	a_2	0	0	-1

Then the corresponding $T_2(i-1,i)$ matrices for the prismatic and revolute single motion primitives are:

$$T_2(0,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2(1,2) = \begin{bmatrix} \cos(\theta_2) & 0 & -\sin(\theta_2) & a_2\cos(\theta_2) \\ \sin(\theta_2) & 0 & \cos(\theta_2) & a_2\sin(\theta_2) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus

$$T_2(0,2) = T_2(0,1) \times T_2(1,2) = \begin{bmatrix} \cos(\theta_2) & 0 & -\sin(\theta_2) & a_2\cos(\theta_2) \\ \sin(\theta_2) & 0 & \cos(\theta_2) & a_2\sin(\theta_2) \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The forward homogeneous transformation matrices for the rest of the two DOF machine primitives illustrated in Figure 4.5 can be found in Appendix B.

6.2.3.2 The inverse kinematics of two DOF articulated axis groups

The inverse kinematics of a two DOF motion primitive deals with the issue of how to obtain the servo-input values of the two individual motion primitives, thereby establishing a desired motion of the end-point of a higher order primitive. Since in this case there are only two motion axes, the inverse kinematic problem can be solved by using the method of comparing the corresponding elements of the matrices on each side of following equation,

$$\begin{bmatrix} a_{i1} & b_{i1} & c_{i1} & p_{i1} \\ a_{i2} & b_{i2} & c_{i2} & p_{i2} \\ a_{i3} & b_{i3} & c_{i3} & p_{i3} \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_i(0,2) = T_i(0,1) \times T_i(1,2)$$

where $T_i(0,2)$ is the combined transformation from the local frame of the higher order primitive to the last single DOF motion primitive in the group, i=1, 2, 3, 4, 5, 6, 7 for the combinations in Figure 4.5. The left side of the equation is a generalised vector p and a 3*3 matrix. The vector p represents the position of the higher order primitive end-point with respect to the higher order primitive local coordinate frame; whereas the matrix specifies the orientation of the end-point. For the situation (1) in Figure 4.5, the following equation can be established:

$$\begin{bmatrix} a_{11} & b_{11} & c_{11} & X_{11} \\ a_{12} & b_{12} & c_{12} & Y_{12} \\ a_{13} & b_{13} & c_{13} & Z_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_1(0,2) = \begin{bmatrix} 0 & 0 & 1 & d_2 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Therefore, the solution is $d_2 = X_{11}$ and $d_1 = Z_{11}$. For the configuration of (2) in Figure 4.5, the equation is

$$\begin{bmatrix} a_{21} & b_{21} & c_{21} & X_{21} \\ a_{22} & b_{22} & c_{22} & Y_{22} \\ a_{23} & b_{23} & c_{23} & Z_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_2(0,2) = \begin{bmatrix} \cos(\theta_2) & 0 & -\sin(\theta_2) & a_2\cos(\theta_2) \\ \sin(\theta_2) & 0 & \cos(\theta_2) & a_2\sin(\theta_2) \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

A solution to enable control of individual single-DOF primitive axes can be expressed in terms of the given position (only a two dimensional position value set of x and y being required here):

from

$$\begin{aligned} Z_{23} &= d_1 \\ X_{21} &= a_2 \times \cos(\theta_2) \\ Y_{22} &= a_2 \times \sin(\theta_2) \end{aligned} \qquad \text{hence} \qquad \begin{aligned} d_1 &= Z_{23} \\ \theta_2 &= \mathrm{atan2}\,(X_{21}/a_2, Y_{22}/a_2) \end{aligned}$$

where atan2(y, x) returns = tan-1(y/x) adjusted to the proper quadrant. The inverse kinematic solutions for other situations in Figure 4.5 are described in Appendix B.

6.2.4 Higher order primitives formed from three DOF articulated axis groups

Three degree of freedom articulated axis groups are important higher order modular machine primitives, providing the necessary articulation to reach a three dimensional position. Such a requirement has been widely reported and adopted in industrial robotics [Craig 1989, Tourassis and Ang, Jr. 1989, Gupa 1986, Fu et al. 1987, Takano 1985]. Since a further degree of freedom is introduced by chaining a further axis of motion, the complexity is increased requiring an additional homogeneous transformation; i.e. matrix multiplication to obtain the forward transformation. Figure 4.6 shows all possible twelve combinations of three single degree of freedom articulated machine groups which can be created with perpendicular or parallel joint motion directions. Typical combinations are discussed as follows.

6.2.4.1 Three prismatic axes articulated in the Z, X and Y directions

For the case of (1) in Figure 4.6, based on the conventions of coordinate frame establishment, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure 6.3. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	d_1	0	–90	0	d_1	0	–1
2	d_2	90	–90	0	d_2	0	–1
3	d_3	–90	0	0	d_3	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2} and frame{2} to frame{3} are respectively

$$T_{11}(0,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{12}(1,2) = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{13}(2,3) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1(0,3) = T_{11}(0,1) \times T_{12}(1,2) \times T_{13}(2,3) = \begin{bmatrix} 0 & 0 & -1 & -d_3 \\ 1 & 0 & 0 & d_2 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

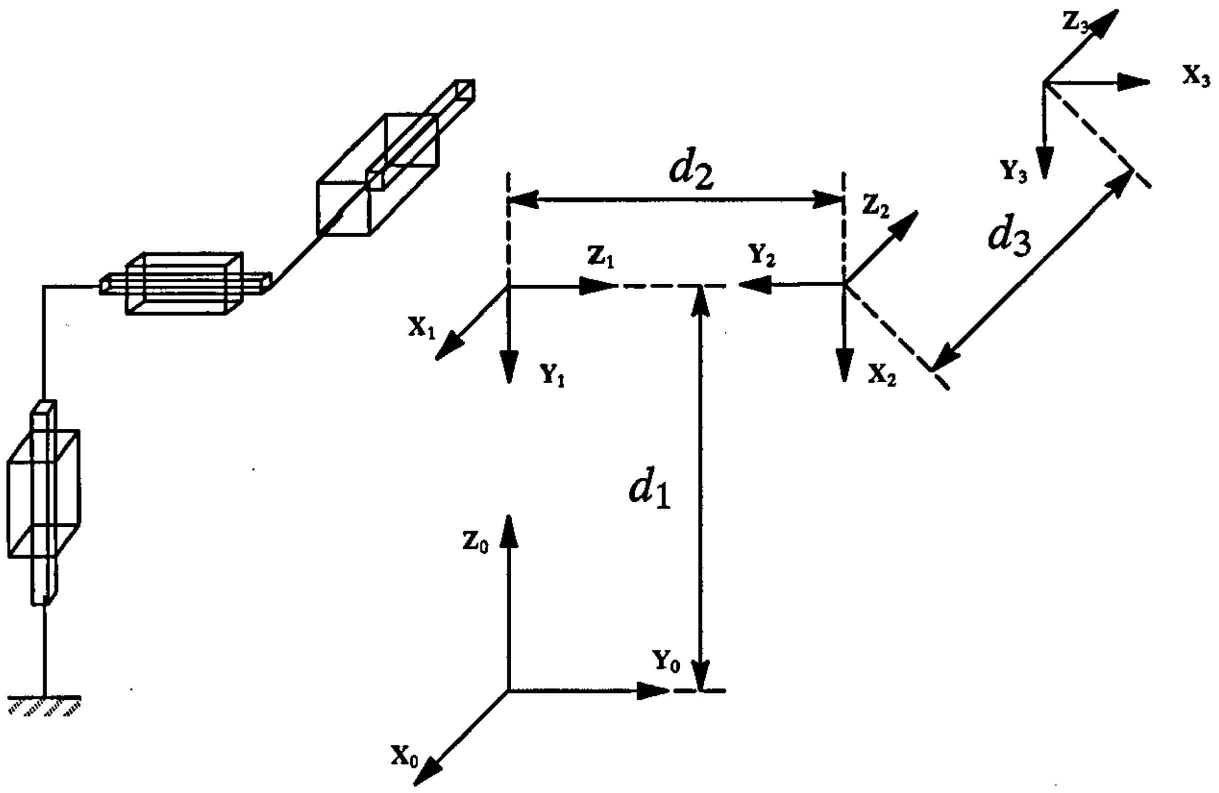


Figure 6.3 A three articulated prismatic axis group and their coordinate frame assignments

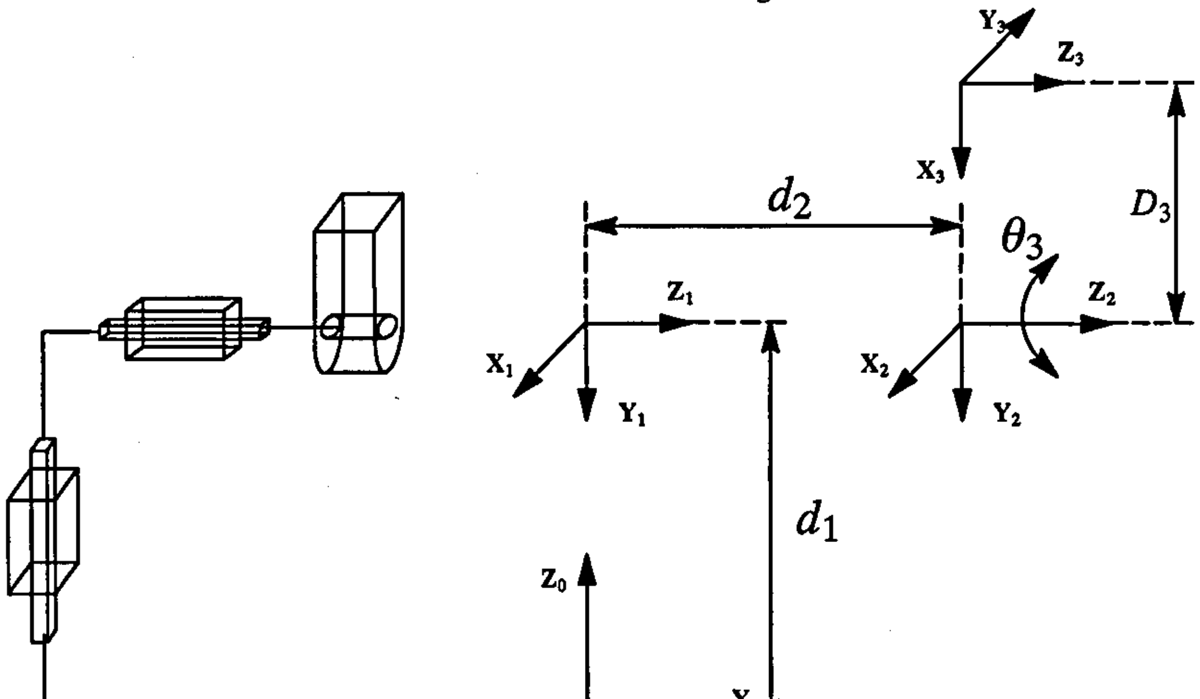


Figure 6.4 Another three articulated axis group and the axes coordinate frame assignments

The orientation matrix and position vector of the primitive is given by

$$T_i = \begin{bmatrix} a_{i1} & b_{i1} & c_{i1} & p_{i1} \\ a_{i2} & b_{i2} & c_{i2} & p_{i2} \\ a_{i3} & b_{i3} & c_{i3} & p_{i3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad .$$

After the comparison of T_i and $T_1(0,3)$, the position of the end-point of the last single motion primitive can be expressed by

$$P = \begin{bmatrix} p_{1x} \\ p_{1y} \\ p_{1z} \end{bmatrix} = \begin{bmatrix} d_3 \\ d_2 \\ d_1 \end{bmatrix} \qquad \text{or} \qquad \begin{aligned} P_{1x} &= -d_3 \\ P_{1y} &= d_2 \\ P_{1z} &= d_1 \end{aligned}$$

From the above equations it is very easy to obtain the inverse kinematic solutions to control the separate primitive axes when a desired position is given:

$$\begin{aligned} d_1 &= p_{1z} \\ d_2 &= p_{1y} \\ d_3 &= -p_{1x} \quad . \end{aligned}$$

6.2.4.2 Two prismatic axes articulated (or chained) in the Z and X direction with a third revolute axis connected to the second prismatic unit

In this case a revolute axis is introduced and a third control variable θ_3 should be considered. A coordinate frame assignment for the three articulated axes must be made and one attempt is depicted as in Figure 6.4, according to the conventions of coordinate establishment. The associated link parameter table is listed in the following table.

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	d_1	–90	0	0	d_1	0	–1
2	d_2	0	0	0	d_2	1	0
3	θ_3	90	θ_3	D_3	0	0	1

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2} and frame{2} to frame{3} are respectively

$$T_{21}(0,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{22}(1,2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{23}(2,3) = \begin{bmatrix} \cos(\theta_3) & 0 & \sin(\theta_3) & D_3 \times \cos(\theta_3) \\ \sin(\theta_3) & 0 & -\cos(\theta_3) & D_3 \times \sin(\theta_3) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where D_3 is the link length of the revolute axis. The compounded transformation is

$$T_2(0,3) = \begin{bmatrix} \cos(\theta_3) & 0 & \sin(\theta_3) & D_3 \times \cos(\theta_3) \\ 0 & 1 & 0 & d_2 \\ -\sin(\theta_3) & 0 & \cos(\theta_3) & -D_3 \times \sin(\theta_3) + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

If comparison is made between T_i (i=1,2...12 for the twelve combinations) and $T_2(0,3)$, the position of the end-point of the high order primitive can be given by

$$\begin{bmatrix} p_{2x} \\ p_{2y} \\ p_{2z} \end{bmatrix} = \begin{bmatrix} D_3 \times \cos(\theta_3) \\ d_2 \\ -D_3 \times \sin(\theta_3) + d_1 \end{bmatrix} \quad \text{or} \quad \begin{aligned} P_{2x} &= D_3 \cos(\theta_3) \\ P_{2y} &= d_2 \\ P_{2z} &= -D_3 \sin(\theta_3) + d_1 \end{aligned} .$$

From these equations, the inverse kinematic solutions can also be derived as follows.

From $p_{2x} = D_3\cos(\theta_3)$, then $\theta_3 = \pm \arccos(p_{2x}/D_3)$

If $\theta_3 = \arccos(p_{2x}/D_3)$, substituting θ_3 in $p_{2z} = -D_3\sin\theta_3 + d_1$,

then $d_1 = p_{2z} + D_3\sin(\arccos(p_{2x}/D_3))$

If $\theta_3 = -\arccos(p_{2x}/D_3)$, repeat same substitution

then $d_1 = p_{2z} - D_3\sin(\arccos(p_{2x}/D_3))$.

Obviously, two groups of solutions exist for this type of three degree of freedom serially chained manipulator: the distinction being with respect to the given position as follows.

$$\begin{aligned} d_1 &= p_{2z} + D_3 \sin\left(\arccos\left(\frac{p_{2x}}{D_3}\right)\right) \\ d_2 &= p_{2y} \\ \theta_3 &= \arccos\left(\frac{p_{2x}}{D_3}\right) \end{aligned} \quad \text{and} \quad \begin{aligned} d_1 &= p_{2z} - D_3 \sin\left(\arccos\left(\frac{p_{2x}}{D_3}\right)\right) \\ d_2 &= p_{2y} \\ \theta_3 &= -\arccos\left(\frac{p_{2x}}{D_3}\right) \end{aligned} .$$

This means that given one set of local Cartesian coordinate values, there are two possible configurations which will satisfy the required position (assuming that the axis rotates around Z_2 and starts from X_2 as its zero angle).

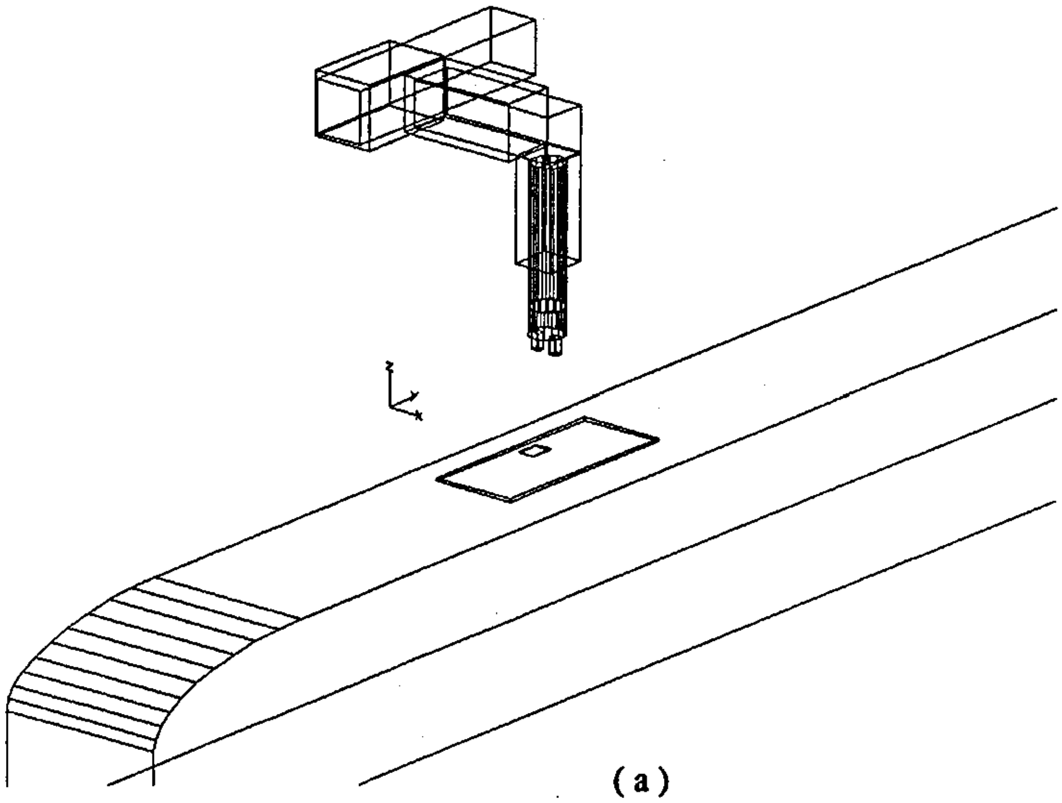
The same coordinate frame assignment rules can be applied to the rest of the twelve three axis combinations, and by comparing the T_i (i=1, 2, 3...12 for the twelve configurations) and $T_i(0,3)$, the forward kinematics solutions can be derived in terms of the given single primitive axis rotation or translation. The same equations can be used to derive the inverse kinematic solutions as demonstrated above. See Appendix 6 for solutions derived by the author to the rest of the twelve configurations.

6.2.5 Motion of distributed manipulators

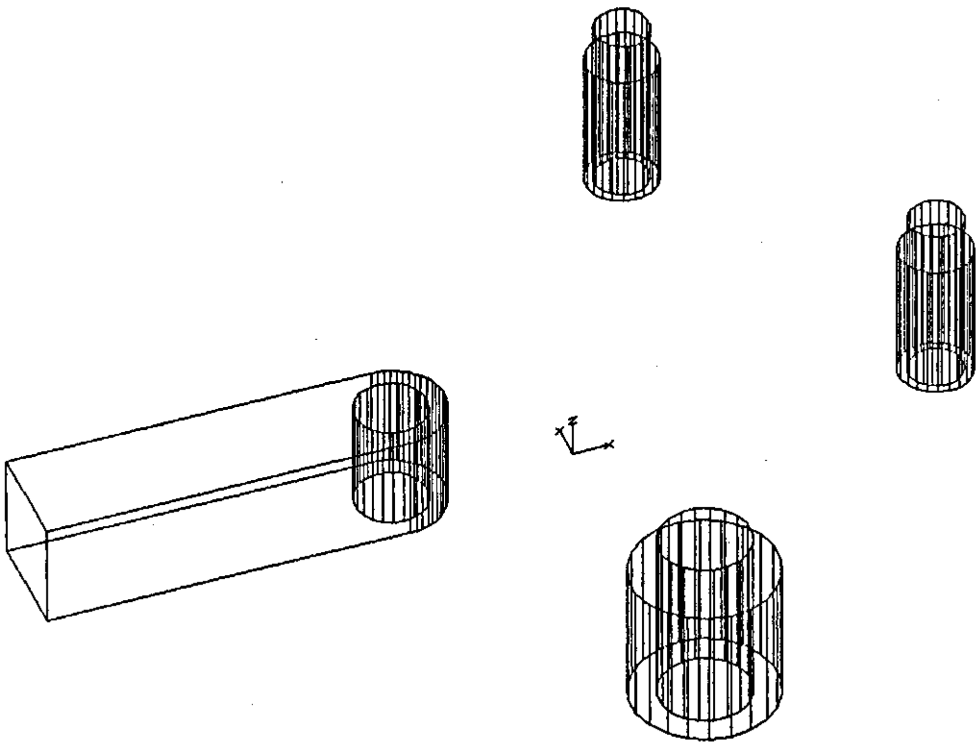
As earlier discussed, many types of manufacturing machine require that the relative motion of more than one axis group be controlled concurrently. In such situations the individual groups may be formed from mechanically decoupled modules to provide one, two or three degrees of motion as required; e.g. this may or may not involving the serial chaining of axes to achieve articulation in the form previously described and analysed in this chapter. Thus there is a need to simulate the relative motion of two or more groups of axes, where this relative motion will be referred to as distributed motion to emphasise that the different groups may be only logically related rather than physically linked together and that the groups can be located in different co-ordinate frames as required. Describing the kinematics of distributed motion (as defined above) involves a relatively simple extension of the kinematic solutions presented in the previous sections of this chapter. This essentially being because the kinematics of individual axis groups will still be valid with respect to their primitive local coordinate frames. However, the main interest in simulating distributed motion lies in that the relative positions of each end-point of two or more motion primitives have to be aligned or related accurately in terms of the local coordinate frame of the distributed axis primitive.

To illustrate this requirement further, consider typical assembly operations where often there is a need for accurate alignment of two or more physically decoupled motion primitives to achieve parts presentation and fixing (e.g. insert a component into its mating part). One typical example is that after the arrival of a printed circuit board (PCB) on a transporting motion primitive, a second axis group provides articulated motion to align electronic components perpendicular to the insertion position on the PCB board before inserting the component. (see Figure 6.5 a). This type of multi-device operation is characterised by mechanical concurrency and the need for device coordination.

A different but common requirement for distributed motion is found when two or more motion primitives have to move or rotate while maintaining some pre-defined relationship. A typical example of this type of distributed motion is found in software gear box systems where several physically decoupled revolute motion primitives have to rotate while maintaining a pre-specified gear-ratio and rotation-ratio between the driven gears and the driving gear (see Figure 6.5.b). In this case the distributed motion is characterised by the



(a)



(b)

Figure 6.5 (a) A PCB assembly mechanism and (b) a group of physically decoupled gear-driven axes in a distributed motion situation need for concurrency but also a requirement for motion synchronisation.

In terms of data representation for each of these two types of coordinate frame system, a special data item is reserved to describe the frame type in the general data block (which in respect of this study was also the head block of the distributed motion primitive ring). Here the character "A" in word5 of the head block of the ring was used to denote that this axis group is an articulated group, where the alternative use of "D" denotes a distributed axis group in its configuration. For a complete description of data contents of the two types of axis group, see tables 6.1 and 6.2. Data control and manipulation will be discussed in the next chapter.

6.3 Position definition

Having established a geometric modelling facility for both articulated and distributed higher order motion primitives, together with aggregation techniques, it is necessary to facilitate means of controlling their motion. Two methods of controlling motion were considered to be important in this study. The first and most simple concerns movement of one axis at a time. The second involves co-ordinated motion of a complete axis group (higher order primitive) to enable a desired position to be reached and involves the computation of motion control values through solution of the inverse kinematics of the corresponding higher order motion primitive. It is a fairly straightforward matter to provide motion control of the first class of manipulation, since all motions are manipulated at the single axis level (i.e. each axis moving one at a time with reference to kinematic solutions of each axis). However, as for robots it is very important that modular machines can be manipulated in such a manner that they can be positioned along some path to accomplish a

Word No	Data Contents	Comments
Word 0	Type & length	Data type and length
Word 1	P R P	Principal Ring Pointer
Word 2	S R P	Secondary Ring Pointer
Word 3	N B P	Name Block Pointer
Word 4	D S F	Display Status Flag
Word 5	'D' or 'A'	Denote for distributed or articulated
Word 6	'R' or 'A'	For relative or absolute coordinate
Word 7	Gripper Pointer	Gripper head data block pointer
Word 8	Axis1 mptptr	The moving part pointer of first axis
Word 9	Axis2 mptptr	The moving part ptr of second axis
Word 10	Axis3 mptptr	The moving part ptr of third axis
Word 11	Axis1 basptr	Axis1 base part data block pointer
Word 12	Axis2 basptr	Axis2 base part data block pointer
Word 13	Axis3 basptr	Axis3 base part data block pointer
Word 14	Axis1 pointet	Axis1 head data block pointer
Word 15	Axis2 pointer	Axis2 head data block pointer
Word 16	Axis3 pointer	Axis3 head data block pointer
Word 17	WCP pointer	End-point of axis group pointer
Word 18	Empty	word18 and word 19
Word 19		Empty for other data manipulation
Word 20		purpose
Word 21	'P' or 'R'	Axis1 type (Prismatic or Revolute)
Word 22	'P' or 'R'	Axis type of second axis
Word 23	'P' or 'R'	Axis type of third axis
Word 24	Empty	Empty for other use
⋮	⋮	⋮
Word 31	Empty	Empty for other use

Table 6.1 The head data block contents of articulated axis group

Word No	Data Contents	Comments
Word 0	Type & length	Data type and length
Word 1	P R P	Principal Ring Pointer
Word 2	S R P	Secondary Ring Pointer
Word 3	N B P	Name Block Pointer
Word 4	D S F	Display Status Flag
Word 5	'D' or 'A'	Denote for distributed or articulated
Word 6	'R' or 'A'	For relative or absolute coordinate
Word 7	Primitive1 ptr	Primitive1 head data block pointer
Word 8	Primitive2 ptr	Primitive2 head data block pointer
Word 9	Primitive3 ptr	Primitive3 head data block pointer
Word 10	Primitive4 ptr	Primitive4 head data block pointer
⋮	⋮	
Word 16	Primitive9 ptr	Primitive9 head data block pointer
Word 17	Primitive10 ptr	Primitive10 head data block pointer
Word 18	End primitives	'E' for the end of primitives group
Word 19	'P' or 'R'	Axis1 type (Prismatic or Revolute)
Word 20	'P' or 'R'	Axis type of second axis
Word 21	'P' or 'R'	Axis type of third axis
⋮	⋮	
Word 24	'P' or 'R'	Axis type of sixth axis
Word 25	'P' or 'R'	Primitive type of seventh axis
Word 26	'P' or 'R'	Primitive type of eighth axis
Word 27	'P' or 'R'	Primitive type of ninth axis
Word 28	'P' or 'R'	Primitive type of tenth axis
Word 29	Empty	Empty for other use
⋮	⋮	⋮
Word 31	Empty	Empty for other use

Table 6.2 The head data block contents of distributed axis group

complex task without the need for excessive programming time. Thus a means of achieving positioning along such a path was derived and implemented. This included three ways of defining positions described in the following sub-sections.

6.3.1 A position relative to a local frame of a motion primitive

A point to which the motion primitive will refer to or pass through can be defined with respect to the local coordinate frame of a higher order motion primitive. Let point $P_{ij}(X,Y,Z)$ be a position in the coordinate frame of a motion primitive. P_{ij} can be specified relative to this local frame $O_iX_iY_iZ_i$(where i=1, 2, 3,...n and n is the total number of local frames, or motion groups in the whole machine model) by local coordinates (X_j, Y_j, Z_j), where j= 1, 2, 3,...m (m being the number of points in frame{i}) (see Figure 6.6). The convenience of this type position definition lies in the directness and simplicity when obtaining kinematic solutions. The inverse kinematic solutions for an axis group to reach a point P_{ij} defined locally can be obtained by calculating the moving increment values of each axis in the local frame of the axis group in respect to the point P_{ij}. Therefore there is no need to transform the motion target point P_{ij} from any other coordinate frame (e.g. from the global or an object frame to the local one). This is of great importance in accurately representing a target position of a manipulator since a relative accuracy between the point P_{ij} and the motion primitive local origin O_i is achieved through eliminating the number of transformations between coordinate frames (i.e. reducing the approximation errors in the computation of a coordinate frame transformation). However a potential problem with this type of position definition is that sometimes it is difficult to obtain the coordinate values of P_{ij} relative to $O_iX_iY_iZ_i$.

Once a point is defined, all position and reference frame information can be stored in one

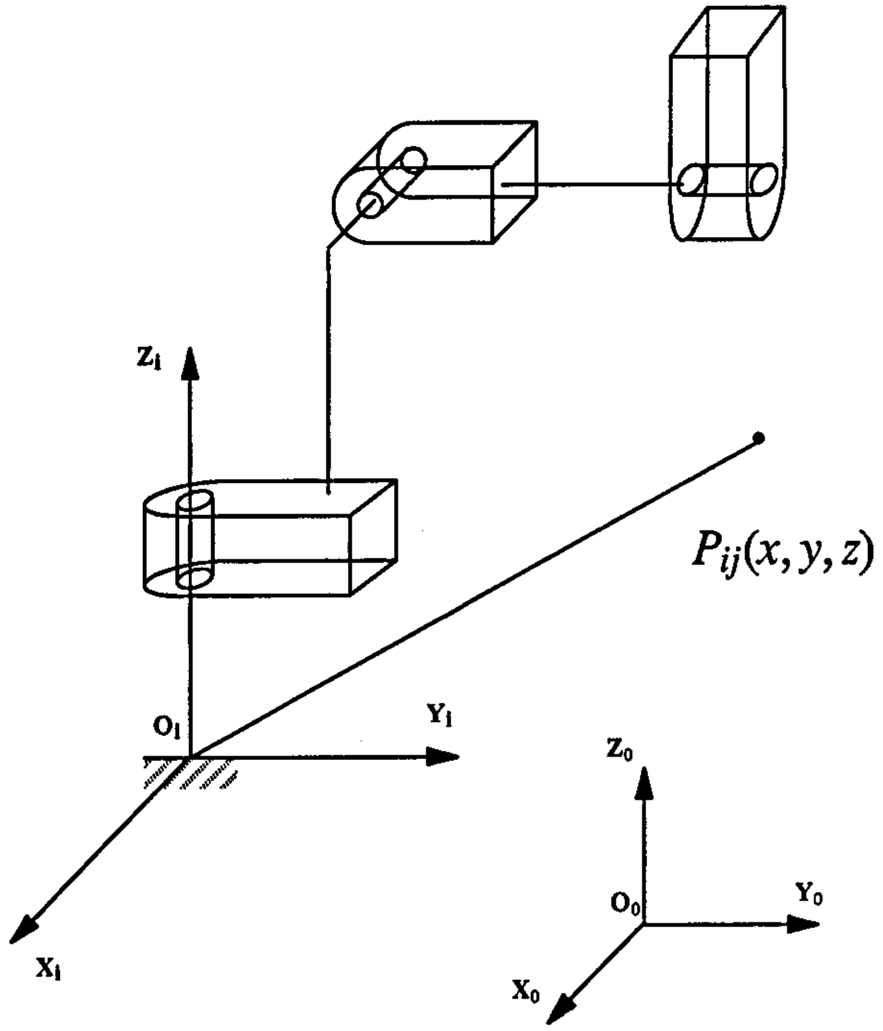


Figure 6.6 The position definition relative to local motion primitive frame

Word No	Data Contents	Comments
Word 0	Type & length	Data type & length
Word 1	Prin. Ring Ptr.	Principal Ring Pointer
Word 2	Secnd. Ring Ptr.	Secondary Ring Pointer
Word 3	Name Block Ptr.	Name Block Pointer
Word 4	Disp. Status Flag	Display Status Flag
Word 5	Ref. origin ptr.	Reference origin pointer
Word 6	'P' or 'O' or 'G'	'P' for primitive frame defi-
Word 7	Empty	nition, 'O' for non–motion
Word 8	Empty	object definition, and 'G' for
Word 9	Empty	global definition
Word 10	X Value	X Value of the position
Word 11	Y Value	Y Value of the position
Word 12	Z Value	Z Value of the position
Word 13	Empty	Empty
Word 14	Empty	Empty
Word 15	Empty	Empty

Table 6.3 The data contents of a position data block

general data block of the position data ring, this being a branch of the kinematic ring. The data contents chosen in this study are shown in table 6.3, where the reference coordinate origin pointer is stored in word 5 and the reference coordinate type is stored in word 6. A "P" in word 6 denotes that the reference coordinate frame is the local coordinate frame of a motion primitive.

6.3.2 A position relative to a non-powered object frame

In a situation where high relative accuracy is required (e.g. when establishing the relative position of an object and a point P_{ij}) position definition may best be done with respect to a local object. Since the position of P_{ij} is defined relative to the object origin O_i, improved relative accuracy can be achieved through direct transformation from the coordinate frame $O_iX_iY_iZ_i$ to point P_{ij}. In this case, each point P_{ij} is transformed into a point in the local coordinate frame of a motion primitive through the object frame $O_iX_iY_iZ_i$ to obtain the kinematic solutions for a target point specified in an object frame, i.e. the relative position between P_{ij} and O_i is guaranteed to be exact and explicit to a user although the kinematic solution is finally in the form of P_{ij} relative to the local coordinate frame of the motion primitive (axis group). The end-point movement of a motion primitive from P_{ij} to another point in the frame $O_iX_iY_iZ_i$ can be determined by applying kinematic algorithms describing the higher order primitive, considering the coordinate frame transformation from the object to a motion primitive frame.

Another advantage of this type of position definition is that it maintains a relative spatial relationship within the local frame $O_iX_iY_iZ_i$. When executing a control program for the primitive, should the motion primitive be reconfigured or modified, the position data referenced need not be changed. This property improves the system's modularity and improves its "friendliness" at the user-interface. In this study the character "O", in word 6 of the position data block, is used to denote positioning relative to a non-powered object frame. Here the reference origin pointer contains the address of the object coordinate frame, while X_j, Y_j and Z_j are the coordinate values relative to the object origin (see Table 6.3).

6.3.3 A position relative to the global frame

A further useful facility can be offered by allowing positions in the modular machine simulation model to be defined by referring them to a global coordinate frame. This type of position definition is useful when certain positions are required to enable several motion primitives to refer to them equally in the terms of importance in accuracy. In this situation, a position relative to the global frame can be referred to by several motion primitives for co-ordinating a position. When sharing a common position (in the sense of equal reference importance among more than two motion primitives) the position definition should be neutral with respect to the primitives concerned (i.e. the position should be defined without direct reference to any coordinate frames of their associated motion primitives. This type of neutral position definition can associate motions and lead to multi-primitive coordination, as discussed in the next chapter.

Another use of this type of position definition occurs when the complete model needs global reference points. A coordinate frame can be added at such a global point to visualize the reference frame position.

6.3.4 Methods of assigning position values

Position definition values for each of the three types can be assigned in one of the two following ways:

(i) Inputting position information interactively with respect to an appropriate coordinate frame; or

(ii) Moving motion primitives until they reach a required position, then recording the position with reference to an appropriate coordinate frame.

It is important to identify position uniquely and in this study unique names are assigned within the model. Thus position names are assigned whenever a position is defined along with the chosen type of reference coordinate frame. Having defined a number of positions using the above methods they are arranged to form a positional data ring. Implementation of the head data block of this kinematic modelling data ring was achieved within word 5, in positional head data block containing a "P" to denote the start of the positional data ring (see Figure 6.7). An initial check of word 5 in the head data block can easily lead to a search of the position data ring where the desired position information is found by comparing names. This structure provides a hierarchical organisation, easy access to position information and maintains the system modularity.

6.4 Path definition for motion primitives

6.4.1 Introduction

In some application areas (such as laser cutting, arc welding, and peg in the hole assembly operations) movement along a precise spatial path is of paramount importance to the accomplishment of that task. For this research a key element of path definition is that the X-Y-Z relationship of a curve is specified so that the end-point of a higher order motion primitive can follow that profile as closely as possible. A curved path can be expressed by a spatial equation reflecting the relationship of the X, Y and Z coordinates in a Cartesian frame. Usually, an equation can be explicitly expressed in the form of $Z = f(X,Y)$, where Z

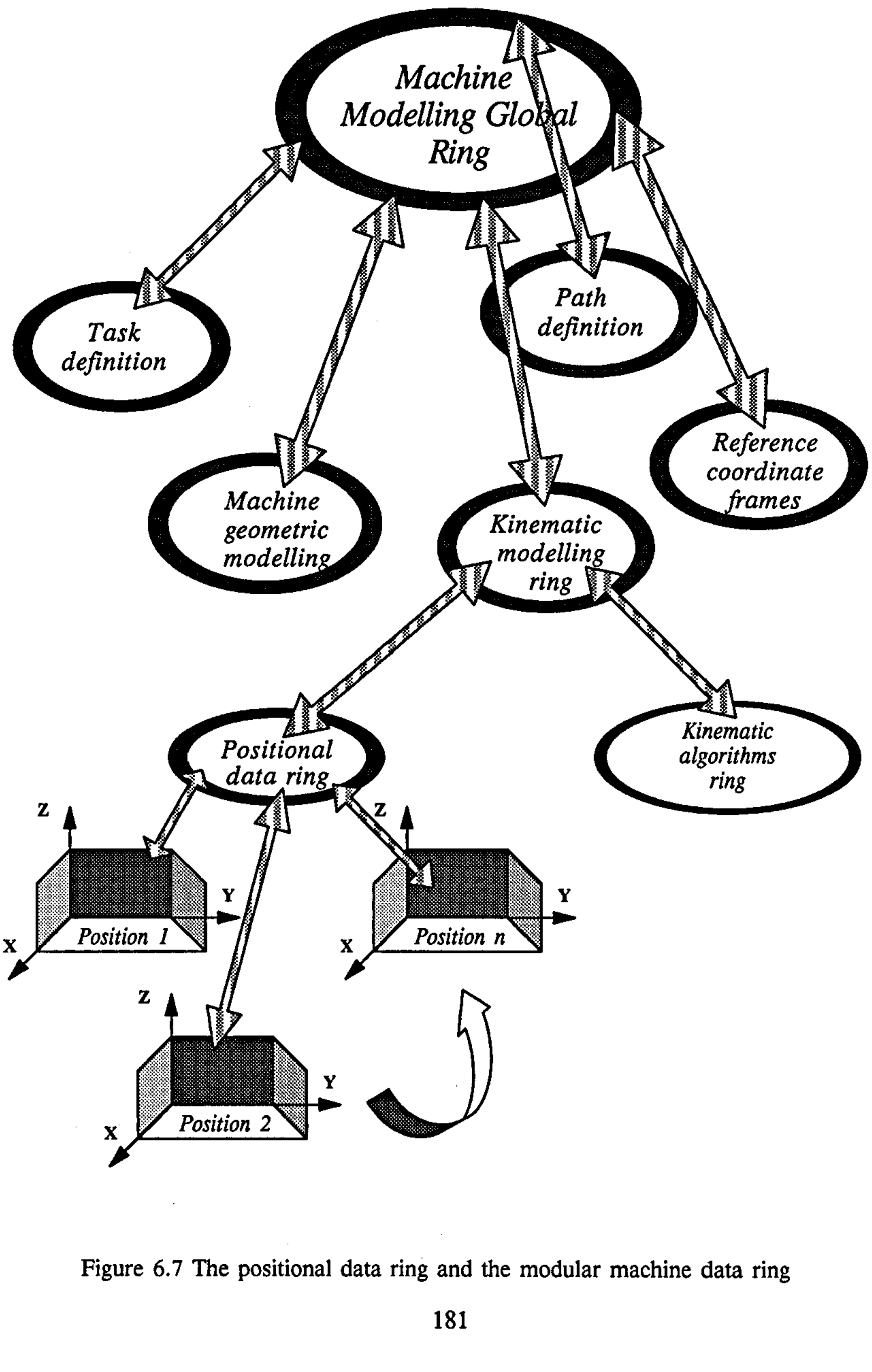


Figure 6.7 The positional data ring and the modular machine data ring

is unique function of X and Y. There are situations where such an explicit expression is not possible. However a complex curve can be subdivided into several segments and corresponding equations derived to characterise those segments, where each of them is unique and can be expressed explicitly. Only curves which can be explicitly represented are considered here.

There are two issues which need to be considered to avoid ambiguity and multi-solutions for obtaining Z_i with given X_i and Y_i in 3-D space. The first is that the segmentation of a curve is precisely defined by its $Z = f(X, Y)$ function and its initial parameter values, i.e. the valid region of (X_i, Y_i) to (X_{i+1}, Y_{i+1}). Incorrect segmentation and initialization can lead to an incorrect path specification. The second aspect is that the equation must possess the uniqueness characteristics referred to above. Otherwise, a multi-specification of a curve is likely and can cause confusion when achieving computer control of the machine. In the case of distributed motion primitives, a three dimensional curve can be realised and sub-divided into two dimensional curves by using two or more distributed motion groups to constrain the path complexity; rather than using one single complex device like a multi-axis robot. Properties of these two dimensional curves are discussed in the following.

6.4.2 Definition of two dimensional curves

A two dimensional curve in the X-Y plane can be expressed by one or more algebraic equations of the form $Y = f(X)$, $X_i<X<X_{i+1}$, where i=1, 2, 3,...n. Each of these equations can be stored in one modular machine database block to describe paths along with their initial conditions. In this study the path data block structure was chosen so that it belongs to the path sub-tree in the modular machine hierarchy (see Figure 5.4) and starts from the

path data head block which specifies the number of paths for the machine and introduces the first curve from its Curve Ring Start of word 5. At the level of the path head data block, if a complete curve can be expressed in a single algebraic expression, then the curve data block stays at the same level at its head data block; if a curve consists of several sub-curves, then a new sub-curve data ring is introduced from a Sub-curve Ring Start word in the path head block of its parent, where the same number of sub-curve data blocks are created to achieve information storage of these sub-curves. A possible curve example and its data representation data ring is illustrated in Figure 6.8. All equations describing each segment of the curve are stored in their corresponding sub-curve data blocks.

6.4.3 Common path primitives

The paths mentioned in the last section are closely related to curve types and their initial conditions, i.e. these paths are motion primitive specific move paths. However there exist some common types of paths which are motion primitive independent and can be generalised as path classes for many motion situations. These common paths can be classified into a number of path types. They can be modelled and stored as primitive building elements of more complex paths, and then during machine modelling they can be selected as required. The path types modelled and included in this study were as follows:

(1) a straight line path type which requires the end-point (WCP) to travel along a straight path passing through two defined positions. The two points in Cartesian space can be arbitrarily defined and it is the straight line pattern which is common to this type of motion, therefore straight line motion is defined as a common path.;

(2) circular paths which require the motion primitive end-point (WCP) to pass through three points which are not co-linear. These types of paths are characterised by a common circular feature, and the specific circular path is determined by the location

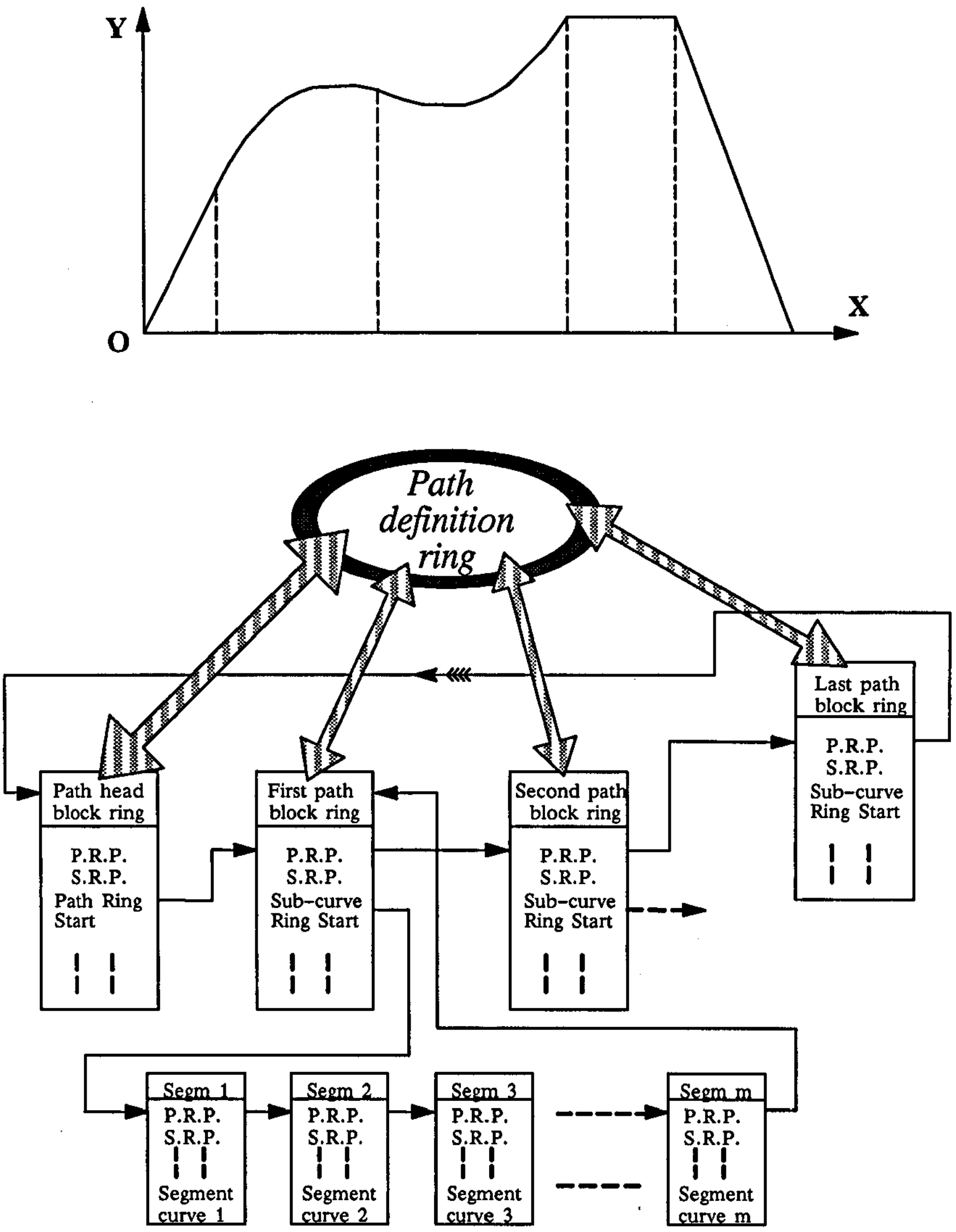


Figure 6.8 Path profile and its data block representation

of the three points. For the straight line and circular paths a corresponding path data block is created and included within the path data ring;

With these two types of path definition, along with the use of equation based paths as discussed in 6.4.2, a set of path definition tools were provided for modular machine configuration. Combination of these paths can be defined at the simulation stage to define spatial relationships for motion primitives as required. This offers great flexibility when specifying the geometry of different manipulator motions. In the next section, kinematic definitions of motion are considered.

6.5 Specification of velocity and acceleration

The maximum permissible velocities and accelerations of machine mechanisms are vital determinants of the cycle time with which motion related tasks can be accomplished. Flexible methods of describing such parameters are required to enable comparative studies of modular machines and to cater for various velocity and acceleration requirements. Although the velocity and acceleration of certain mechanisms used in manufacturing machines may not be accurately controlled, it is practical and appropriate in many situations to modern control technology to enable the precise control of these variables at least for selected mechanisms.

6.5.1 Kinematic specification at the level of single degree of freedom motion

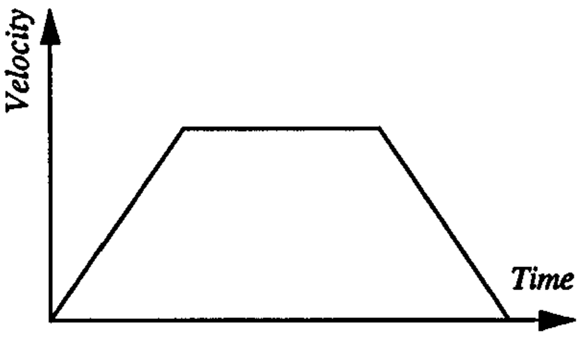
The specification of velocity and acceleration can be discussed at two levels, namely the single degree of freedom axis level and the device level. It is straightforward to define the velocity and acceleration for a single axis in terms of either translation or rotation profiles. At this level, the user has a clear view of the actual kinematics of that axis. For a device with only one single motion axis or a distributed device constructed with same type of axis, this is an ideal method. For rotational motion the angular velocity and acceleration of a revolute axis can be easily converted into equivalent linear values. If the motion primitives of a multi-degree of freedom manipulator system move sequentially, the same clarity of kinematics remains. To facilitate proof of concept kinematic modelling in this study the first five types of the six typical velocity profiles illustrated in Figure 6.9 were implemented to enable control of velocity and acceleration at the single degree of freedom level.

6.5.2 Kinematic specification at the multi-axes level

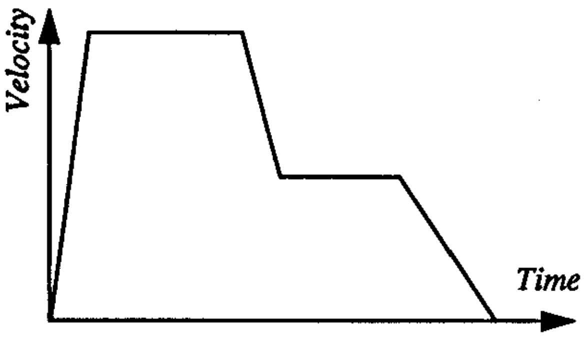
For serially chained or articulated manipulator systems, there is not the same level of clarity and simplicity with respect to the device's end-point velocity and acceleration since the kinematics of such a device's end-point (WCP) is determined by a combination of the velocities of its constituent axes and the configuration of those axes. If several motion primitives are activated at the same time, the end-point velocity becomes the compound velocity of all motion velocities, which can be classified as follows:

(1) the resultant velocity and acceleration of a manipulator system, which comprises prismatic axes of motion is determined by the addition of the various axis velocity and acceleration vectors (see Figure 6.10.1). In this situation neither the compound velocity nor acceleration will have a dependence on the geometry of motion axes. In this study the velocity profiles were stored in a velocity data ring.

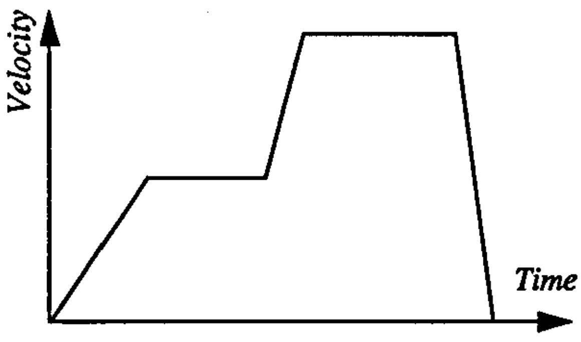
(2) devices with one revolute axis and the rest prismatic, have only one direction along which the velocity and acceleration are straightforward. Since the revolute axis rotates around its local coordinate axis, it can only contribute two of three elementary velocity and acceleration components respectively along two directions (X and Y; or Y and Z; or Z and X) in Cartesian space. The velocity along the rotating coordinate



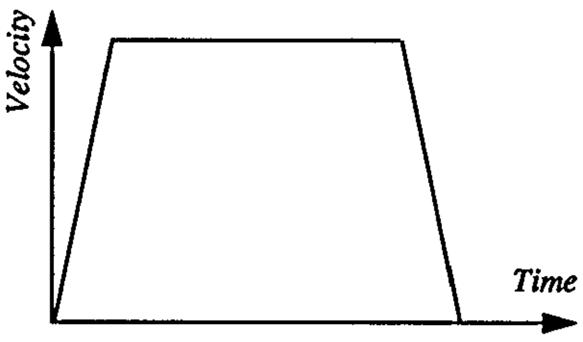
(a) User specified duration of a move



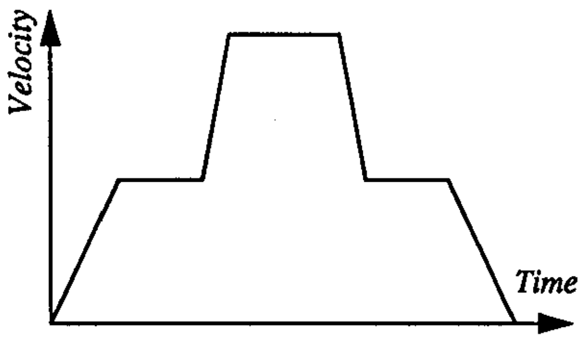
(b) Fast start and slow finish



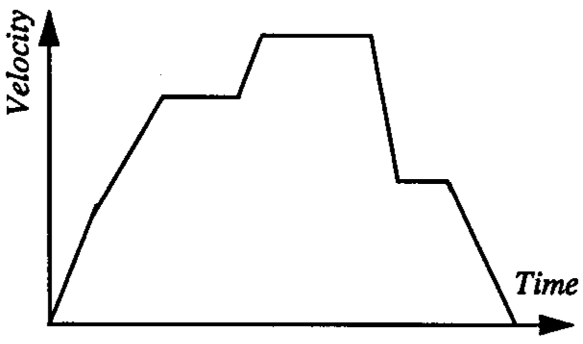
(c) Slow start and fast finish



(d) Minimum duration velocity profile



(e) Slow start and finish and fast motion in between



(f) User defined velocity profile

Figure 6.9 Velocity profiles of motions

axis is determined by the motion primitive along which direction it translates (Figure 6.10.2). The component velocity and acceleration compounds are not constant because the rotating axis changes its two contributory elements in the two Cartesian directions.

(3) devices with two or more revolute axes in their configuration present a very complicated problem when deriving the end-point velocity and acceleration. The kinematics of such manipulator systems are determined not only by the articulated axis configuration at a particular instant in time but also by the geometry of the individual axes. For instance, the velocity of a three degree of freedom articulated primitive is much greater when each of its axes are fully extended than when the axes are in a retracted position. The compound velocity and acceleration of the end point will not be constant and this will require calculation for each axis configuration at each instant in time. This calculation can be very difficult as it depends on the axis geometries and configuration.

The velocity for a higher order primitive with three prismatic axes has been implemented in this study and the same methodology can be used to derive solutions for (2) and (3).

6.6 Conclusion

This chapter began by identifying the fundamental mathematical equations for deriving the spatial transformations of motion objects. Subsequently the parameters required when modelling a modular machine were determined, thereby providing an interface for a user to determine machine characteristics. Three key elements required when defining a motion were considered along with methods of specifying parameters of those elements. The need for various coordinate frame systems was also illustrated to facilitate kinematic definition of machines.

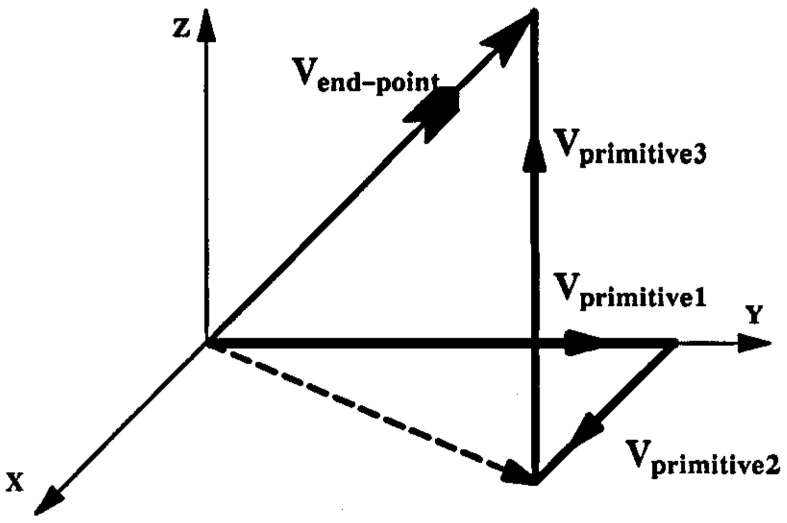


Figure 6.10.1 The analysis of velocities for axis group with three prismatic axes

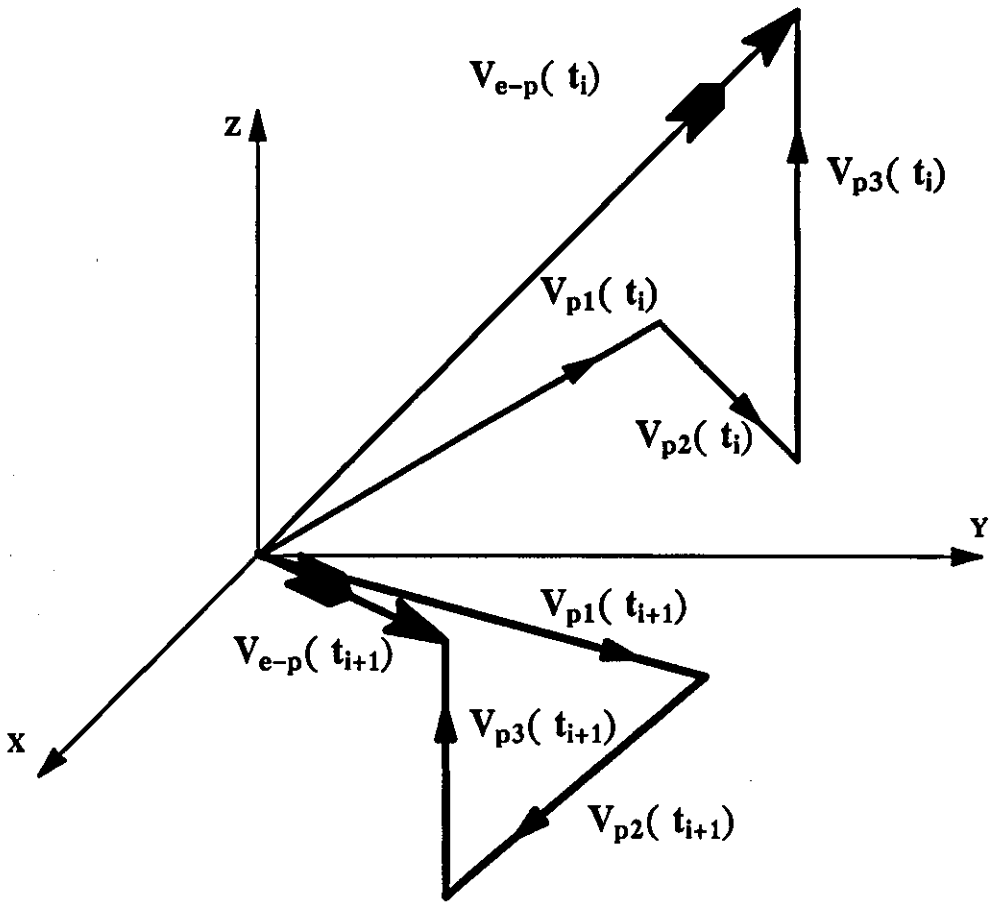


Figure 6.10.2 The analysis of velocities for axis group with only one revolute axis

Although comprehensive geometric and kinematic modules of machines can be created using the facilities described, there is a requirement for modular machine control mechanisms, which can control the machine motion and simulate run-time control. However transformation equations for different device configurations have been derived which allow a modular machine model to be manipulated in a design environment according to a user's specification. The next chapter discusses the control and drive of a modular machine simulation model, and various software mechanisms to meet complex user requirements.

Chapter 7 Control and drive of a modular machine model

7.1 Introduction

The various motions discussed in the last chapter must be generated by forces exerted in the physical machine's drive systems. The dynamic properties of physical manipulators can be characterised by determining relationships between forces and motions in two respects; namely solution of the forward dynamics (being the calculation of the accelerations of motion axes, which are then integrated to obtain the manipulators' velocities and coordinate positions in response to the applied forces or torques), and the inverse dynamics (being the calculation of forces or torques which must be exerted on a manipulator to achieve the desired coordinates and associated velocities and accelerations) [Featherstone 1987]. Starting from well known physical laws, there are essentially two approaches towards obtaining a dynamic model of a manipulator system, namely the Newton-Euler approach and Lagrange's-Euler approach [Armstrong et al. 1986, Stone 1985, and Kane and Levinson 1983]. These approaches have been briefly discussed in the Chapter 3.

In this chapter, a strategy for simulating a modular machine is proposed and the necessary aspects of manipulating a modular machine model in the simulation environment are discussed based on the use of simulation tools implemented according to the discussion of previous chapters. The methodology and strategy of a kinematic simulation approach to modelling both articulated and distributed mechanisms are outlined. The coordination of motion primitives through reference to the operation of sensory primitives is also described. The simulation of motion concurrency is discussed as are some application areas of the integrated simulation system so created.

7.2 A control and driving structure for modular machine models

Following the assumptions described in Chapter 4, graphical simulation and animation of modular machines can be concentrated on describing kinematic properties, only attempting very limited dynamic modelling. As stated earlier, the motion primitives can be manipulated without applying forces in the simulation environment. Thus a systematic strategy for simulating the operations of a modular machine implemented in this study took the form of Figure 7.1. Since there is no serious concern about the time that the modelling system spends on the various simulation activities, a comprehensive top-down hierarchical data structure was adopted. This facilitates the management and manipulation of objects with their relationships and interrelationships represented by the data structure and associated kinematic calculations being executed during simulation. Although such a simulation approach may take longer than one using structures with fewer levels of hierarchy, it enables simulation of various activities, including motion communication, coordination and sensory information processing. A detailed description of control and driving of a modular machine model is illustrated as follows.

A modular machine task was decomposed into several sub-tasks, i.e. events which achieve the target task by involving sufficient modular machine devices in a cooperative manner. It was considered necessary to synchronise and control the sequence of events followed by each task where in the physical system these events may occur concurrently. In this study a scheme was implemented where the event controlled at the very beginning of the simulation creates an event data block for each event of the first task based on the information provided by the command type of an event description. One example of an event command type which was implemented is the **Move** *primitive-name* <u>distance</u>. The

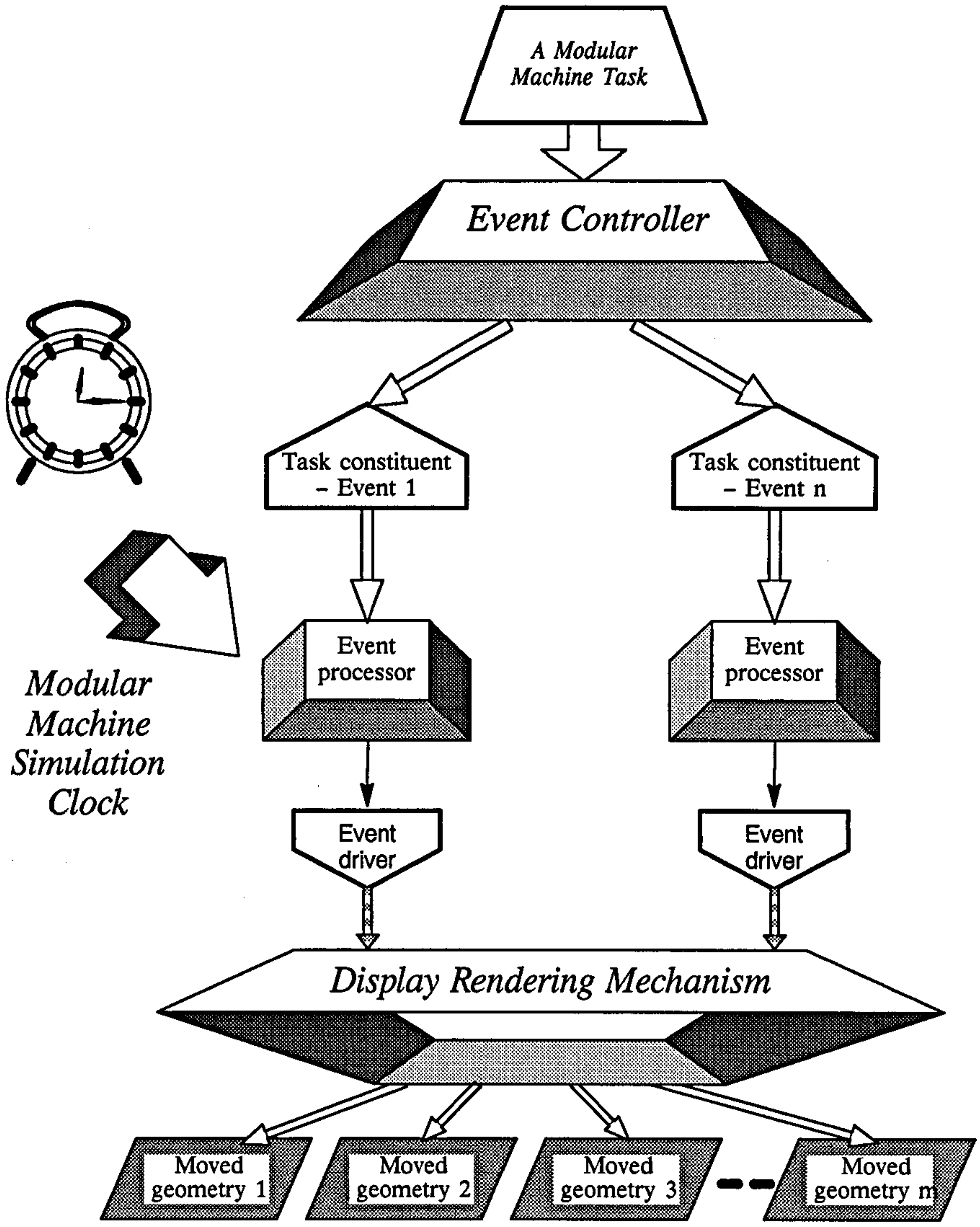


Figure 7.1 The simulation strategy of modular machines at animation stage

event controller interprets this command to achieve control of the simulation and also creates other useful information according to the command type. Table 7.1 shows an example of data table for a motion event. The simulation clock provides a time base for the simulation and when it starts to run, the event controller checks the status of each event data block. If the block is **Active**, then the primitive's event processor starts to calculate the new state of the modular primitive corresponding to movement in the user specified time interval, based on the initial or previous state. If the primitive has a **Waiting** status, then the controller ignores this event until the event status becomes **Active**. An **Idle** state is used to denote the end of an event. Once the event controller finds an event is idling, it creates another event data block in accordance with the next command for that model primitive. In terms of event based simulation, the time spent on the creation of event data blocks is not counted, as it is an overhead which will not be occurred with the actual modular machine. The new state of a primitive (or machine element), e.g the new position of a motion primitive, or the new status of a sensory device, is then passed to the driver of the specific primitive. The driver changes the state of that primitive in terms of its data content in the appropriate position of its data block. At this stage, the information for the particular primitive relating to a given new moment of time ($t_{i+1} = t_i + \Delta t_i$) is available and can refresh the graphical model. However, the remaining processors for other primitives in a modular machine may still be referencing the "old" moment of time (t_i). It is therefore necessary to return the processing time to the beginning of the processed event simulation until all events in an **Active** state are processed. The Display rendering mechanism finally updates the state graphic (position and status) of all active event related machine primitives by reference to the data blocks. At the end of this computational loop, the simulation moves onto its next time cycle. The flow chart relating to this procedure is detailed in Figure 7.2.

Word No	Data Contents	Comments
Word 0	Type & length	Data type and length
Word 1	P R P	Principal Ring Pointer
Word 2	S R P	Secondary Ring Pointer
Word 3	N B P	Name Block Pointer
Word 4	D S F	Display Status Flag
Word 5	'A' or 'D' or 'W'	Denote for "Active", "Idle" or "Wait"
Word 6	'Q' or 'S' or 'L'	For "Sequential", "Synchronization" or "Loosely coupled"
Word 7	Primitive1 ptr	Primitive1 head data block pointer
Word 8	Primitive2 ptr	Primitive2 head data block pointer
Word 9	Primitive3 ptr	Primitive3 head data block pointer
Word 10	'P' or 'R'	Axis1 type (Prismatic or Revolute)
Word 11	'P' or 'R'	Axis type of second axis
Word 12	'P' or 'R'	Axis type of third axis
Word 13	Start position X	Starting position for the event of the X direction
Word 14	Start position Y	Starting position of the event along the Y direction
Word 15	Start position Z	Starting position of the event along the Z direction
Word 16	Goal position X	Target position of the event along the X direction
Word 17	Goal position Y	Target position of the event along the Y direction
Word 18	Goal position Z	Target position of the event along the Z direction
Word 19	Increm. for axis1	The calculated increment for first axis after time interval *t*
Word 20	Increm. for axis2	The increment for axis 2 after the time interval *t*
Word 21	Increm. for axis3	The increment for axis 3 after the time interval *t*
Word 22	Time interval *t*	Time interval *t* specified by user or the smaller left time
Word 23	Max. velocity 1	Maximum velocity for axis 1 specified at axis creation
Word 24	Max. accelera. 1	Maximum acceleration for axis 1 specified by the user
Word 25	Max. velocity 2	Maximum velocity for axis 2
Word 26	Max, accelera. 2	Maximum acceleration for axis 2
Word 27	Max. velocity 3	Maximum velocity for axis 3
Word 28	Max. accelera. 3	Maximum acceleration for axis 3
Word 29	Scaled velocity 1	Scaled velocity 1 based event type and longest motion time
Word 30	Scal. accelera. 1	Scaled acceleration for axis 1 by longest motion time
Word 31	Scaled velocity 2	Scaled velocity for axis 2 based on the longest time
Word 32	Scal. accelera. 2	Scaled acceleration for axis 2 by the longest motion time
Word 33	Scaled velocity 3	Scaled velocity for axis 3 based on the longest motion time
Word 34	Scal. accelera. 3	Scaled acceleration for axis 3 based on the longest time
Word 35	Start time	The start time of the event
Word 36	Duration	The execution period of the event
Word 37	Elapsed time	The elapsed time before this time interval *t*
Word 38	Time interval	The user defined simulation time interval *t*
Word 39	Empty	Empty

Table 7.1 The data block contents of an simulation event

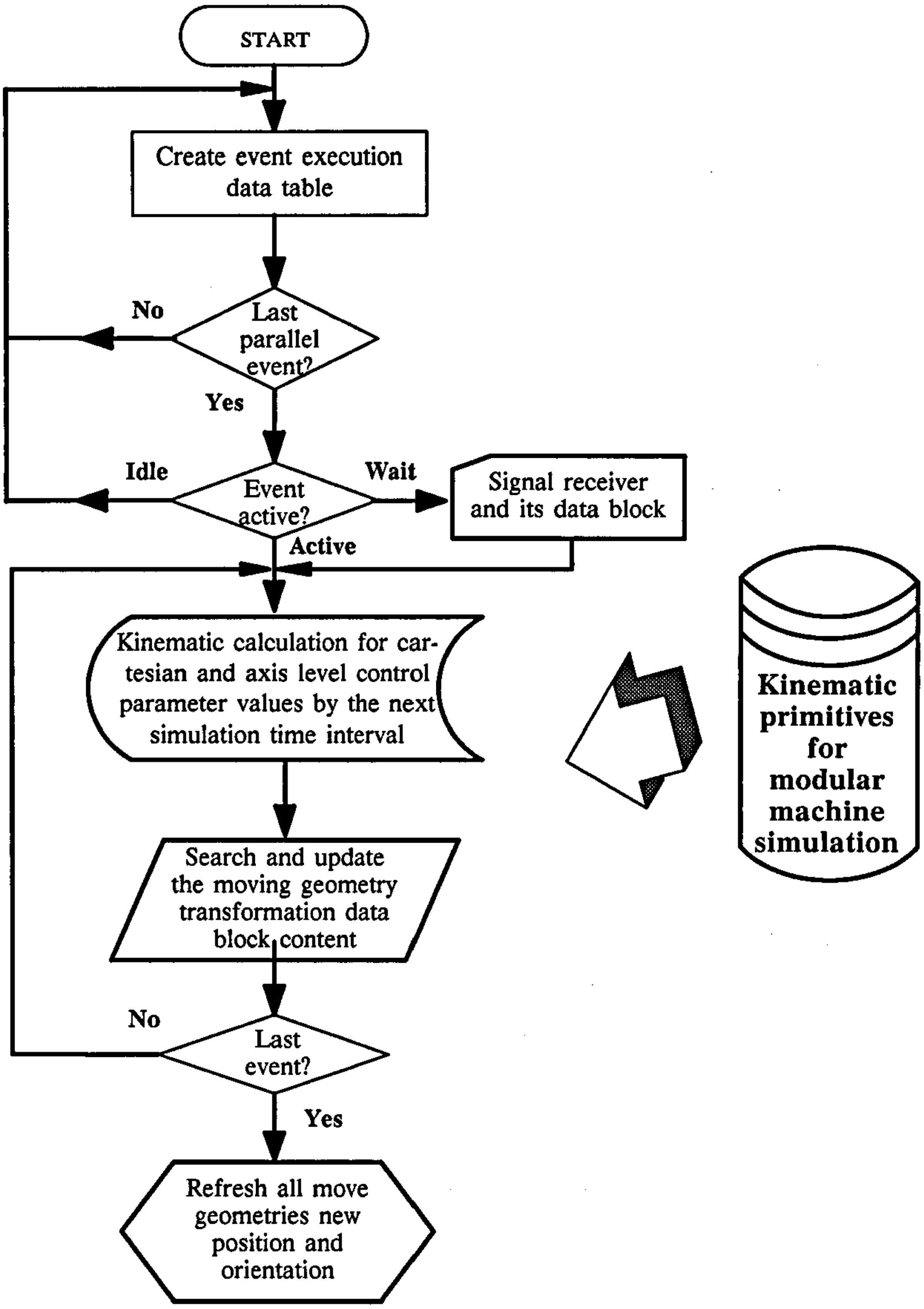

START
Create event execution data table
Last parallel event?
No
Yes
Event active?
Idle
Wait
Active
Signal receiver and its data block
Kinematic calculation for cartesian and axis level control parameter values by the next simulation time interval
Kinematic primitives for modular machine simulation
Search and update the moving geometry transformation data block content
Last event?
No
Yes
Refresh all move geometries new position and orientation

Figure 7.2 The schematic of event control and animation processing

7.3 The modular machine simulator

The simulation system implemented in this study operates in a "control and drive" manner, administering the execution of event-based application tasks. It includes a simulation time based clock, event controller, event processor and event drivers. A detailed description of these constituent parts is given below.

7.3.1 The simulation clock

The simulation of modular machines is essentially a time-based activity; and therefore it must be possible to refer the three dimensional kinematic information to a time base. The simulation clock only calculates the time the machine primitives actually spend in carrying out events. Other non-event related delays (such as the computing time spent on the creation of the event data table, calculation of the motion kinematic solutions etc.) are not counted since they will not be replicated in the real modular machine activities. This is based on the assumption that the physical controllers obtain their control parameter values without significant delay. Therefore the "simulation time" is the sum of durations spent on each event in sequence. The minimum time unit (or time between 'ticks') for the simulation clock is set to a user specified time interval at the beginning of the simulation. With this time interval, the clock then advances its time by one unit after the event controller finishes each execution loop of the whole simulation environment. If the time duration left to simulate an event is less than the specified time interval, the event controller sets the time duration left for that event as a new time interval, thus allowing the new state of the whole model to be obtained at the end of the new time interval. Once this time advancement is completed, the user specified time interval is re-set to the simulation minimum time unit, to allow the same procedure to be repeated for the next simulation cycle. At the end of a simulation exercise, the clock stops and indicates the total time taken by the modular machine to finish its specified task.

7.3.2 The simulation event controller

The simulation event controller is a mechanism based on a control program and it administers tasks in the simulation environment. The simulation event controller accomplishes the following:

i) Creation of event data block tables based on the event command information when the previous event is complete or the simulation has just started;

ii) Initiation of execution of an event related processor;

iii) Provision of an information service as required to machine primitives; such as the time elapsed; a list of active primitive names; a list of currently executing event names, etc.;

iv) Maintenance of coordination amongst single events, i.e, coordinating several event executions with respect to time;

v) Establishment of concurrent motion control information for the event data table of each motion axis, such as initiating several devices at different locations in the simulation environment to achieve parallelism;

vi) Production of accurate synchronisation of several motion axes by indicating the motion coordination type and scaling the velocities of other motion events based on the longest time event, during which the slowest axis completes its event (or motion).

7.3.3 The simulation event processors

The specific execution of an event, which is issued by the event controller, is carried out by one of the event processors according to the event type. These processors can create new machine primitive control parameter values based on the information contained in the event data table. This is enabled by these processors having direct access to the data structures of machine primitives. Such an arrangement for processors (to be able to communicate with the machine model) improves the efficiency of the resulting simulation and reduces the burden on the event controller (the event control concentrates on the control issues rather than on data manipulation). Since each machine primitive needs different processing capabilities, it is easier for a processor to obtain the control value with respect to the primitive control algorithm. For this study, event processors of the following types were created:

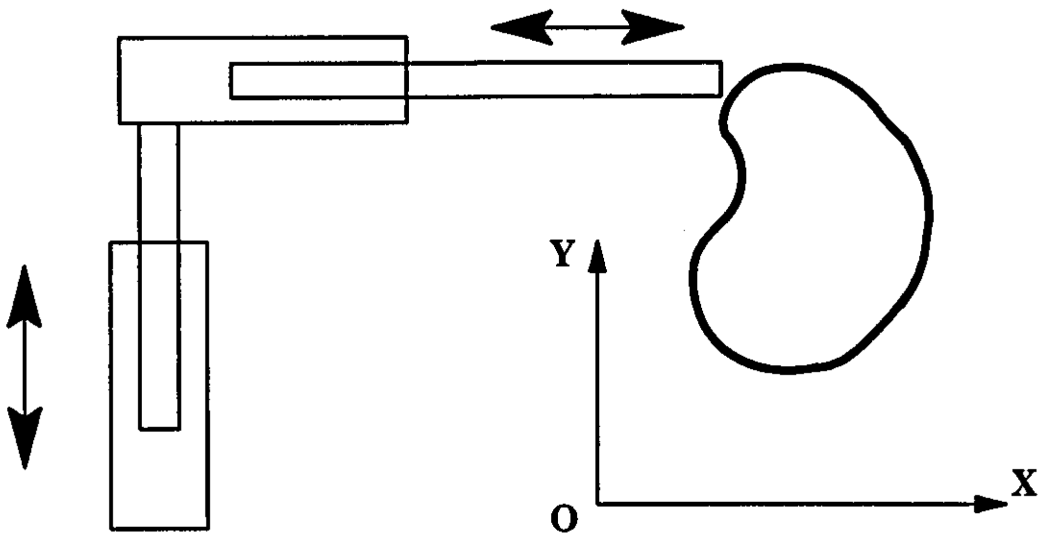
i) Single axis motion processors, which calculate the new position of the current primitive by the end of next time interval where that calculation is based on the current position and a knowledge of the velocity of the axis.

ii) Articulated higher order primitive motion processors, which obtain the new position for the end-point of the primitive in terms of the primitive local frame. The velocities for each individual axis of motion are determined by the event coordination type of the higher order primitive. If the event uses sequential or loosely coupled coordination, each axis will independently reference its own specified velocity. However, in the case of synchronization the velocities are determined and scaled based on the longest time of flight that an individual axis of motion (belonging to the articulated group) takes in moving to the required position.

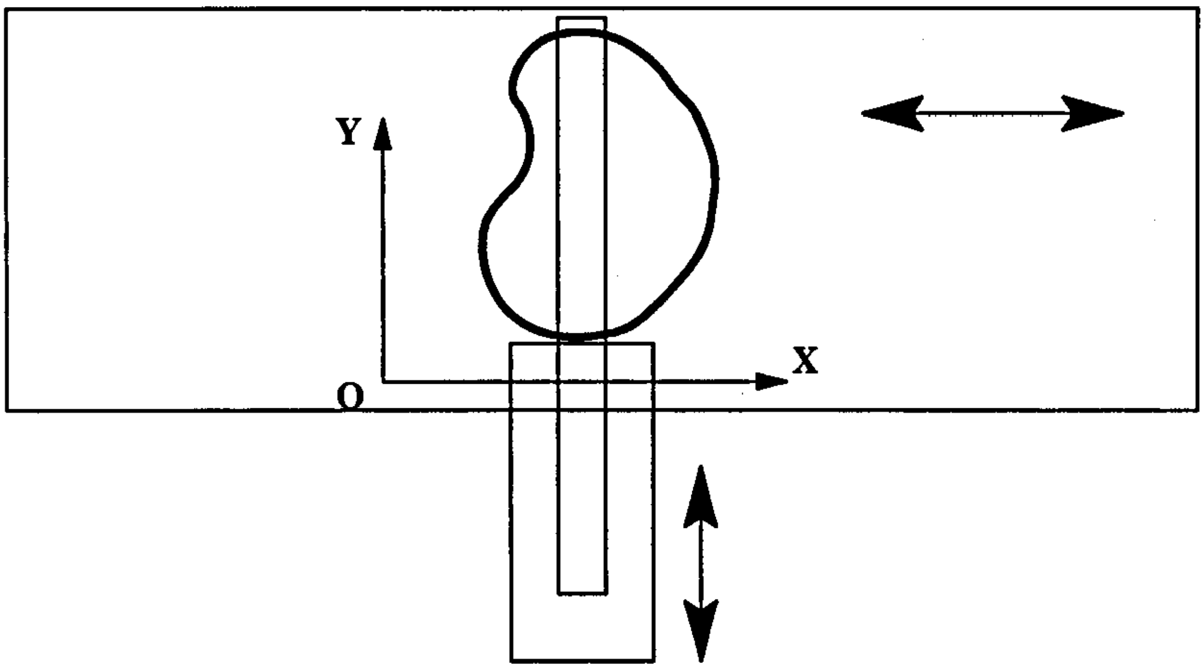
iii) Path event related processors, which are required to calculate the new control parameter values for the axes of motion which will move during the next time interval in compliance with the path requirement. Three dimensional paths are achievable, but the current research concentrates on two-dimensional paths. Since the path is the result of two or more relative motions, two approaches can be employed to achieve a two- dimensional path, viz: using an articulated high order primitive motion or using a distributed (physically decoupled) device. Figure 7.3 demonstrates the principles involved here.

iv) Distributed device processors, which deal with the complex machine situation where the machine comprises several physically distributed machine devices. In such a case several processors may be involved in establishing motion calculation and animation.

v) Sensory processors which cope with the signalling of crucial state changes in the simulation model. Based on the spatial and dimensional information describing the model and its constituents, the functioning of sensory devices (such as positional sensors, distance sensors and contact sensors) can be simulated. For example a presence /absence sensor can simply detect the existence of an object at a specified position in the simulation environment. If an object has reached a defined position, then it sends a presence signal (e.g. a 'one' state on a line). Simulation of such a sensor can inform the simulation system that the instantaneous distance between the specified object and the sensor is some preset value. Contact sensors then report if there is a close contact between a specified object and the contact sensor. The use of 3 D boundary representation can enable the detection of objects, and can ensure that all these three kinds of sensors can obtain their positional information and send a signal correctly. Three types of



(1)



(2)

Figure 7.3 The two dimensional path follower of articulated axis chain (1) and distributed axes device (2)

processors were created, corresponding to the above mentioned sensory devices to perform their sensory functions and details of these sensory processors can be found in section 7.5.

7.3.4 The simulation event drivers

In this context a driver will be considered to be a mechanism which provides a standard interface between the temporal logic (or algorithms) of the machine control and the realization of that logic or algorithm in the simulation environment. With respect to each type of control processor described in section 7.3.3, there exists a corresponding driver for each processor depending on its specific features. Each driver executes the machine primitive activities based on the driver's specification. The reason for establishing a driver mechanism is that there is a need to deal with various types of physical manipulators in a standard and consistent way. The Universal Machine Control approach has derived such a method in the real time control phase [Weston et al. 1989a]. The establishment of standard interface mechanisms (drivers) between processors and graphical primitives in the simulation system can facilitate the future integration of the simulation system and other graphical CAD-based systems. It also enables the physical controller to utilize some of the control data at the real machine control stage. The integration aspects are discussed further in Chapter 9.

7.4 The path based control mechanism

A two dimensional path is defined as an equation (or a set of equations) at the path definition stage of kinematic modelling. The definition is achieved either by direct equation input or by using a computer graphical curve definition facility (in this study the

Face creation facility in GRASP is used). The equation (or equations) defined can be retrieved from the path data ring through the Machine Modelling Global Ring (see Figure 5.4). If an axis group has a constant acceleration **a** and constant velocity **v** moving on a defined path, then

$$v = \int_{t_0}^{t_1} a \times dt = a \times (t_1 - t_0)$$

$$d = \int_{t_0}^{t_1} a \times (t - t_0)\, dt = \frac{1}{2}at^2 - a \times t_0 \times t$$

$$= \left(\frac{1}{2}a \times t^2 - v_0 \times t\right) \quad .$$

Distance along a two dimensional curve y = f(x) in the region of x = a and x = b can be obtained from

$$s = \int_a^b \sqrt{1 + \left(\frac{dy}{dx}\right)^2} \times dx = g(x)\,|_a^b = g(b) - g(a) \quad .$$

To establish path control, a symmetric trapezoid (see Figure7.4.1) is assumed, and the distance d can be expressed as

$d = v \times (t_1 - t_0) + K_1 \times t_0^2$,

where K_1 is the slope of the acceleration section of a trapezoid motion profile (see Figure 7.4.1).

Let d = S, the total time spent on a symmetric trapezoid path is given by

$t_2 = \dfrac{v^2 + Sa}{va}$, where S is the total length of a defined path, v and a are constant velocity and acceleration respectively. The reader can consult appendix D for details of deriving t_0, t_1 and t_2, d and S equations for an assumed trapezoidal path.

With the desired velocity v and acceleration a (see Figure 7.4.2) t_2 is the estimated time

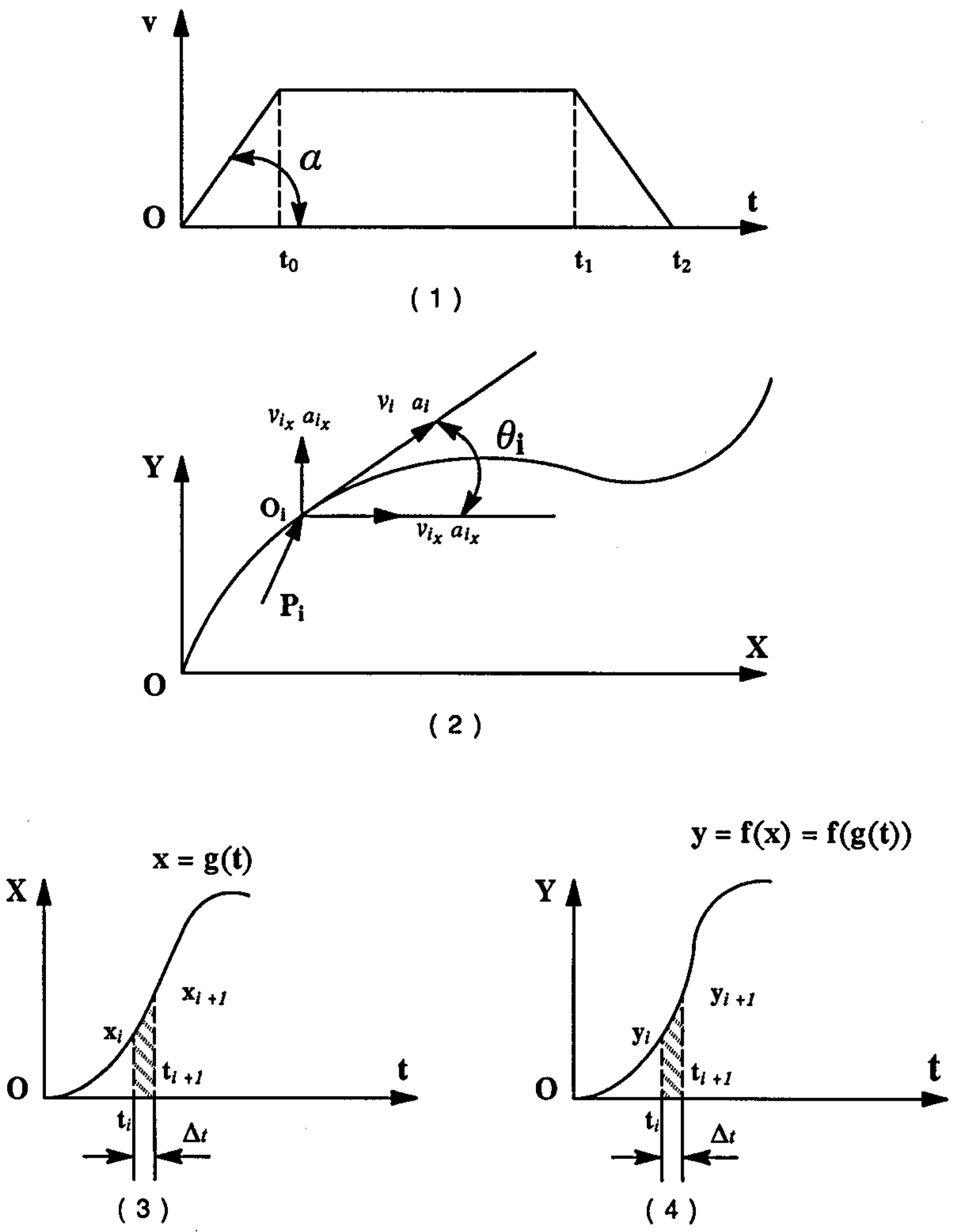


Figure 7.4 The analysis and derivation of the relationship between the path distance along X and Y direction and time *t*.
(1) Velocity profile; (2) Path profile; (3) The establishment of X and time t relationship; (4) The Y and time t relationship based on the X=g(t).

spent on the specified path. The next step is to find the corresponding point (x_i, y_i) on the specified path at a specific time t_i. In evaluating the position $P_i(x_i, y_i)$ it is extremely difficult to establish an explicit equation for the relationship between the X coordinate and the curve length S, therefore it is often impossible to calculate the exact point $P_i(x_i, y_i)$ on the path with a given time t_i because from $S = \int_0^x \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx$, it is impossible with most curves to obtain the inverse function x = g(s) where s is a function of time. If an explicit relationship y = y(t), x = x(t) is defined, then the point $P_i(x_i, y_i)$ can be obtained through this set of equations. An approximate relationship describing x = x(t) can be established through the kinematic analysis with reference to the desired path. While the end-point of an articulated motion primitive is moving on a specified path, the defined acceleration and velocity can be divided into two elements along x and y directions. The slope of the curve at the point $P_i(x_i, y_i)$ can be obtained from the relationship $y' = \frac{d}{dx} f(x)$, and θ can thus be obtained. The distance moved along the X direction can be approximately expressed as (see Figure 7.4.2).

$$\Delta x = V_x(t_i) \times \Delta t + \frac{1}{2} ax \times \Delta t^2 = V(t_i) \cos(\theta_i) \times \Delta t + \frac{1}{2} a \times \cos(\theta_i) \times \Delta t^2$$

and

$$x(t_{i+1}) = x(t_i) + \Delta x(t_i)$$

$$= \left(x(t_i) + V_x(t_i) \times \Delta t + \frac{1}{2} \times a(t_i) \times \cos(\theta(t_i)) \times \Delta t^2 \right)$$

where x (t_i) is the distance moved along the X direction at time t_i ;

V_x (t_i) or $V(t_i)\cos(\theta_i)$ is the elementary velocity along the X direction at time t_i ;

$a_x(t_i)$ or $a(t_i)\cos(\theta_i)$ is the acceleration along the X direction at time t_i ;

Δt is the time interval ;

x (t_{i+1}) is the estimated distance that the motion primitive will have moved after a time interval Δt .

Times t_0, t_1 and t_2 can be used to evaluate the velocity v and acceleration a in Figure 7.4.1 in the above calculations. Using such an approach x (t_{i+1}), y (t_{i+1}) can be easily obtained knowing the path specification equation y = f(x). After calculating values for x (t_{i+1}) (or x_i for short) and y (t_{i+1}) (or y_i for short) on the path, these Cartesian coordinates need to be converted into axis coordinates so that the chosen machine axes of motion can establish the target position on the path. The inverse kinematic transformation equations described in the last chapter were employed to calculate the axis coordinates at time t_{i+1}.

Although the velocity and acceleration at time t_i are used to calculate the distance for time interval Δt, improved accuracy can be achieved by using the velocity and acceleration at the mid-point between x_i and x_{i+1}. This is achieved at the expense of recalculating x_{i+1} with mid-point v and a after the first time calculation of x_{i+1}.

Two restrictions are imposed on the path definition to allow correct execution of the above algorithm. The first is that a path must be continuous so as to maintain the existence of y_i for a given x_i. If a path has a discontinuity, the path should be treated as two separate paths. The second restriction is that although the trapezoid velocity profile is defined with sudden changes, a blend should be introduced at the unsmoothed corners of the velocity profile in order to maintain a smooth acceleration. This is specially true in real time control since jerk and violent acceleration could increase wear and make accurate real-time computer-based control very difficult. Run-time control of the joint path requires smoothness in terms of joint displacement change and the change rate should not exceed allowable limits of joint acceleration. In terms of simulation, the same rules should be applied to model the kinematic motion. The velocity and acceleration elements along X and Y directions calculated in the above equations can be used to compare with the axes velocity and acceleration. If the required acceleration or velocity for any axis exceeds the maximum limit, the Cartesian trapezoid velocity profile should be modified to agree with the axes' kinematic constraints.

7.5 Simulation of sensory device functions

The sensory device processors are also key constituents for responsive simulation. The methodology which these processors employ lies in the correct calculation of an accurate distance between the sensory device and the detected object surface. The simulation of the functional properties of these three types of sensory devices is illustrated as follows.

An object has its own local coordinate frame which establishes the location of the object at a desired position in the simulation environment. Based on the relative position of the object and a sensor, the distance between the sensor and the object (frame) can be easily obtained. Here the "positional sensor" is defined and characterised as one which predicts an object position relative to a coordinate frame coincident with the sensor. This frame can be a common reference frame for other objects or motion primitives. See Figure 7.5.1.

Another type of sensor which will be defined and characterised is a "distance sensor" which predicts the distance between the approaching surface of an object and the distance sensor, this being based on the local frame orientation of the object and the relative position of the object and sensor (see Figure 7.5.2). This type of sensor is used to measure a distance so

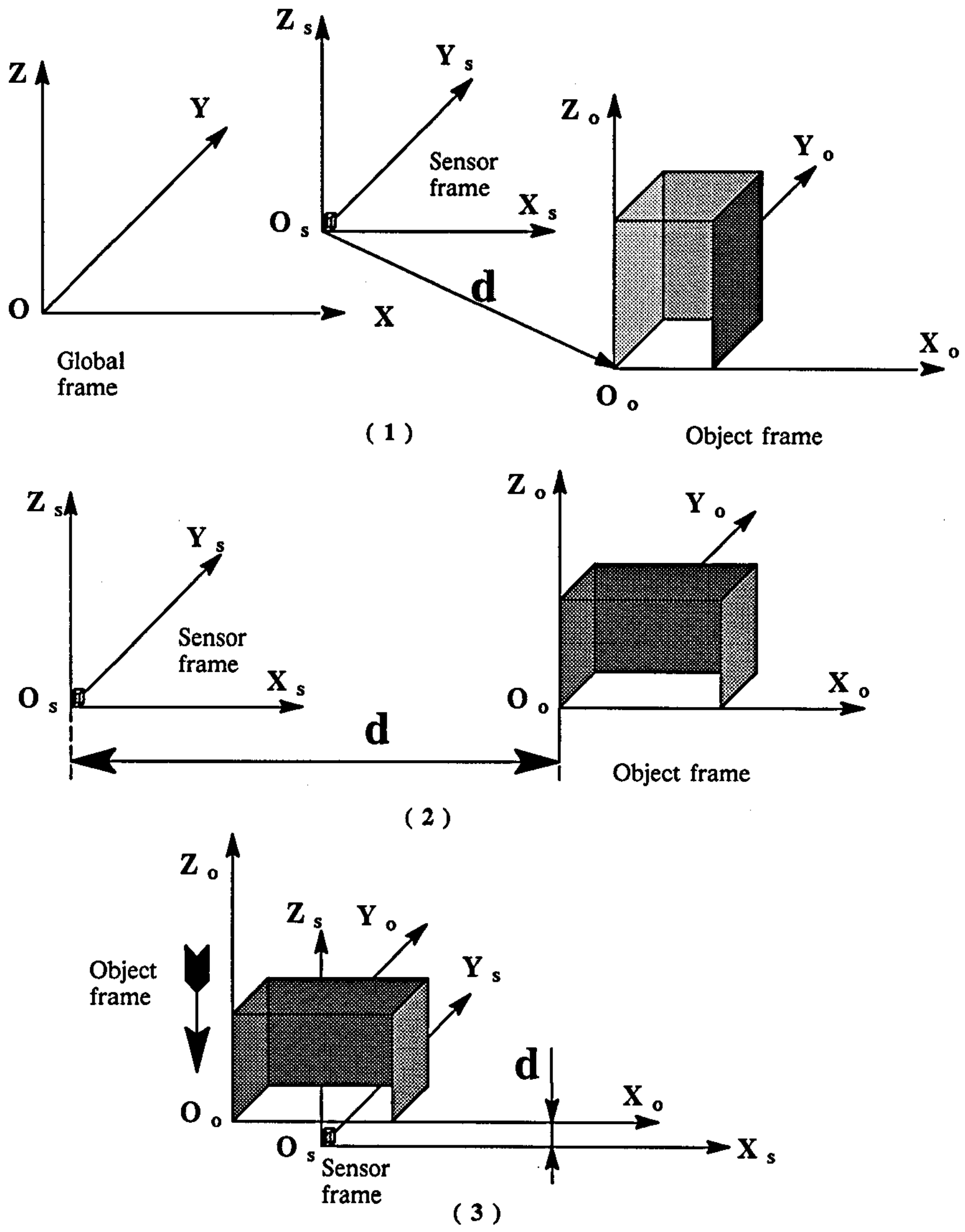


Figure 7.5 (1)The coordinate frame system for a position sensor; (2) the coordinate frames for a distance sensor; (3) the coordinate frames for a contact sensor

that associated processing capability can make decisions based on the distance obtained. One example application is the use of a distance sensor to simulate a switching action corresponding to the distance between the object surface and the sensor becoming zero (e.g. a part arrived at the sensor position) or indeed some non-zero value as required.

"Contact sensors" will be defined and characterised as ones which detect the relative position of an object and a sensor, where that distance is very small and usually embedded in the surface of another object (see Figure 7.5.3). A contact sensor is mainly concerned with the normal distance between an approaching object and the sensor. If the distance equals zero, then the sensor can transmit a signal to the event controller to stop the approaching motion to the sensor surface or to make other decisions. This type of sensor is mainly used for collision detection of some collision-prone parts in the simulation model.

In the case of distance and contact sensors they function in one direction only which is defined at the sensor creation stage, and hence the calculation is made easier by arranging the directions of these sensors co-incident with that of the local coordinate frame of sensory related objects. Each sensor has its own local frame, therefore it is possible to locate the sensor with a desired orientation to facilitate its measurement. The direction of a sensor is always chosen to be the approaching direction of a detected object. If more than one dimension distance detection is required, one or two more sensors can be embedded in the same sensor to expand the sensor dimensional capability and form a compound sensor. There is no consideration in the current implementation of the orientations of these sensors relative to a detected object.

Since dynamics are not considered in this thesis, force sensors and their simulation is also beyond the scope of the discussion. Since the condition of any of the above mentioned sensors can be determined at any time within the time interval of the user specification, the time interval should be substituted by the remaining time (less than one time interval unit) before a condition is reached. The remaining time is obtained according to the velocity of the approaching motion and the distance to be traversed in the next time interval. In this study the distance moved during the previous time interval and the distance between the object and the sensor are kept in the sensor data block for comparison. Therefore the exact time instant can be calculated before the object will move through the sensor work space.

7.6 The distributed device processors

A special class of processor is required to co-ordinate the operation of distributed devices, be that in the simulation environment or when achieving run-time control of the real machine. This type of processor is characterised by the need for co-ordination amongst motion primitives which are physically distributed. The physically distributed layout requires the establishment of spatial relationships between devices, and the need for coordination requires precise definition in the device event data block. A primitive event data block is created for each machine primitive to enable the event controller to execute the device task properly. These data blocks form a local event data ring and the first or head event data block in the ring points to another event data block relating to another device, i.e. another motion primitive with separate event or a sensory event data block. The complete set of data blocks then form a possible two level event data ring which is attached to the kinematic modelling ring (see Figure 7.6).

With the aforementioned distributed local event data ring, the distributed event processor

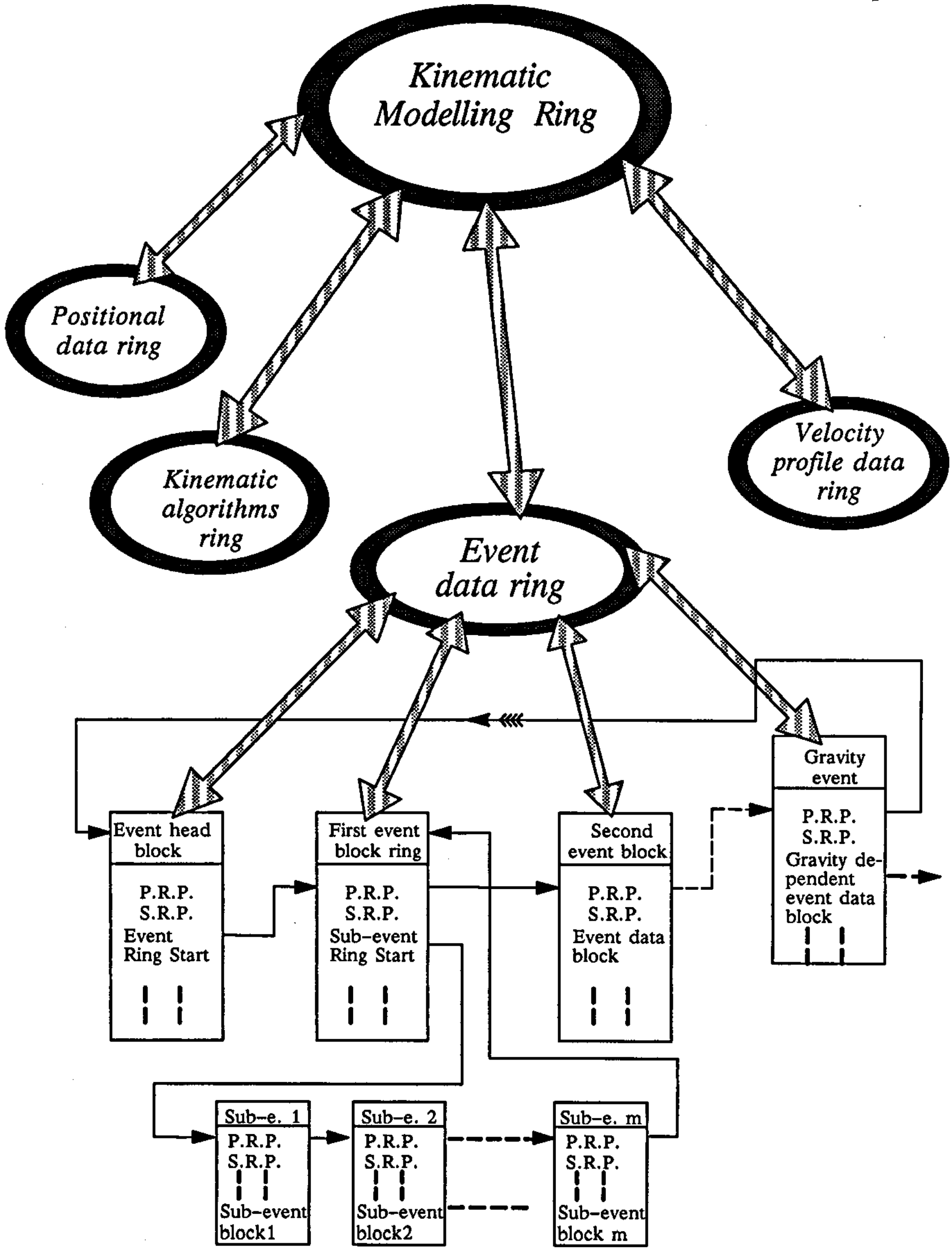


Figure 7.6 The kinematic modelling and the event data ring structure

takes over the event controller's responsibility (at distributed device level only) and checks the event type. Once the sub-event in a distributed device event is identified, the distributed event processor calls a sub-event dependent processor (in this case as a sub-processor for the sub-event) to execute the sub-event. This sub-processor can be a single motion processor, a high order primitive processor or a sensory processor. Once this sub-event is finished, the sub-processor returns its next coordinated sub-event data block address and the distributed event processor checks the new sub-event type before calling the next sub-processor based on the sub-event type. This procedure repeats until all active sub-events in the distributed event advance their states and time base by another simulation time interval. At this stage, the distributed event processor returns its administration responsibility to the event controller and the other top level events in the event data ring are evaluated by the event controller in order to advance their states and time base to the current time. In the present implementation, the sub-event block for a distributed device uses a data block similar to that of the single primitive event block (containing 20 words). However coordination between machine primitives requires a more descriptive event data block hence a similar data block to that used for higher order primitives was adopted and found to be appropriate (i.e. 40 words in the block).

7.7 Motion and processing concurrency

Motion concurrency is characterised by several devices in motion at the same time and requiring concurrent processing. This capability is of great importance in improving the efficiency and capability of a manufacturing machine. Manufacturing concurrency has been achieved for a long time through hard-wired control (parallel processing) and mechanical parallelism in various types of machine. The approach is commonly employed in accomplishing large batch manufacturing tasks where there is seldom much flexibility required or available (in terms of the frequent changing of the concurrency relationships).

For conventional hard automated machines however product changes often cause very high levels of reconfiguration overhead. Computer controlled modular machines can provide sufficient flexibility to enable product changes whilst also offering operational concurrency. This form of concurrency exists at two levels, viz: the device level and machine level. At the device level operation concurrency is typified by the distributed device event where the single primitives in the device can achieve two forms of concurrency, namely loosely coupled co-ordination and close synchronization of several motions. Concurrency at the machine level is largely enabled by the physical layout of the modular machine - i.e. any arrangement of distributed devices. The layout of modular machines can be designed such that complex tasks are decomposed into multiple instances of simple sub-tasks with motion groups assigned appropriately to sub-tasks. Accordingly the joint and other machine primitive groups can be decomposed from a possibly complex machine for complex tasks into a number of simple devices. In terms of each joint group, they perform simple sub-tasks. However the combination of a number of relatively simple sub-tasks can be aggregated to configure a complex task [Kusiak et al. 1990].

Two levels of motion concurrency are essential at the simulation stage in order to emulate practical situations. The low level (device level) concurrency is at the device motion primitive level (as described in section 7.6), where the parallel operation of several single degree of freedom primitives is enabled. The machine level motion concurrency is achieved by advancing the simulation clock at the same interval for all active devices in the model and this is reflected into the design of the two level event data ring structure. The execution of all active primitives in each device is organised in such a way that when the execution of sub-events in one event are finished the next event at the same time interval is executed until all active events are advanced. In terms of real time computation, the above execution of events is a sequentially based method. However in the context of scaled simulation time motion, concurrency and parallel device event execution is achieved and if the available processing power is sufficient "real time" simulation is achievable. Computation time is not a critical criterion during simulation since no real time control response is required. However such a real time simulation system could prove highly beneficial for run-time control. The benefit of this having this type of event concurrency during simulation is that it can provide an estimate of real time machine performance and can provide assistance in the actual machine event control and task planning.

7.8 Example application of the simulation system

In this section, the control of some frequently used devices and primitives, such as motion axis group end-effectors (grippers), gravity feeders and conveyors are discussed to exemplify the design methodology and tools created in this research study and described in this chapter.

7.8.1 Control of a gripper

A gripper mechanism suitable for attachment to a group of axis primitives can be modelled by aggregating two or more single motion primitives on a gripper base (see Chapter 4). The control requirements of the gripper can be characterized by the gripper type, its motion constraints and motion sequence. In this study, grippers were modelled so that the gripper type specifies the number of fingers and the number of motion primitives on each finger etc., the motion constraints define the working range, velocity and acceleration features of each motion primitive, and the motion sequence defines the motion sequence of each axis. This information enables the gripper processor to calculate its complete operation cycle (open plus close duration). With the execution of a gripper operation event, the gripper processor advances the spatial state of the gripper with reference to its Work Centre Point (WCP) by the simulation time interval. Grippers with two fingers represent a simple case where two motion axes usually rotate or slide symmetrically and synchronously. Once one control parameter for one finger is obtained, the magnitude of state change is the same for the other except that it is in the opposite direction.

The control of more than two motion axes resembles distributed device control where more than two groups of distributed fingers (devices) interact with each other in a coordinated manner. The main difficulty is the specification of target positions for specific fingers of a hand. Once the position and orientation are defined, the kinematic equations for corresponding high order articulated primitives can be used to calculate the forward and inverse transformations. To simplify the position definition for each individual finger, an approach based on establishing a finger tip grasping shape was concerned and adopted in this research. As indicated by the comparative study of human hands, the grasping of manufacturing components can be roughly divided into two types, viz: circular (radial and symmetrical) grasping and prismatic (opposed parallel) grasping [Cutkosky et al. 1990, Li and Sastry 1988]. For the circular type of grasping (see Figure 7.7.1), a circle passing through the tips of all fingers can be established as a variable of the hand. All finger positions can be defined with reference to this circle and the fingers always move towards or away from the centre of the circle as if in contact with the variable circle. Prismatic grasping is characterized by the parallel and opposite movement of fingers (see Figure 7.7.2). A straight cylinder perpendicular to the direction of finger axis motion is created as a reference for this type of grasping. The position of the fingers is then defined with reference to the centre line of the cylinder.

Based on knowledge of the hand structure (circular or linear), it is easy to determine the other finger's position: once one of them is calculated and the fingers are always in contact with the variable cylinder. The inverse transformation equations can be used to calculated each axis control value.

7.8.2 The control of gravity feeders

The effect of gravity is an important issue in simulating the motion of any mass related object simply because it affects the spatial relationship of an object with reference to its simulation environment. A gravity component feeder (see Figure 7.8.2) is a typical example of a device where gravity effects spatial relationships. Once the first component at the "ready to be picked up" position is removed, gravity forces the remaining components in the feeder slider to fall until they are stopped by some end-stop mechanism. This effect can be simulated by checking (at every simulation time interval) the distance between the next component and the "ready to be picked up" position along the Z direction of the feeder local frame. If the next component is located above the "ready to be picked up" position, gravity effect starts to come into play. The distance moved under gravitational forces in the vertical and horizontal directions, with respect to simulation time, are respectively (see Figure 7.8.1)

$$D_v = \frac{1}{2} \times a_v \times t^2 = \frac{1}{2} \times g \times \sin(\theta) \times \cos(\theta) \times t^2$$

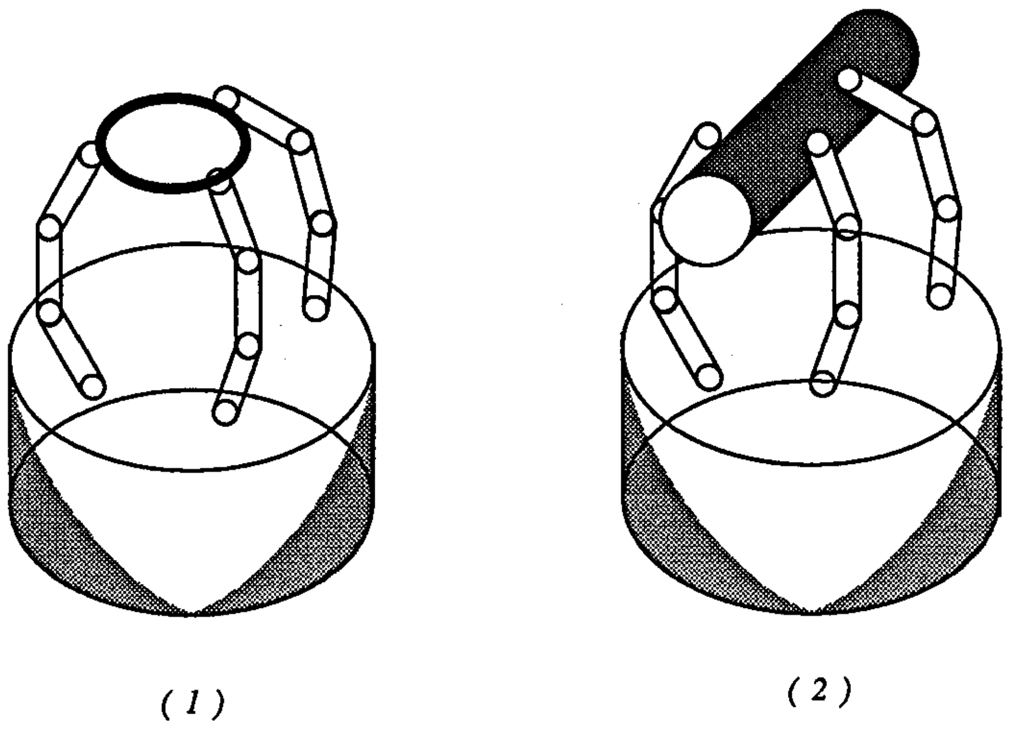


Figure 7.7 The circular grasping *(1)* and the prismatic grasping *(2)* of typical hands

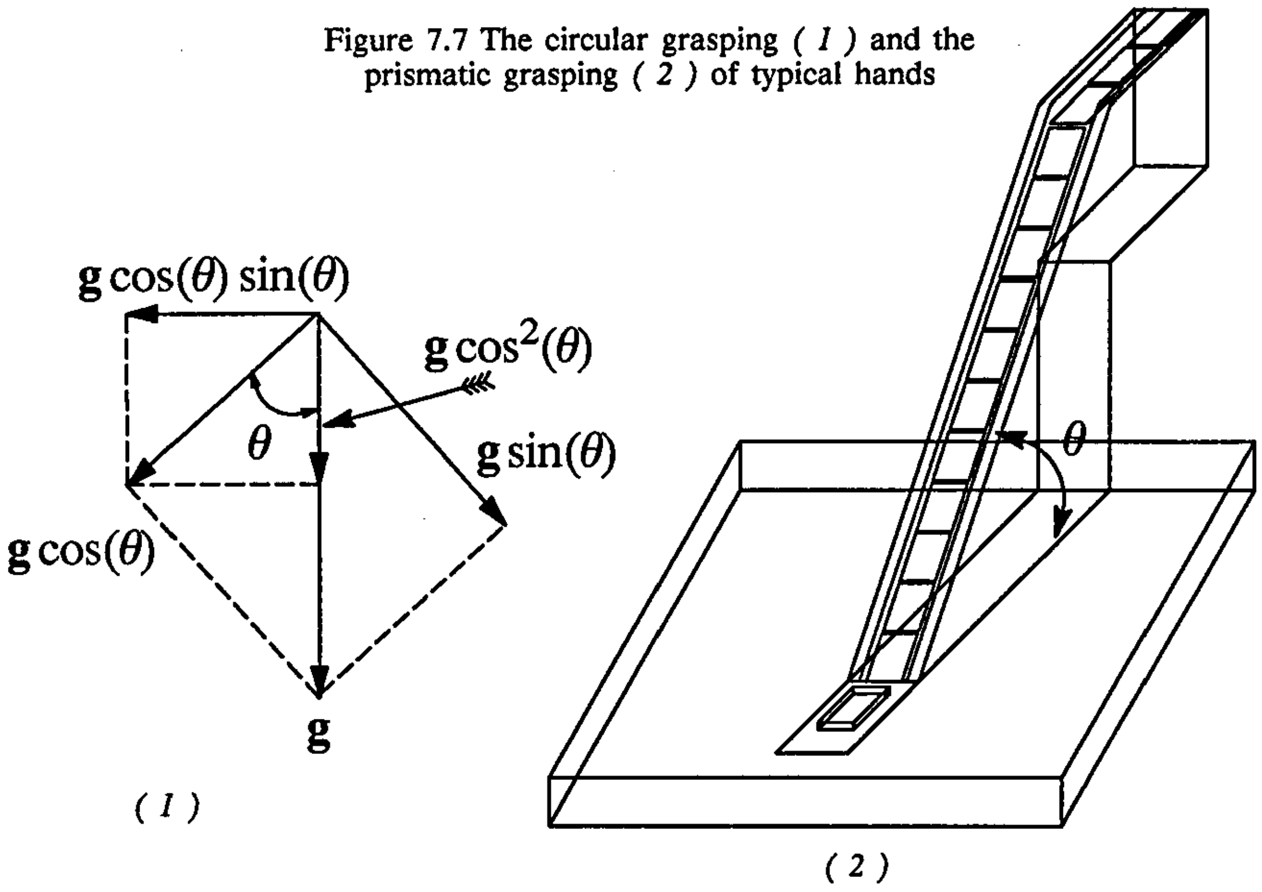


Figure 7.8 A gravity feeder *(2)* and its side view of the analysis of gravity acceleration *(1)*

$$D_h = \frac{1}{2} \times a_h \times t^2 = \frac{1}{2} \times g \times \sin(\theta) \times \sin(\theta) \times t^2$$

where, a_v and a_h are the acceleration components of gravity along vertical and horizontal directions respectively, and t is the time interval.

At the end of each simulation time cycle, the event controller activates the gravity processor to advance the state of any objects subject to the gravity effect. Gravitational characteristics are assigned to individual components or component groups (if all components in the feeder slider are defined as an array) before animation of the model starts.

A similar methodology can be applied in checking the effect of gravity at the model design stage. If an object does not belong to the *workplace* or another objects and its mass centre is not on the top of another object, then in the simulation it falls onto either the ground floor (*workplace* X-Y plane) or the top of some object which is vertically below the falling object. If an object belongs to another object, then the root object of this group of objects (data ring) is checked with respect to the gravity effect. This function was only implemented for a small number of object models but the methodology can be extended to a complex model at an expected high cost in computation time.

7.8.3 The control of conveyor devices

Conveyors are widely used as transportation equipment in various industries and especially for assembly tasks. Conveyors can be classified into the following types based on their motion activating types:

- constant moving conveyors;

- condition moving conveyors which start to move when certain conditions are satisfied.

The first situation is very straightforward as a simple event can be defined to activate the conveyor following which the conveyor then moves continuously. However the condition based conveyor waits for an activating signal from the event controller. Typically sensors can issue activating/deactivating signals for the conveyor. However the state changes of a sensor must be processed before the sensor processes and activates a conveyor. Otherwise an incorrect model animation can occur. The event controller must ensure that the advancement of each primitive is in the correct sequence.

This chapter has illustrated a major part of the simulation system for modular machines implemented in this PhD study. With the consideration of both articulated and distributed mechanism configuration, various simulation event processors were described in detail which provide an essential core for the simulation of modular machines study. It has also been demonstrated that the adopted simulation methodology can be used to model and simulate both configurations involving sequence based coordination and time based synchronisation.

Chapter 8 Modular machine programming

8.1 Introduction

Modular machines can be used in complex manufacturing operations, and thus a comprehensive methodology of programming is desirable through a user-friendly interface. Programming of modular machines is unlike robot and NC (Numerical Control) machine programming because of the enhanced generality, multi-device environment, flexibility and reconfigurability for different manufacturing tasks. Hence a general approach towards the programming of modular machines is required. In this chapter a programming study is made of the features of modular machines in manufacturing and a general systematic approach towards high level utilization and control of modular machine is outlined. This approach is illustrated through simulation with the possibility of implementation on the physical machine by the utilization of a common data format interface between the simulation and the physical machine control. The common data format interface is important for the systematic integration of simulation, physical machine and other CAD based systems and is demonstrated in detail in Chapter 9.

8.2 Methods used for robot programming

Since modular machines are still at the research stage and little attention has been paid to programming issues, the programming languages and methods are rarely found in the literature although programming of multi-robotic devices in coordinated motion was considered by some researchers [Agapakis et al. 1990]. However a study of robot programming languages can generate some useful ideas for modular machine programming because both types of device are manufacturing automation machines although there are some differences in their configuration methods. As illustrated in section 2.4.7, it has been generally realised and accepted that a robot language is divided into levels. Ideally five levels (see Figure 2.5) are used, where each level of programming implies a different level of abstraction [Dupourque 1986]. Most current implementations of robot programming languages are at the manipulator level and above [Volz 1988 and Wagner 1990].

There are conventionally two common approaches adopted by these implementations, either using an existing language, such as Fortran or C, or a proprietary language such as VAL. The advantage of the first approach is that it permits the full power and benefits of the language to be used. However there are problems with the former approach due to the lack of robot specific characteristics [Gini 1987, Hutchinson and Kak 1986]. The advantage of a proprietary language is that it is customised to the capability of robots but it clearly suffers the disadvantage of being robot dependent and not standardised or generally accepted.

Most recently a new method of deriving a robot programming facility has been proposed and devised under the concept of a programming environment, where a programming system can be coupled to various modelling systems and sufficient information abstraction and sharing between these systems can be enabled. This approach is thought to be an appropriate method towards future robot programming [Volz 1988].

Based on the evolution of robot programming systems, a method derived from the abstraction of a multi-level programming system and a programming language environment was adopted to devise a new programming system for modular machines. A three level programming system was partially implemented within the established modular machine design and simulation environment. Currently an interactive programming facility has been devised and a compiler can be created in the future for interpreting the textural input of a task description of a modular machine. The detailed implementation is described in section 8.3 onwards where the characteristics of modular machine programming are considered with reference to articulated and distributed device programming.

8.3 An analysis of modular machine operations

8.3.1 Generic manufacturing operations

The actions of a manufacturing machine are characterised by two aspects, namely the machine generic operation and the application dependent operations with respect to the machine's functionality [Volz 1988, Chatila and Giralt 1987, Van Aken et al. 1988, Sanderson and Homem-de-Mello 1987]. The machine generic operations are defined as those which exist to such a wide extent that they can be found in various application machines of automation. From a perspective of manufacturing industry, the generic operations can be abstracted from various specific industries. The assembly industry provides a typical application for such abstraction of generic operations. Due to the time limitation, only motion, sensory and communication related operations in the context of assembly operations were considered in this study.

A motion operation in the assembly industry can be further divided into preparatory and task-achieving operations [Gini 1987 and Laugier 1988]. A preparatory motion is typically used when a component is required to be transferred from its initial position to a position where it is ready to be processed. For example an electronic component is transferred from its feeder to a ready-to-be-inserted position and this transferring motion is a typical preparatory motion which prepares for a task-achieving motion. A task-achieving motion is typified by a device moving to accomplish a manufacturing task (e.g. insertion, material removal and so forth) by associating the motion with the task related objects.

The signalling operation of sensory devices is another type of generic operation. The reason for such a generalisation is that the feature of such operations (carried out by sensory devices) is to sense manufacturing environment changes which are common to all machines. The sensors provide a feedback for the machine controller to make a decision to change the machine state independent of the type of sensors used. A communication operation between a device and a controller is also a generic operation. With the advent of Computer Integrated Manufacturing there is a great opportunity to achieve modular machine manufacturing concurrency and parallelism in order to improve manufacturing flexibility and efficiency. Modular machine manufacturing concurrency is typified by multi-device operations at the same time in a cooperative manner. To be more specific from the control aspect, these devices are coordinated in such a way that one device's operation is possibly dependent on the execution of another device's operation. All devices' operations are arranged correctly in order for the whole machine to achieve the specified task with optimal efficiency whilst physical non-interference between devices is maintained. Success in this accurate coordination relies on correct communication between devices and their awareness of the machine environment. Therefore, in a computerised manufacturing environment, communication among the devices of a modular machine can be considered as a generic operation. This is also true at a manufacturing cell or even factory level.

At the current time, the author believes that the above three types of generic manufacturing operations encompass most of the generic side of manufacturing machines in assembly and other industries, and that these operations can be used as a target for modular machine programming. A wider discussion about application dependent manufacturing operations is given in the next section.

8.3.2 Application dependent operations

Each task in different applications requires special (application dependent) operations to achieve its objectives. Some such special operations are listed as follows:

Spatial operations which change the position of a task related object by non-generic motion. For instance, an electronic component is relocated in another place in an assembly by gravity force;

Geometric operations which change the dimension and shape of a task related object., e.g, machine tools cut the component into the designed shape and dimension from the raw material;

The processing one of the physical properties of an object. For example heat treatment of metal changes the mechanical characteristics of that metal;

Ownership related operations which change the ownership of an object from the previous owner to a new one. Fixture, indexes and jigs are typical examples of this type of operation which are non-gravity dependent owners.

There are many others of which some can be included in physical property operations, such as welding, spraying, soldering etc. Since this research is focused mainly on assembly and related manufacturing industries, the above operations provide a wide enough range to express the needs of such manufacturing activities.

Like the generic operations, the application dependent operations can also involve different levels of a modular machine task description. This feature of requiring a multi-level task description naturally leads to the multi-level modular machine operation description. Some of the application dependent operations can be composed by some of generic operations. In this case, the task description for the generic operations at this constructing level of the operations can be employed by the application dependent operation and all related control parameters can be adopted. However, if this operation involves the machine device level function formation, then an application dependent task description is usually required in order to facilitate the user application description. The manipulator end-effector or gripper is a typical example. The motion of each jaw or finger can be precisely defined by using the generic operation - preparatory operation description. However, at the high level, which is the device level, an application dependent task description is required to simplify the description of the gripper operation. "Open" or "Close" can serve the task description purpose.

However, some other special operations, such as physical property related operations, need special task descriptions which are beyond the combination of machine generic operations. The control of welding parameters can not be described in terms of machine generic operations even if the control of these parameters are integrated into the machine control capability. In these cases, a special set of application dependent task descriptions should be introduced as a sub-set of the whole machine task description.

It can be clearly seen that an application dependent operation is characterized by its special requirement of introducing a new command description no matter at which level it is used. But, due to the generic feature of the four types of generic operations, one set of operation descriptions can be always used at different level of a modular machine in different application areas. This can lead to a general approach towards a consistent machine task description with respect to the machine generic operations. There is a need and opportunity to produce a framework for the generic side of the operation. together with some sub-set of the application dependent tasks and operation descriptions. Thus a robust modular machine programming methodology can be derived.

8.4 An operation-oriented programming methodology with a three level user interface

Manufacturing operations can be categorised into generic and application dependent operations [Laugier 1988]. Each of these categories can be further divided into some specific operations with a common feature. Although this decomposition is described with respect to assembly machines, it is also true of other manufacturing machines due to the machine's common feature (to change a product's geometry, physical property, relationship etc.). As modular machines take advantage of a machine's functional decomposition, the above analysis and classification is even more beneficial to modular machine programming and task description. Since multi-machine or device co-ordination in a computerized manufacturing environment is considered in the above discussion, the distributed devices layout aspect is also encapsulated in the operations discussed earlier.

It is natural that each generic operation exists at each level of the physical machine hierarchy and that the same type of generic operation can overlap two or more physical levels. Because of the hierarchical feature and the operational decomposition, the lower level operations can form a higher level operation which is in the form of some greater abstraction of the task description. The low level generic operation primitive can not only form a higher level generic operation, but can also produce some application dependent operations since the modular configuration is adopted in constructing complex devices from low level primitives. A manipulator gripper can be classified as this type of application dependent operation constructed by two or more low level geometric operations - rotation and sliding. This is especially true if the dependent operation is made of motion operations. The outcome of this analysis completely agrees with the machine primitive decomposition in terms of a machine's physical construction.

Based on the operation analysis and the modular machine function decomposition, the three level operation oriented machine task description method shown in Figure 8.1 is proposed and partially implemented to program a modular machine. At the lowest level (machine single primitive level), the task description is focused on the individual primitive feature, and each device is precisely specified to achieve its function in an exact way. This is a complete level of a machine task description, but is often very cumbersome and logically error-prone. The task description at the device level provides a better user interface and has some intelligence. Along with the low level machine definition, this level of programming can achieve the machine task description with a reasonable efficiency. A higher level of task description is needed to realize high level machine intelligence, and is denoted as task level programming. These three level programming are described in details as follow.

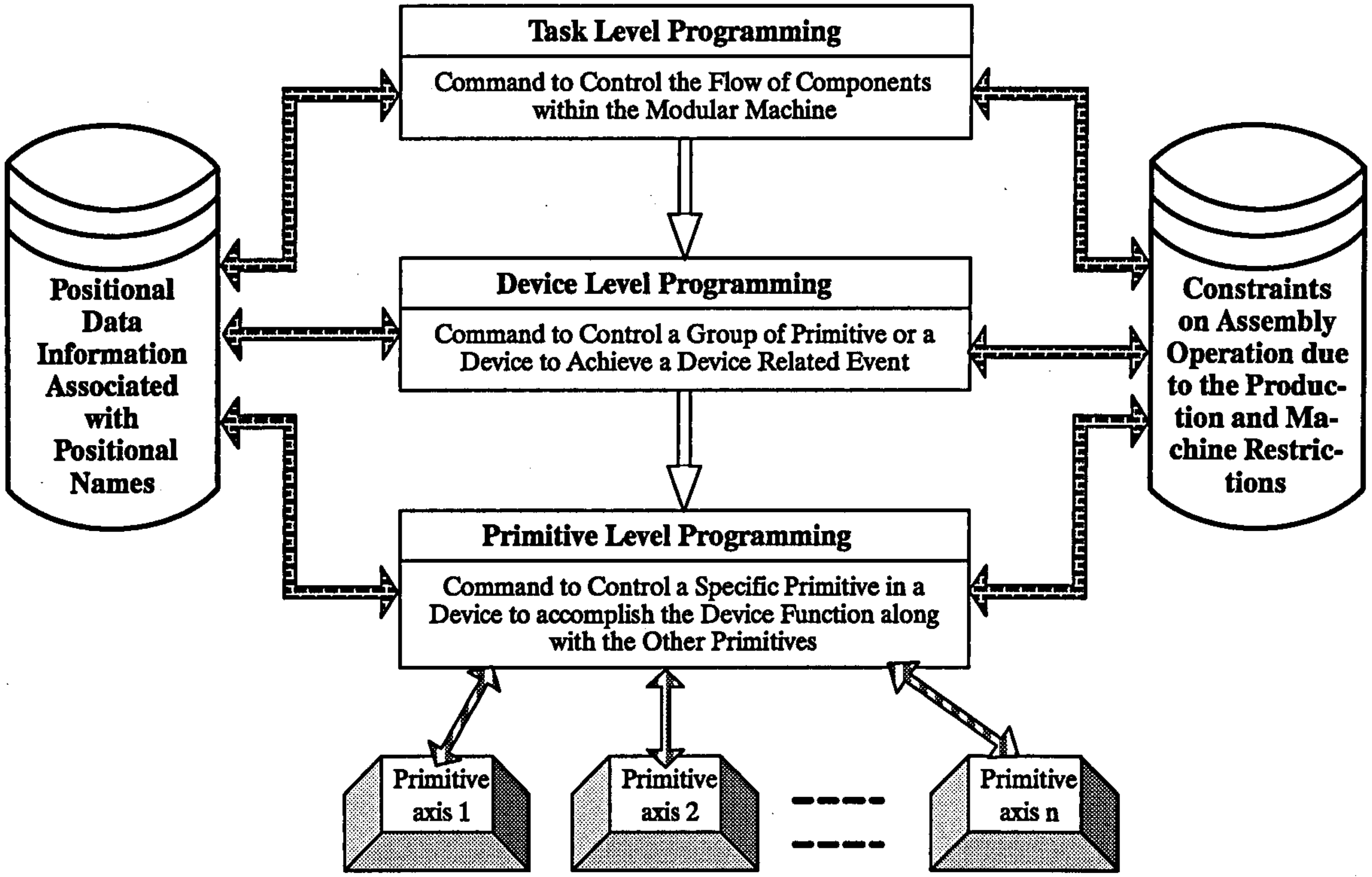

Task Level Programming
Command to Control the Flow of Components within the Modular Machine
Device Level Programming
Command to Control a Group of Primitive or a Device to Achieve a Device Related Event
Primitive Level Programming
Command to Control a Specific Primitive in a Device to accomplish the Device Function along with the Other Primitives
Positional Data Information Associated with Positional Names
Constraints on Assembly Operation due to the Production and Machine Restrictions
Primitive axis 1
Primitive axis 2
Primitive axis n

Figure 8.1 Three level task programming system

8.5 Operation-oriented programming method at the lowest level

Based on the above discussion, a modular machine's manufacturing task can be defined by its operation and corresponding attributes. An operation definition describes the machine's action type, whereas the related attributes then define the action details in a quantitative manner. In this study modular machines were decomposed into a three level hierarchy in terms of functionality, and thus a different criterion is used to measure the operation and functionality at each level.

At the lowest level of the hierarchy the focus of function description of single motion primitives or sensors relies on the isolated view of the single primitive performance. Therefore, the operation of these primitives is described by the primitive function type and its attributes. For example, a single motion primitive's operation for a prismatic joint can be described by the function **MOVE** and the attributes, primitive name, moving distance, velocity, and acceleration. As to the rotation of a joint, the operation type **MOVE** is replace by **ROTATE** and the corresponding attributes are described in the context of the rotation operation. Although there are two types of generic motion operations at this motion primitive level, there is no fundamental difference in describing them both in function type and attributes. The difference appears at the device level discussed in the next section. Sensors are another type of low level primitive which simply send conditional signals to provide raw information for the machine controller to make decisions. The signalling operation at this level was treated as a generic operation as they can be found and used in computerised modular machines in many application areas to achieve low level "intelligence". The function description for a sensor is

SIGNAL *sensor_device_name*;.

The other two types of generic operations - communication and decision making, derived at the machine level based on the device and machine functionality, are not considered at this low primitives level since they are concerned with higher level coordination and management of the machine's operations.

Application dependent operations at the lowest level are defined as those that accomplish elementary operations which are application dependent other than the above mentioned three - sliding, rotation and signalling operations. Sometimes one of the elementary operations forms an abstracted operation at a high level. For example, the mixing of chemical solution for PCB board soldering is one such application dependent operation. Since the application dependent operations at this level often involve other physically and chemically related operations, motion and signalling based operations are concentrated in this study at the low operation level.

8.6 Programming methods at the device operation level

8.6.1 Motion generic operation

At the device function description level, the view point is that of a group of single primitives and their relationships. Therefore the focus of the device description is on the function type and its attributes with respect to the group. Abstraction in the device function description may be possible to a certain extent, but it varies from one device to the other depending on the device complexity. If a device is of a transportation type, such as a group of relationships established between motion primitives to achieve a certain motion type in its working envelope, the function description of the device is still centred on its compound motion description. Since at this stage more motion primitives are introduced in the device, there are two aspects which need to be described due to this structural change.

The first is that the relationship of these primitive motions is of great importance in extending the device functionality. The spatial information is initially embedded in the machine model and this gives one possible method of dealing with the issue. The second aspect is the functional combination of these primitives, and this is defined by the device function description type in conjunction with some of the attributes. The articulated and distributed devices are the main interest of these function descriptions. The articulated situation is discussed in this section and the distributed one is considered later.

For the articulated motion axes group, as described in previous chapters, there are basically three types of primitive motion relationships, viz: *sequential*, *loosely coupled* and *synchronized* motions. The function descriptions for these three motion types were implemented in the abstracted form of

> MOVE *device_name*, motion_type Q (primitive 1,2,3) default type(Q) or L or S(path_name), *target_position*, *velocity*, *acceleration*;.

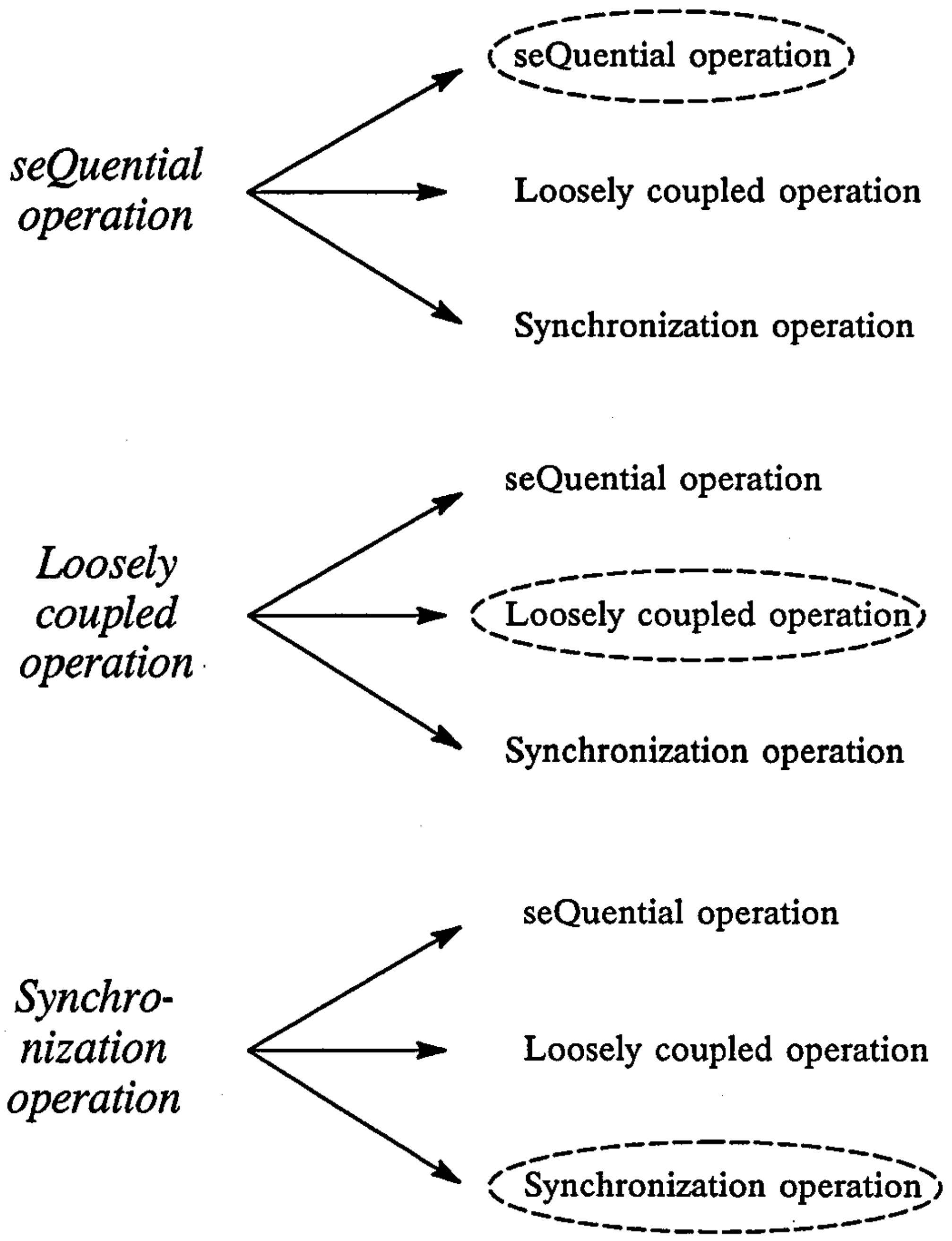
The above form of task description does not necessarily have a complete set of attributes every time when a task is described. For example, when the motion type is sequential, each primitive uses its own maximum velocity and acceleration if the device velocity and acceleration are not specified. The default velocity and acceleration of each motion primitive are also used in the case of loosely coupled motion if the velocity and acceleration are not defined explicitly at this device level. For the synchronized situation, the default velocity and acceleration for the longest motion execution time is chosen to scaled the rest of them based on the synchronization condition if the device velocity and acceleration are not defined.

8.6.2 Operations for the distributed devices

As for the distributed devices (built up by physically distributing all single motion primitives or higher order primitives), there are similar basic motion types, i.e. the sequential motion, loosely coupled motion and closely coupled motion which can be a path required motion [Jouaneh et al. 1990, Maimon 1990 and Tao et al. 1990]. Since motions are relative to each other in this case, the overlap of two distributed motions in an interest plane can produce a defined path as described in section 7.3.3. Apart from these three single motions, there are other possibilities where one higher order primitive in a device can move with any type of the above three motions and rest of motion primitives in the device are co-ordinated with the 'main" motion in one of these three motion types. Hence, three possible combinations can be derived to describe the relationships of motion primitives within a device. Since all types of motions are time based, the combination of the same type of motion as a sub-event repeats the same description and can be replaced by two separate events, hence these sub-events are redundant and initially excluded from the discussion (see Figure 8.2). If any type of event embodies a same type of sub-event motion, it is easy to define such situation as two separate events for the same device. For loosely coupled motion of a device, the sequential motions should be defined explicitly to ensure the correct sequence although this sub-event starts at the same time as other primitive motions. The command in such a situation is

> MOVE *device_name*, L (1st, 2nd, ..nth), *target_position_name*, *velocity_name*, *acceleration_name*; ,

where 1st, 2nd, .. nth are 1st, 2nd, .. nth primitive name of the sub-event, and the sub-event



seQuential operation
seQuential operation
Loosely coupled operation
Synchronization operation
Loosely coupled operation
seQuential operation
Loosely coupled operation
Synchronization operation
Synchro-nization operation
seQuential operation
Loosely coupled operation
Synchronization operation

Figure 8.2 Nine combinations of motion types with the automatic elimination of three same type motion in one device

is embedded in the event description. Since the device is distributed in this situation, the position, velocity and acceleration actually refer to one set of related data which are respectively defined at the position, velocity and acceleration specification stages.

In the situation of a synchronized sub-event in a loosely coupled event, up to three motion primitives can be synchronized to achieve a path specification. In this case, such sub-events can be scheduled into sequential events and executed one after another. However the parallelism is not maintained and the desired efficiency of the task execution becomes poor. There is a need to deal with this sub-event complexity in terms of machine programming. The following command was created to encompass the related information:

> MOVE *distributed_device_name*, L (1st, 2nd, 3rd of the synchronization sub-event), *target_position, path_name, velocity, acceleration*;

where a *path_name* specifies the path which the synchronization sub-event primitive group has to follow. The *velocity* and *acceleration* here define the user expected end-point velocity and acceleration of each primitive or primitive group. If this velocity and acceleration violate the joint velocity and acceleration constraints, the joint velocity and acceleration are used to replace the user specified ones at this level of programming. It is impossible and impractical to require all primitives in a distributed device to move at one single velocity and acceleration since the sub-event nature mainly dictates the motion kinematic feature. The user specified velocity and acceleration are usually treated as a reference velocity and acceleration and therefore are not necessarily used. If a strict kinematic requirement has to be imposed on the motions of primitives in a device, the velocity and acceleration can be replaced by velocity and acceleration data names, where velocity and acceleration data tables are created at the velocity and acceleration definition stage for the device. This can be very useful for a task-achieving operation definition at this level. In most situations, the default velocity and acceleration can be used to simplify the machine programming in preparatory operation and a certain percentage of the default velocity and acceleration can be used as a fine motion for a task-achieving generic operation at this level. The trapezoidal velocity profile can be defined at the kinematic feature creation stage and be referred to by the preparatory and task-achieving operations at this stage. The exact profile is decided by the user's specification (see Figure 8.3). Therefore at this level, the velocity attribute has three options, namely actual reference velocity, velocity data block name, and velocity profile data block name. The task-achieving operation can be embedded into its preparatory operation or defined as separate one.

8.6.3 Signalling generic operation at the device level

The generic operation signalling at device level is initiated by the receiving of the signal. Signal primitives or sensor devices are created at the modelling stage. Once the simulation starts, they function as physical signal devices. The status of a sensor device can only be altered by the satisfaction of its functional description. For example, a proximity sensor changes its signal status when an object exists between its "emitter" and "receiver". Therefore, the operation of sensor devices are activated by the start of the simulation execution rather than by any command.

Meanwhile the effective utilization of these sensory device information requires the ability to perform logical operations on it. Typical logical operations, such as **AND, OR NOT,** need to be included in the sensory information operation. These operations can achieve the

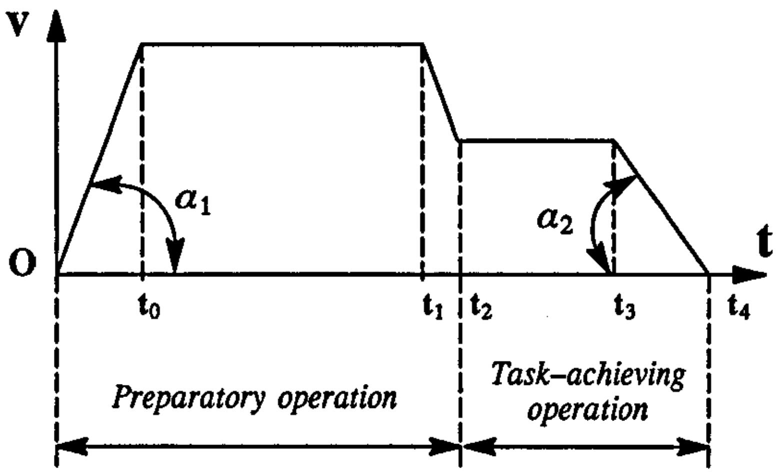


Figure 8.3 The two types of motion generic operations

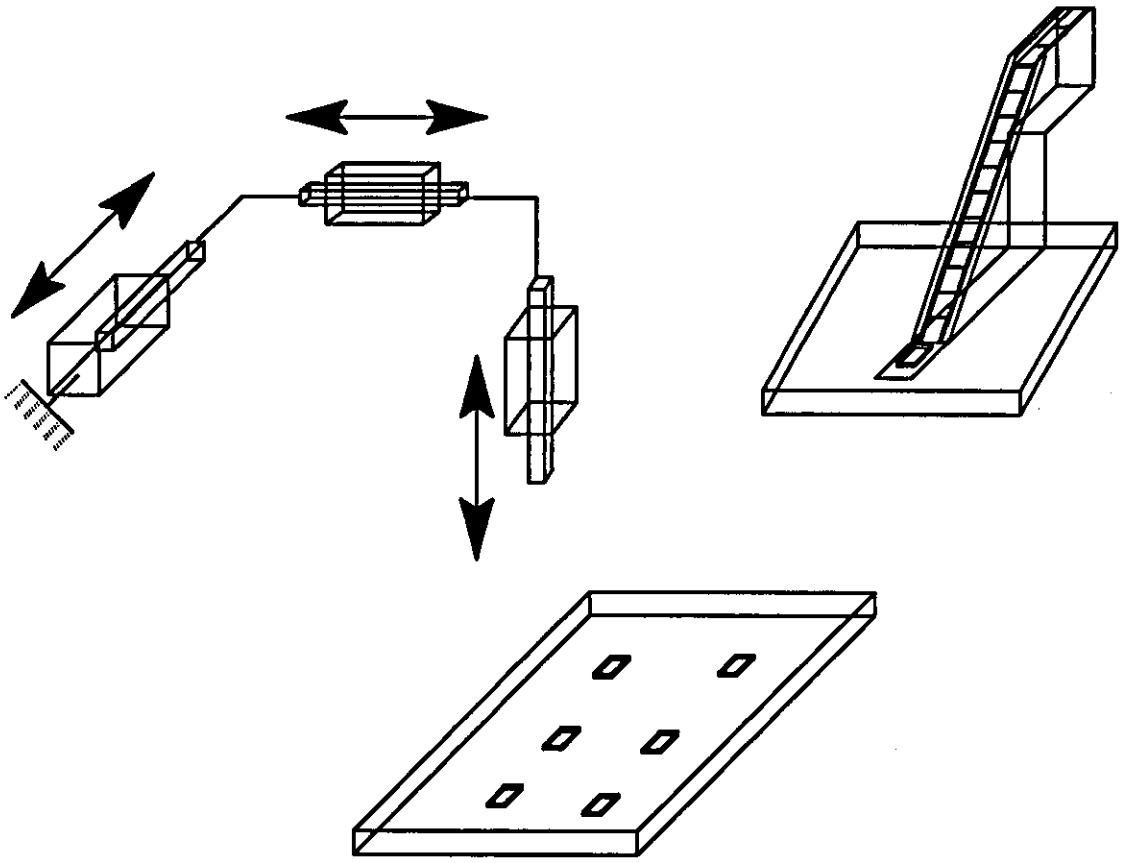


Figure 8.4 The loop application of an assembly operation in populating a PCB board

same level of intelligence as conventional computer languages in the sense of programming capability. However since the operated information is the feedback of simulated machine environment, these logical operations together with some conditional operation commands enables a user to have better awareness of a machine environment. Like the conditional operation command in conventional computer languages, **IF** (logical operation=true) **THEN** machine action, **WHILE** (logical operation=true) machine actions, and **FOR** machine actions **TILL** (logical operation=true) are used in machine programming at the device level. The **WHILE** and **FOR .. TILL** commands are a form of loop. The machine actions in these loops can be a series of sequential or parallel actions. For example, an articulated manipulator group is populating a PCB board. If the condition that a PCB board is at the "Ready" position is satisfied, then the manipulators can pick up several electronic chips and insert them into different positions one after another until the population is finished. The inserting positions can be defined as a variable, and the manipulator group uses the same assembly operation procedure (subroutine). A insertion procedure (see Figure 8.4) could be described as:

```
        WHILE (PCB_Ready = true AND population_not_finished)
        {
MOVE device_name, Pick_chip_position;
MOVE device_name, Insertion_approach_position;
MOVE device_name, Chip_target_position, velocity_profile_name;
MOVE device_name, Insertion_approach_position;
 }.
```

The loosely coupled motion default type is chosen for the above motions. A specified velocity profile is used in task-achieving operation of insertion, and the rest of the motions use the default trapezoid velocity profile. The condition of PCB_Ready is satisfied by delivering a PCB board at the assembly position where a sensor can then send a signal to validate the PCB_Ready condition. The delivery of a PCB board can be achieved by another device controlled by another device level control program. The assembly finishes when all chips are populated on a PCB board

8.6.4 Communication generic operation at the device level

Signalling operations describe the interaction of machine sensory devices with their environment, and they can be classified as external communication between machine controller and sensory devices. Due to the multi-device construction structure of a modular machine, there is a great need to communicate among these device's operations to achieve operation coordination. The interaction among these device's operations is defined as internal communication, which enables task planning coordination, information sharing, and the initialization and activation of other device operations. The efficient operation of a modular machine relies on the correct task operation planning and sub-task allocation to each device. Due to manufacturing requirements, there may be a sequence constraints, i.e. sub-task B can only be carried out after sub-task A finishes. The activation of such a successor sub-task can be realized through the internal communication between sub-tasks. Since this type of conditional variable can be defined as a global logical variable in the simulation phase, the communication was achieved by the command

SEND *destination_variable, message*;

where message can be a single "1" for true or "0" for false, or a long data message. In the situation of information sharing, the information in one sub-task, such as sensory values, velocity etc., can be used in another sub-task if it requires this information. The activation of another device is also achieved by the communication command

SEND *destination_variable, message.*

This then provides a tool for the machine controller to plan and control the machine devices in an intelligent manner. In real time control using the OS9 operating system for UMC, EVENT, PIPE and DATA MODULE can be used to pass information, and each has its own advantage. However, to simplify the communication method (mainly due to the adequate capability of the SEND command in the modular machine simulation), only the SEND command is created for the communication purpose among sub-tasks of a modular machine.

8.6.5 Application dependent task description at the device level

In assembly industries, a typical application dependent device is the gripper or hand which holds a component and assembles it in conjunction with transportation devices. The implementation of this simple gripper operation description for two jaw with symmetric gripped component at device level is described as

OPEN *gripper_name*; or CLOSE *gripper_name*;.

However, for those grippers which have more than two fingers, the task descriptions should be defined explicitly for hand operation. In the case of prismatic grasp of a symmetric object, the hand operation description should encompass the grasp type (prismatic) and gripped finger tip position relative to the central line of grasping the variable cylinder. The grasping event is described as:

CLOSE *hand_name, Prismatic, position*;

where the *position* is the relative position from the hand grasping to the central line of the variable cylinder (or the diameter of the cylinder). A similar event description is employed for the circular grasp

CLOSE *hand_name, Circular, position*;

where the *position* is defined relative to the grasping variable circle centre. The opposite operation of a hand is simpler than **CLOSE** and is described as **OPEN** *hand_name*. The above description is appropriate for symmetric component grasping, but the same description can also be extended to non-symmetric component grasping description by embedding some contact sensors on each finger. The approximate finger tip position definition can lead one of the fingers to touch the object and then the sensors will guide the other fingers to their contact (grasping) position.

The modelling of this type of complex grasping itself can be complicated in most modelling systems, let alone incorporating the hand in its whole environment. However, the modular approach allows the user to decompose the machine as well as the task into devices and sub-tasks, and hence each of the devices can be modelled individually and sub-task operations can also be tested beforehand. This method enables the user to model each device separately and assemble them into a large model. The other benefit of providing a tool to simulate hand operation is that it enables the dexterous hand researchers to simulate the conceptual hand performance before building any unnecessary prototype.

A conveyor is another type of application dependent device which has linear motion (usually driven by a rotation mechanism) with some variations in terms of motion type. A simple conveyor operation is described as

START *conveyor_name*;

An attribute is appended to allow the simulation of more complex conveyor operations. A conveyor operation depending on a sensor condition is described as:

START *conveyor_name, sensor_name, velocity*;

where *velocity* specifies the conveyor variable moving speed.

Modular machines can be used in various applications. The above approach can be employed to deal with many devices which arise in new applications. The method can be generalised as defining the functionality of a new device, decomposing the device into low level operations, studying the variations of the device function, deriving the task description command and attributes at the device level for the application and development of its control algorithm. Depending on the application dependent device functionality, the command to control the operation of the device can be in the abstracted form of the functional description. Again the format of the sub-task description has the form

ACTION *device_name, action_attribute*;

where **ACTION** can be instantiated by a specific device function and the attribute is the quantitative description of the action.This method is only a recommended procedure and the full implementation of other application dependent devices is left for future research and implementation with the advent of new devices.

8.7 Task level programming

Device level programming is characterised by the description of the primitives' separate operations and their coordination. A series of such primitives' operations can often result in the accomplishment of a sub-task. It is attractive to draw the programmer's attention to a level of programming that is one level above device programming [Rock 1988], i.e. to consider the machine operation description as the completion of some sub-tasks accomplished by a series of primitive operations at the device level. The description of a machine's operation is then at a high level of abstraction and the user can program a modular machine with ease. The premise of this level of programming is that the detailed parameters of each primitive's operation in a device are available to the task program instantiation and decomposition mechanism (PIDM). This method is based on the programmer defining some default parameter values for a device primitives at model creation stage and the PIDM having full access to the model information.

The task program in the instantiation and decomposition mechanism can also reason about some parameter values and primitive operation sequence based on the task level description and available primitive definitions (model knowledge) and some Macro operation sequences. All these can lead to easy programming with a higher level of abstraction, enhanced intelligibility of task program, more rapid program development and intensified program abstraction with automated assistance.

8.7.1 The abstraction of insertion assembly operation

In the simulation phase of a modular machine, the geometric information is fully represented in the model and is available to any internal mechanism although hidden from the user. This enables the program instantiation and decomposition mechanism to have access to geometric information of any machine device and component object. The PIDM then has a full spatial knowledge with which to reason about the distance each motion primitive moves. On the other hand, the primitive's operation type and the decision of selecting a particular primitive are determined by the high level task description and its incorporated operations. The operation sequence is usually dictated by production rules. For example, the operation sequence for the insertion of a peg into a hole may be typically expressed as

> MOVE *device_name, approaching_position1*;
> MOVE *device_name, component_gripping_position*;
> CLOSE *gripper_name*;
> MOVE *device_name, approaching_position1*;
> MOVE *device_name, approaching_position2*;
> MOVE *device_name, inserted_position*;
> OPEN *gripper_name*;
> MOVE *device_name, approaching_position2*;
> MOVE *device_name, original_position*;.

As clearly indicated in the above operation sequence (also see Figure 8.5), the assembly operation "Peg in a Hole" is composed of nine separate device operations and the sequence is decided by the correct assembly rules of such an assembly task. In order to describe such a task type at a higher level, the user has to specify the device or devices task type, the operation related object names and its destination to uniquely define a task. A proposed task description for the above assembly task is

> INSERT *object1_name, object2_name, device1_name, device2_name*;.

Since the modelling embeds all information about the machine model, the explicit specification of the object operated upon, operating device and destination object provides all related devices and objects for reasoning and decomposition. The task type INSERT then implies the operation content and sequence at the device level programming. The moving distance for a device is calculated as the relative distance between object1 and

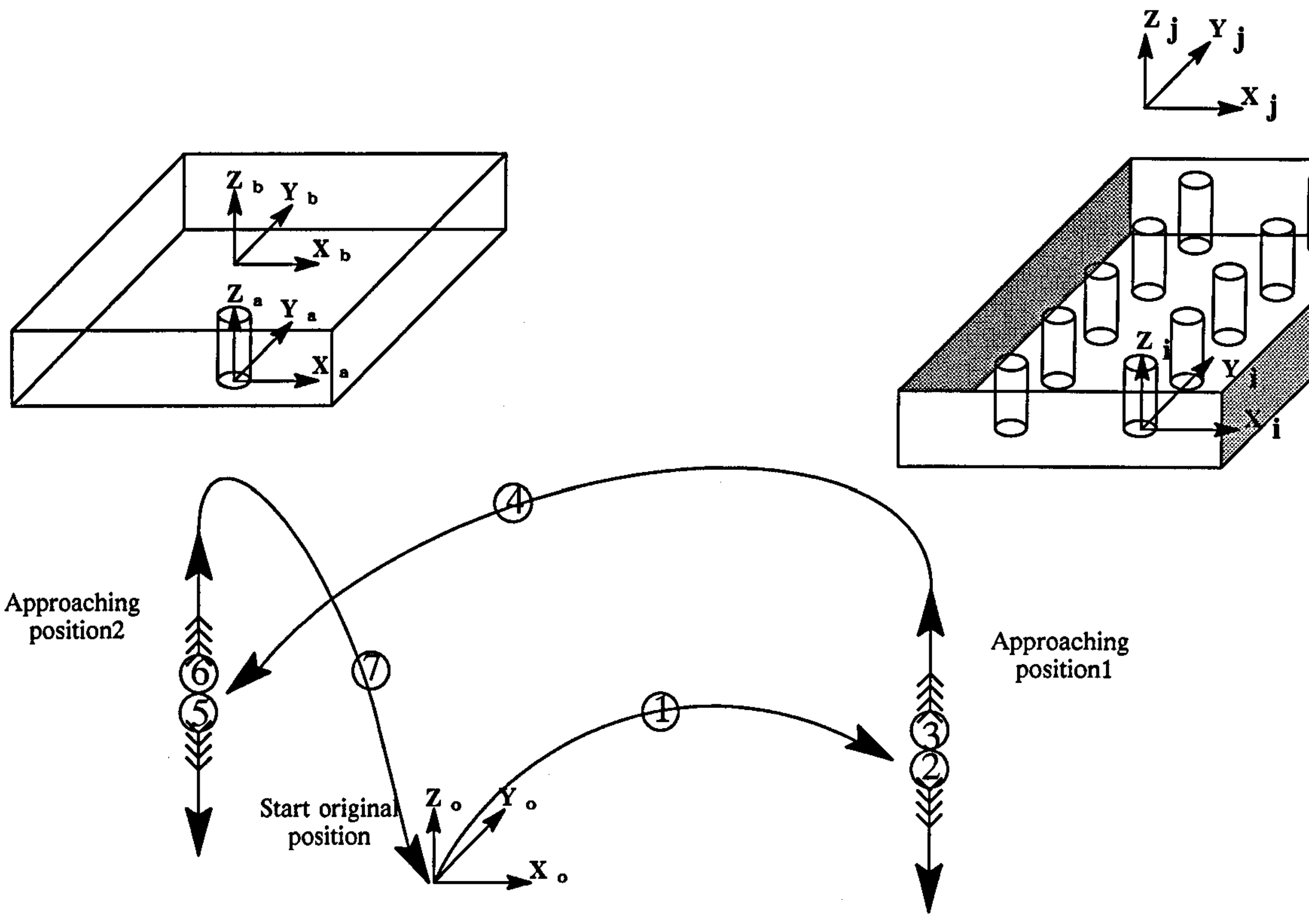


Figure 8.5 Typical insertion operation procedure

object2 along with the assembly rules fir INSERT task (in this case the operated component must be approached and inserted vertically with some safety considerations, and this determines the approach positions). Since there is usually more than one device of each type in a machine, it is necessary to specify the operation device to avoid ambiguity. Expert systems are expected to be employed for the above purpose. Some researchers [Rajan and Nof 1990] have realised the importance of efficient task allocation among several qualified candidate machines, and this issue should be considered at the task level description and task decomposition in modular machine programming. However, this is beyond the scope of the current research and is left as a consideration for further higher level programming (task level programming can then be further divided into low level - object and high level - task or objective programming [Rock 1988 and Van Aken et al. 1988]). However, the spatial relationship between two mating objects and the program instantiation and decomposition mechanism are considered in next section.

8.7.2 The spatial relationship of two assembled mating objects

For objects with a simple geometric shape, such as cuboid and cylinder, ten spatial relationships are possible (see Figure 8.6) excluding the impossible assembly operation of "bottom into" and "beneath" (these two however can be substituted by "top into" and "on" if the assembled sub-assembly is turned upside down). The above ten possibilities can be further reduced if a restriction is imposed such that a mating direction always has to be the Z direction of the local coordinate frame. The system could then automatically work out the "insertion" or "placing" direction according to the convention. However, this can restrict the user model object definition and cause inconvenience for modelling, and therefore all ten possible relationships are accommodated in the object level description by requiring the

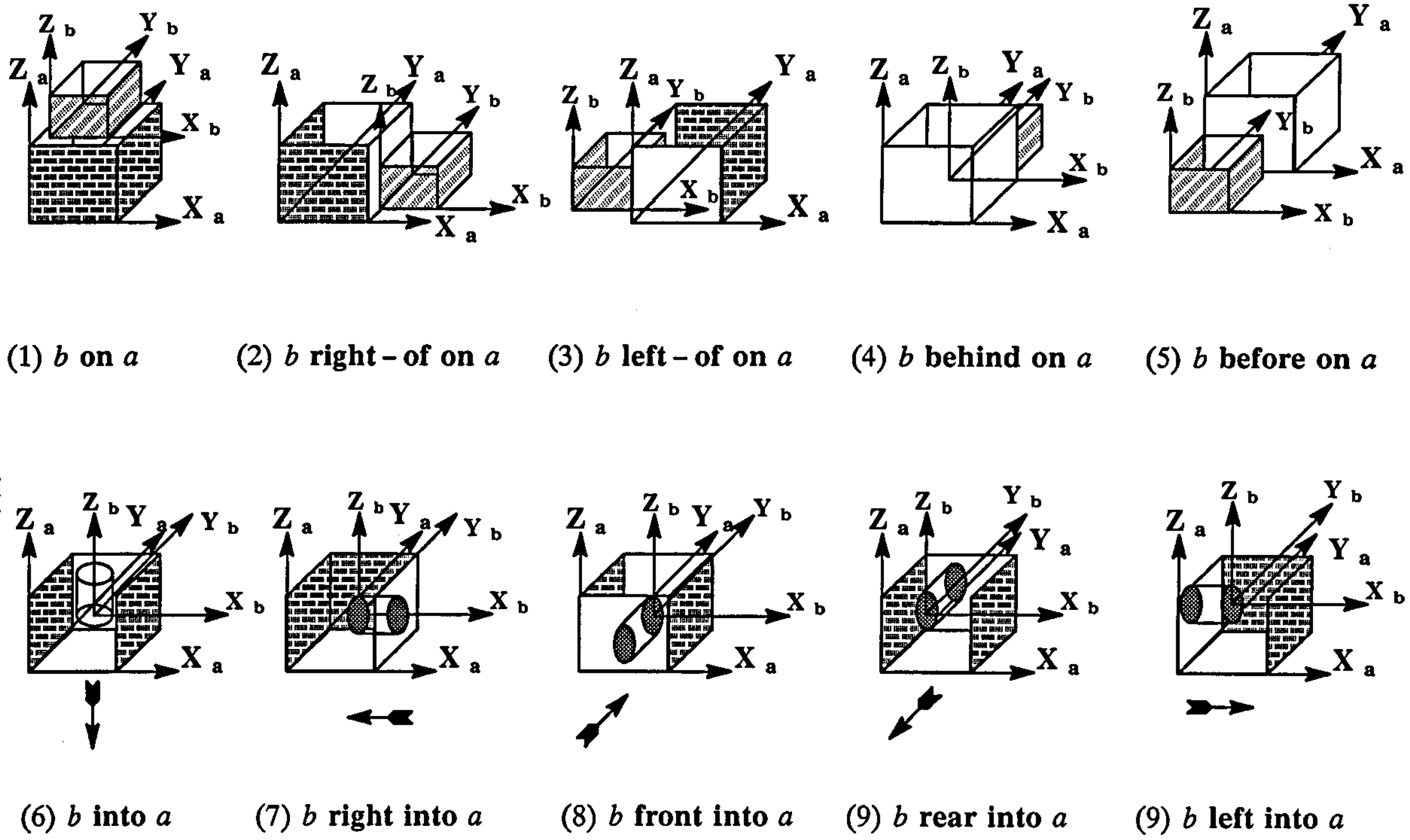


Figure 8.6 Ten possible relationships of insertion and placing operation between two objects

user to explicitly specify the relationship between two assembly objects. The other options to "into" description of "insert" command mentioned early are the rest of relationships listed in Figure 8.6. All these relationships can be embedded as an attribute in any part mating related task description.

8.7.3 The task programming instantiation and decomposition mechanism

Knowledge about the two mating components gives more information for the system to calculate the moving distance. It is easy to calculate the distance between the two frames related to the component local frame. Through a search of the modelling data structures of two component geometry, the boundary, shape and dimensions of each component can be found, and the offset between the coordinate frame and the outline of each component can then be calculated.

Since a set of conventions are used to specify the position and orientation of a local coordinate frame relative to the geometric representation in GRASP [Grasp 1989] and other equivalent modelling systems, the calculation of a coordinate frame offset can be used to obtain the moving distance due to the component dimension variation (see Figure 8.7). The spatial relationship between two mating components is also a very useful in calculation of the moving distance. The distance due to the assembly direction can be based on the spatial information specified in the object level task programming. With the above full knowledge and the names of the devices involved, a lower level (device level) of machine task execution programming can be generated with reference to the task description and the assembly rule.

The object level programming described in this chapter partially achieves the objective of

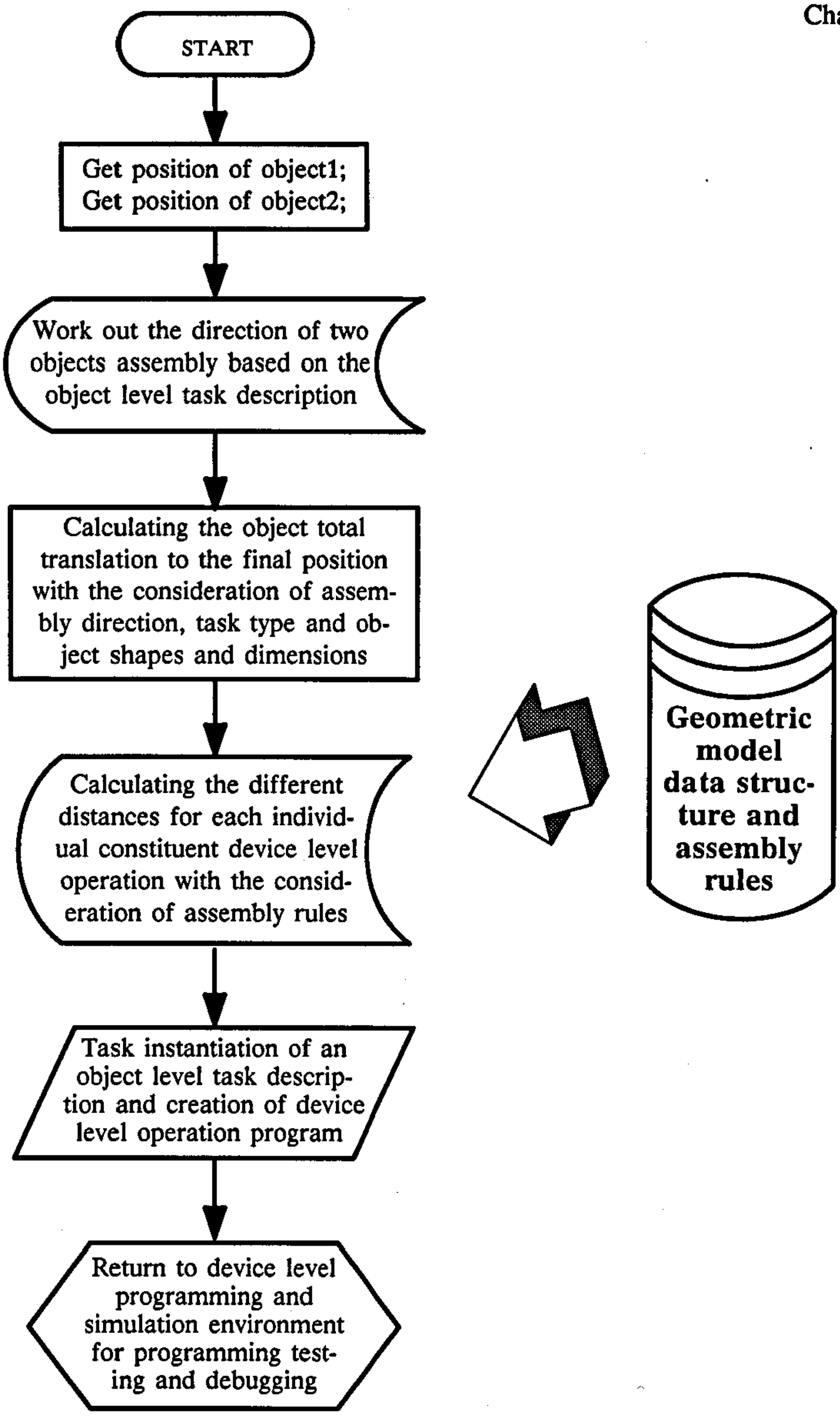

START
Get position of object1;
Get position of object2;
Work out the direction of two objects assembly based on the object level task description
Calculating the object total translation to the final position with the consideration of assembly direction, task type and object shapes and dimensions
Calculating the different distances for each individual constituent device level operation with the consideration of assembly rules
Task instantiation of an object level task description and creation of device level operation program
Return to device level programming and simulation environment for programming testing and debugging
Geometric model data structure and assembly rules

Figure 8.7 The schematic of task program instantiation and decomposition mechanism

intelligent programming. By utilizing model information, the moving distance of a device can be determined automatically. Since the derivation of device level operation type and sequence are based on the substitution of a set of fixed operations, this approach cannot be classified as real intelligence. A genuine intelligent programming approach needs to be dealt with at an even higher level - assembly task programming, and this approach requires more background information and high level intelligent reasoning. However, this area remains untouched and is left for future research.

This chapter has illustrated some implementation of modular machine programming which is essential part of the modular machine design and simulation environment. A three level modular machine programming environment is described and the implementation of the first two levels have been discussed with particular reference to assembly operations. It is suggested that artificial intelligence is desired for generating high level task programming.

Chapter 9 Common data format for the integration of simulation and physical machines

9.1 Introduction

Currently most machine design and simulation software works in isolation within a computer integrated manufacturing environment [Durr 1989, Kemper 1987]. The reason for this is that almost every system relies on its own specially designed and customized data structure. The lack of a standard data structure has greatly impeded easy access to the data structure of different systems, flexible manipulation at the different user levels and efficient information utilization. Due to the existence of various data structures at different levels of the entire engineering life-cycle and also because of different requirements at each application level, it is difficult to define a new standard data structure to be a popularly used and accepted standard within engineering applications. However, Maier et al. [1985] proposed a new data model and Zdonik et al. [1986] described a methodology for complying with an object-oriented programming environment in a database.

With increasing appreciation of the importance of a standard for product model exchange, many researchers have devoted their effort to accelerate the development and implementation of an international standard STEP (STandard for the Exchange of Product Model Data). STEP is targeted to provide a complete, unambiguous, computer-sensible description of a product throughout the life cycle of the product. The data elements required to support a product through its life cycle include not only the geometry but other attributes and features that completely define a product [STEP 1991, Vergeest 1991]. Although the STEP standard has attracted many researchers attention particularly those whose main interests involve geometric modelling and data exchange, only philosophies embodied in the STEP standard was used to achieve system integration. The reasons for adopting such an approach is that the STEP standard is so complicated and comprehensive that it has redundant capabilities for the modelling of modular machines. In addition at the time of this implementation there was no complier commercially available and it is also difficult to write such a compiler. Based on these reasons a local common data format approach was used by the author and other researchers [Goh 1991]. In this chapter, such an approach with reference to the STEP standard methodology is proposed to integrate the various systems at different levels. This common data format can serve as the common interface between the existing machine design and simulation system and the physical machine control as well as other computer aided machine design systems. The common data format is a type of neutral data format which tries to encapsulate the contents of different requirements from each application level and provide a comprehensive and consistent data information for machine design, simulation and physical machine control.

9.2 The demands of common data format

9.2.1 The requirement from the simulation of physical machines

For most industrial automation machines, there is a wide range of control methods used in applications and various control parameter data formats exist. However there is much commonality in manufacturing methodologies, and it is possible to identify generic control task and control parameter data among these different control approaches. It is of great importance to generate a common data content and format from physical machines so as to facilitate the creation of control algorithms and the specifications for the simulation phase. A common data format can facilitate the standardization of design and manufacturing of the industrial controllers and dramatically reduce the large number of pre-processors and post-processors required by both the physical and simulation controllers.

At the simulation phase of machine design, a set of control and drive parameters are required in order to manipulate the graphical representation of a manipulator. Direct access to this control data from the simulation environment can save effort in redesigning the control parameters, increase the possibility of the data being directly used at the physical machine control phase and improve the close connection between a physical machine control and its simulation (device manipulation). One of the major data contents in simulation is kinematic modelling data which is defined as the control data in physical machine control. Use of a data format that is common to the machine and the simulation improves the realism of the simulation. This is because the closer the relationship between the simulation of a machine and the physical machine, the more accurate and reliable the simulation result is.

9.2.2 The requirements from physical machines

With the requirement for quick product changeover and economical production, the manufacturing complexity from machine functionality to task planning increases considerably. The importance of a correct evaluation of a manufacturing machine's functionality and its performance beforehand is so prominent that the simulation of a machine's task execution becomes a key component to assist the management to make responsive decisions. It is easy to intelligently achieve a complex task in the simulation environment based on the large number of calculations and assumptions that a simulated machine model always perform as expected and there are no time constraints during the simulation of a machine. However, due to the real-time constraints and the low level focus of machine control, it is unreasonable to expect a physical machine to achieve the same level of performance evaluation as a simulated machine model in terms of planning, calculation etc. Therefore there is a need for a physical machine to share the simulation results in its physical task execution. A consistent data content and format can undoubtedly improve the easy utilization of such simulation results.

The high level task planning of a machine can be accomplished easily, efficiently and cheaply at the simulation phase. There is usually no equivalent level of planning in physical machine programming. The execution of a high level planning task at the simulation phase requires a decomposition of the task into low level (physical machine programming level) sub-tasks. The common data format at the level of both simulation and physical machine control provides a means to interface the high level task planning and physical machine level control.

A common data format can facilitate the information sharing between simulation and physical machine. The simulation results can be used in physical machine control. It can also make it easier to interface either the simulation or physical machine with other machine design and evaluation systems. If there are m CAD systems for machine design and there are n physical machines, then $m \times n$ processors will need to be written in order to interface all these m systems with the n machines. However, it is only necessary to write $m + n$ pre-processors and post-processors to interface them if the common data format approach is adopted in this integration design issue. In this research, the interface is implemented between a mechanism design software package known as CAMLINKS [1991] in Machine Design and Control and the modular machine simulation environment.

CAMLINKS is designed for mechanism and machine design and analysis, and it provides very useful data on cam motions which can possibly be used for software cam control.

9.3 The common data format approach

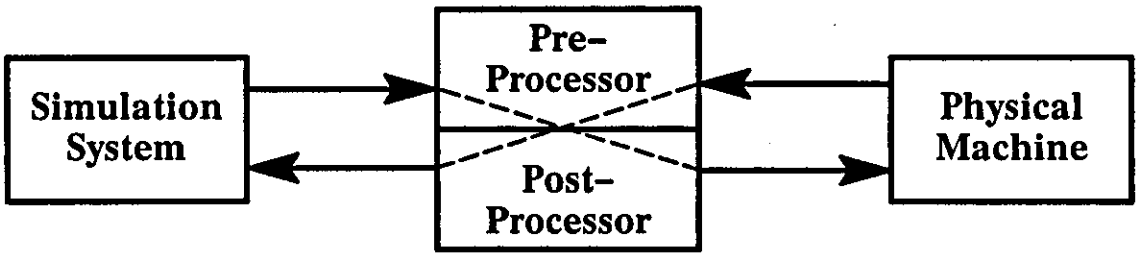
9.3.1 Centralised common data format

There are two typical approaches to achieve the integration of machine graphical simulation and physical machine control. The first is for a machine simulation system to use its own data formats both in terms of geometric and machine simulation information (a CAD system data format is probably used). Alternatively, the physical machine also works on its own data format (possibly a specialised data format), but integration is achieved by a direct processor between the simulation system and the physical machine. The second approach is the same as the first in the sense that both the simulation and the physical machine work on their own data formats, but with a centralised common data format introduced, each side must have a processor in order to establish an interface with the common data format directly. Figure 9.1(a) and Figure 9.1(b) describe the above two approaches respectively. The second approach was adopted in this study and a prototype common data format was designed and partially implemented (described in the section 9.4).

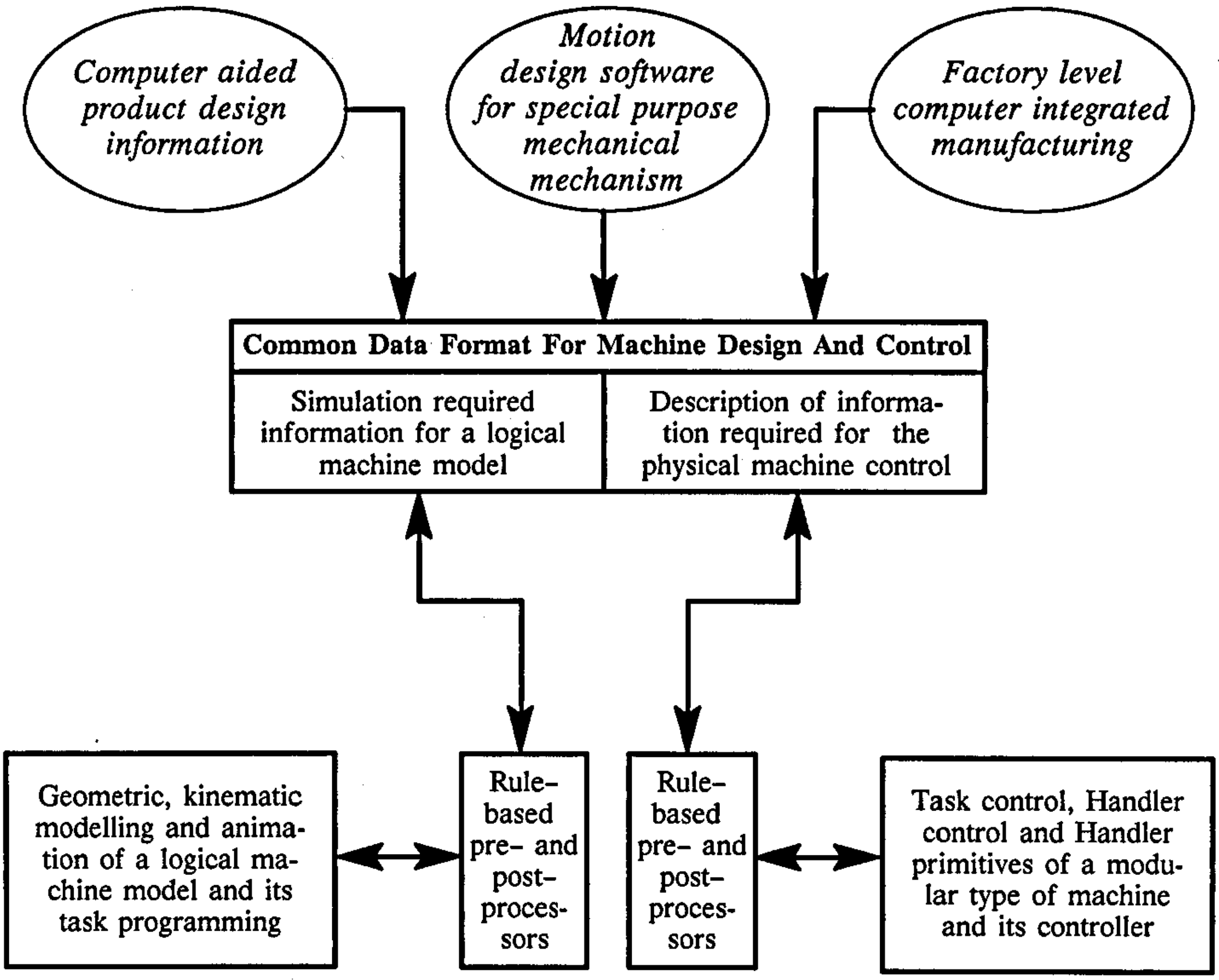
9.3.2 The data analysis of a simulation system and physical machines

From the point of view of the simulation phase, the following data types should be included in the common data format:

(1) Geometrical description data of a simulation machine model which defines the shape, dimension and layout of each machine component within the simulation



(a)


Computer aided product design information
Motion design software for special purpose mechanical mechanism
Factory level computer integrated manufacturing
Common Data Format For Machine Design And Control
Simulation required information for a logical machine model
Description of information required for the physical machine control
Geometric, kinematic modelling and animation of a logical machine model and its task programming
Rule-based pre- and post-processors
Rule-based pre- and post-processors
Task control, Handler control and Handler primitives of a modular type of machine and its controller

(b)

Figure 9.1 Two approaches towards the integration of the simulation system and physical machines and other CAD systems. (a) direct integration; (b) common data format approach.

environment. These data provide essential information about the geometry of a machine and its spatial relationships;

(2) Kinematic specification data of a simulated machine, which defines the capability and constraints of a machine's motion;

(3) Machine task definition of a simulated model, which specifies sequence of machine's task execution, the task type and the target position (task attributes) of each sub-task. The task definition can be made at three different levels for the convenience of a user as described in Chapter 8;

(4) Sensory device and other special mechanism functionality described in a simulation model. Since the device does not exist physically, functional performance is purely dependent on the definition of each device's function and its computational execution of each function.

The physical machine control shares two of these types of data (kinematic specification and task definition) with the simulation. Since sensory and other special devices exist physically in a machine, there is no need to describe their functionalities in its physical machine model. Also since a real machine is already constructed physically, there is generally no need to describe its geometric parameters, although in some advanced machine control this information could assist intelligent machine control and decision making. The following additional data types are required to accommodate the real-time physical machine control issues.

(1) Machine dynamic feature descriptions which define the dynamic performance of each machine's manipulators, such as damping, inertia, gain, moment tensor etc. The physical machine's real-time response is highly dependent upon its dynamic

characteristics. The optimal selection of these parameter values can produce high quality control performance.

(2) I/O checking description of each port or handler. Since the physical communication and task execution devices are not perfectly reliable, frequent checking of each port to ensure a correct status is of prime importance to ensure the correct execution of the whole task.

All these six types of data are separately defined in the simulation and physical machine domains in most systems, but a unified description is required for true integration, which can be described by using the EXPRESS language.

9.3.3 Common data format definition method

It is not necessary to include every type of data at each level of machine evaluation and application as they have differing requirements. However, as data types overlap between different systems in a machine application environment, it is possible to abstract a common data type from more than two systems into part of the common data format. The methodology employed is based on data abstraction and the collection of these local common data types into a common data format. The abstraction is carried out at different application levels and the data collected are arranged in the same hierarchical structure as with their physical machines or simulation model. Therefore the common data format possesses the characteristic of hierarchy, modularity and multi-usage.

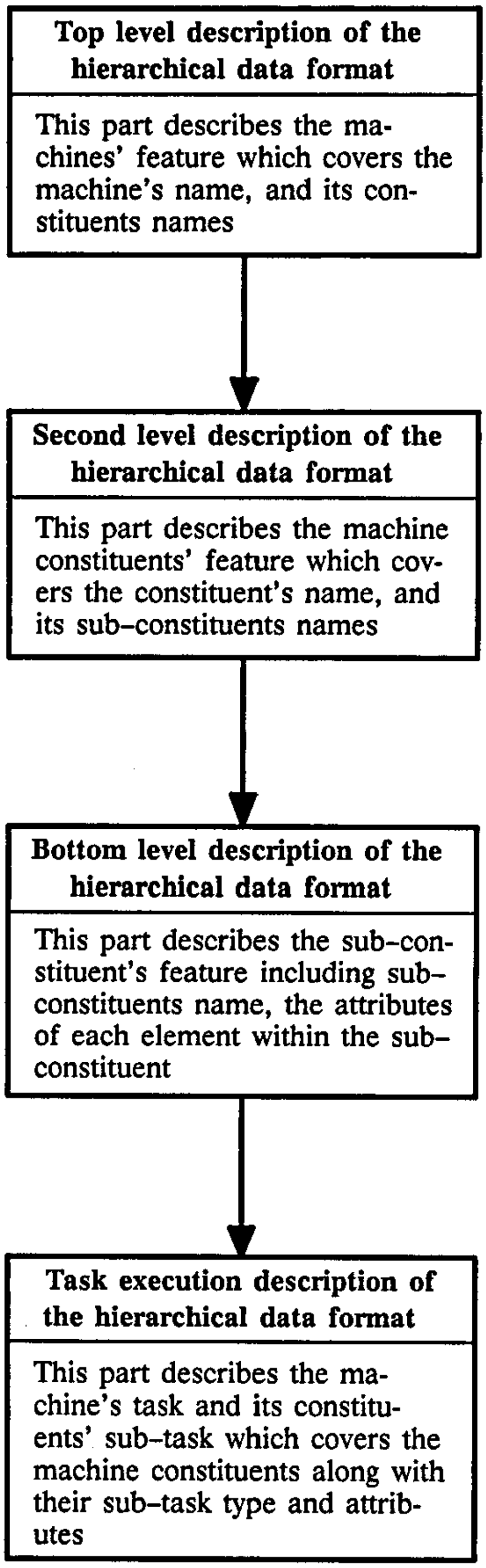
Based on the above methodology and the current implementation environment of the Universal Machine Control architecture, the UMC simulation environment and a specialised CAD based motion design software (CAMLINKS), the first three types of data have been formed into the current common data format. The remaining three types are treated as local data requirements of the simulation and physical machine control but could eventually be included in the common data format for an even more advanced emulation environment.

The representation of each object, device, primitive, position, path etc. is identified by a unique name. Therefore the exclusive name for each entity in the machine application environment (no matter whether it be the simulation environment or physical machine control) is of extreme importance. The syntax for these common data format definitions is based on Entity_*type* Entity_*name*, *attributes*, as illustrated in the previous chapter. An overall common data format description is illustrated in Figure 9.2.

Since a hierarchical structure is embedded in the data format, a top-down machine definition approach is naturally used to facilitate the data definition. Because there is usually more than one element as the higher level descendants or its attributes specification, the common data format possesses the recursive definition feature at the same level definition of a machine element or its attribute data,. All lower level elements are defined sequentially until all elements belonging to the same owner are defined. The geometric and kinematic data are described at the bottom level for different attributes. Task definition data is at the same level as the machine description level, (the top level). The task data output format can also form a hierarchy for different machine constituents - devices. At the top level of the task data a higher level task description is used to generalise the task being executed. The second level task description is sequential task execution description although there can be parallel task execution.

```
Entity_type Entity_name{
    Constituent_element's_name1,
    Constituent_element's_name2,
    Constituent_element's_name3,
                |   |
                |   |
                |   |
    Constituent_element's_last_name;
};
    Constituent_element's_name1[
        Sub-constituent_element's_name1,
        Sub-constituent_element's_name2,
                |   |
                |   |
                |   |
        Sub-constituent_element's_last_name;
    ];
        Sub-constituent_element's_name1(
            Primitive_geometry_information(
                shape_type,
                X,Y,Z dimensions),
            Primitive_location_at_workcell(
                Translation(X,Y,Z),
                Rotation(X,Y,Z)),
            Primitive_constraints(
                axis_home_position,
                park_position,
                maximum_position,
                minimum_position,
                maximum_velocity,
                maximum_acceleration),
);              |   |
                |   |
                |   |
Task task_name{
    Sub-task_name1,
    Sub-task_name2,
    Sub-task_name3,

};
    Sub-task_name1(
        Primitive_manipulation_command1,
        ,       |   |
                |   |
    Primitive_manipulation_last_command
    );          |   |
                |   |
;
```


Top level description of the hierarchical data format

This part describes the machines' feature which covers the machine's name, and its constituents names

Second level description of the hierarchical data format

This part describes the machine constituents' feature which covers the constituent's name, and its sub-constituents names

Bottom level description of the hierarchical data format

This part describes the sub-constituent's feature including sub-constituents name, the attributes of each element within the sub-constituent

Task execution description of the hierarchical data format

This part describes the machine's task and its constituents' sub-task which covers the machine constituents along with their sub-task type and attributes

Figure 9.2 The common data format for the machine geometric, kinematic and task description

9.4 Processors implemented between the common data format and the simulation environment

9.4.1 Post-Processor from the common data format to the simulation

The purpose of establishing a post-processor from the common data format to a simulation environment is to interpret the common data format into the ring and tree structure used in the modular machine simulation environment. All data types embedded in a common data file are not needed to simulate a machine model. The main objective is to process the geometric and kinematic part of the common data file so as to create a data structure and corresponding data blocks (as described in the previous chapters). The first key word expected is "Machine" followed by a name that is stored in a data block at the highest level of the machine modelling data structure hierarchy. The number of machine constituents and their head block pointers are also stored in this data block on subsequent scanning of the file. The machine definition is introduced by a bracket which denotes the beginning of the constituent parts of the machine, identified by their names and separated by comma. The machine definition is terminated by a concluding bracket followed by a semi-colon to indicate the completion of one entity at this level (in this case the machine entity). At this point, the definition usually drops one level of the hierarchy to specify the machine constituents' attributes. The same starting symbol "{"and concluding symbols "}" are used throughout the common data file to clarify the hierarchical structure for the post-processor.

At the second level of the data structure hierarchy, the corresponding data blocks are created for each of these machine constituent parts based on their type. Each of these data blocks belongs to the machine data block and remains at the same level to form a machine constituents data block ring in the simulation environment. The next level of the hierarchy defines sub-constituent parts of a machine constituent part. At this level, the detailed geometry, location information and kinematic constraints of the sub-constituent part are specified. At this level the processor generates the standard data tree and ring structure for the motion axis in the simulation. The successive creation of those sub-constituent parts forms another ring at this level before the concluding symbols. The creation of these data blocks for sub-constituents then collectively produces a structured data representation which is ready for graphical manipulation and displaying.

The concluding symbols of one entity at a particular level, followed by a key word - ***Task*** indicates the completion of geometric and kinematic description of a machine model and the start of the machine's task description. A similar hierarchical structure was also employed in the task description and therefore a similar hierarchical structure is created for the task description. The schematic of the complete post-processor is illustrated in Figure 9.3.

9.4.2 Pre-Processor from the simulation environment to a common data format file

This pre-processor abstracts the common data format related information from the tree and ring data block structure in the simulation environment. The entity name (or machine's name), its constituents and the sub-constituents' attributes are obtained from the data structure. With this complete set of information, the simulation data structure can be down loaded into a file complying with the common data format syntax. The data abstraction is based on a data block search throughout the whole simulation environment. The search is started at the model owner - ***Workplace*** (the default highest level of the GRASP hierarchy). The Principle Ring Pointer (PRP) is extensively used to find the next object in the ring,

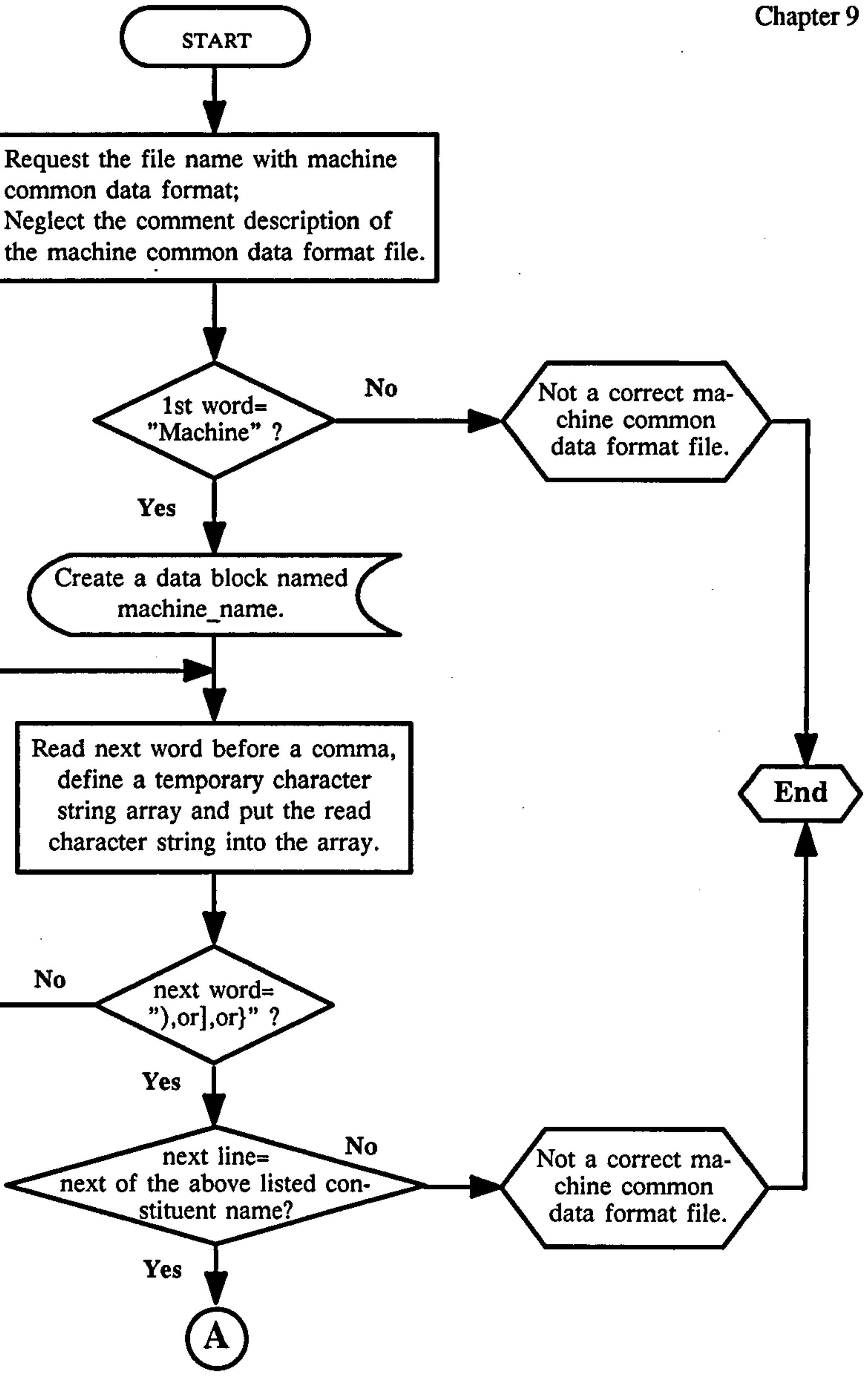

START
Request the file name with machine common data format;
Neglect the comment description of the machine common data format file.
1st word= "Machine" ?
No
Not a correct machine common data format file.
Yes
Create a data block named machine_name.
Read next word before a comma, define a temporary character string array and put the read character string into the array.
End
No
next word= "),or],or}" ?
Yes
next line= next of the above listed constituent name?
No
Not a correct machine common data format file.
Yes
A

Figure 9.3 The flowchart of post-processing of a machine data file with common data format(to be continued)

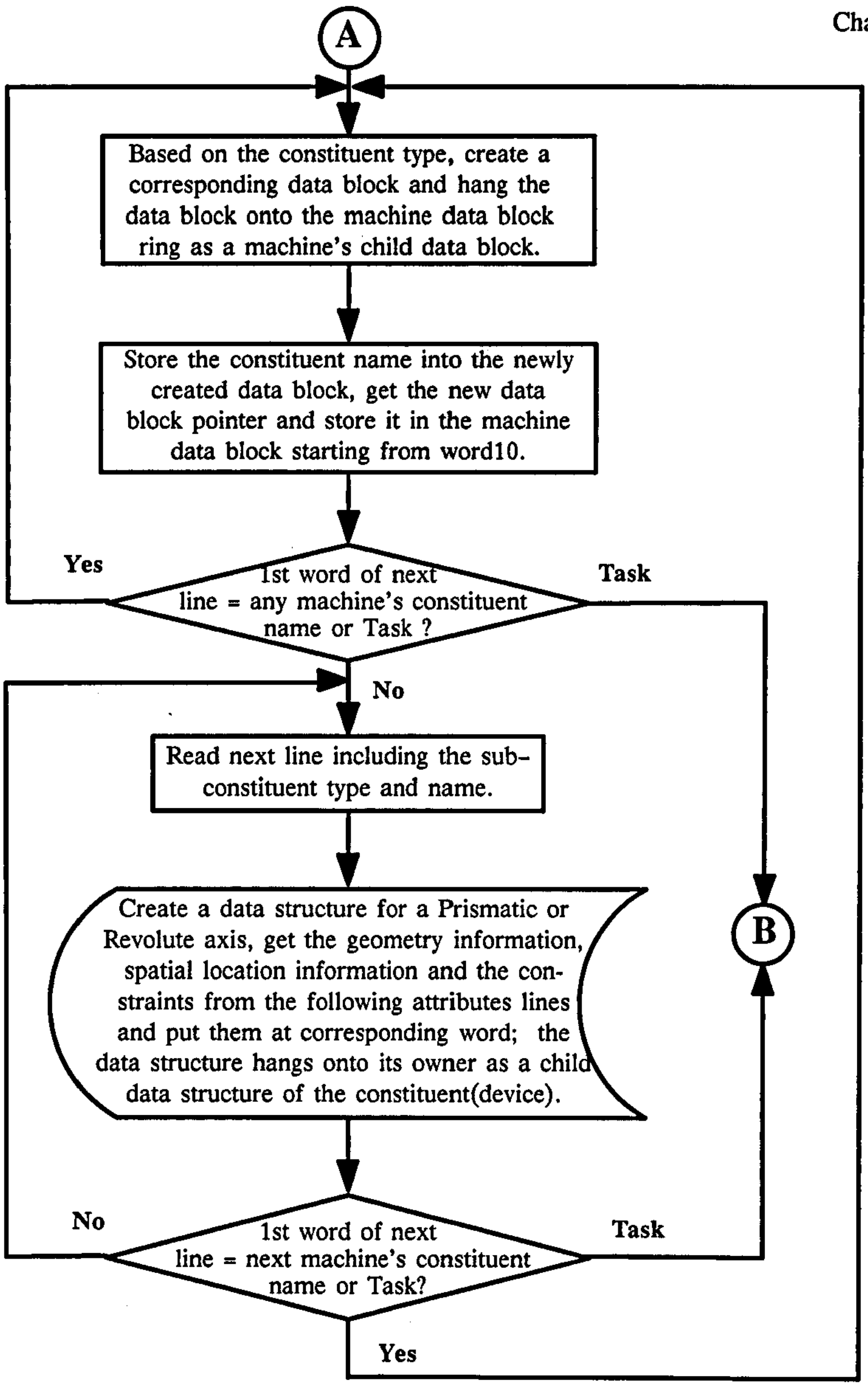

A
Based on the constituent type, create a corresponding data block and hang the data block onto the machine data block ring as a machine's child data block.
Store the constituent name into the newly created data block, get the new data block pointer and store it in the machine data block starting from word10.
Yes
1st word of next line = any machine's constituent name or Task ?
Task
No
Read next line including the sub-constituent type and name.
Create a data structure for a Prismatic or Revolute axis, get the geometry information, spatial location information and the constraints from the following attributes lines and put them at corresponding word; the data structure hangs onto its owner as a child data structure of the constituent(device).
B
No
1st word of next line = next machine's constituent name or Task?
Task
Yes

Figure 9.3 The flowchart of post-processing of a machine data file with common data format(to be continued)

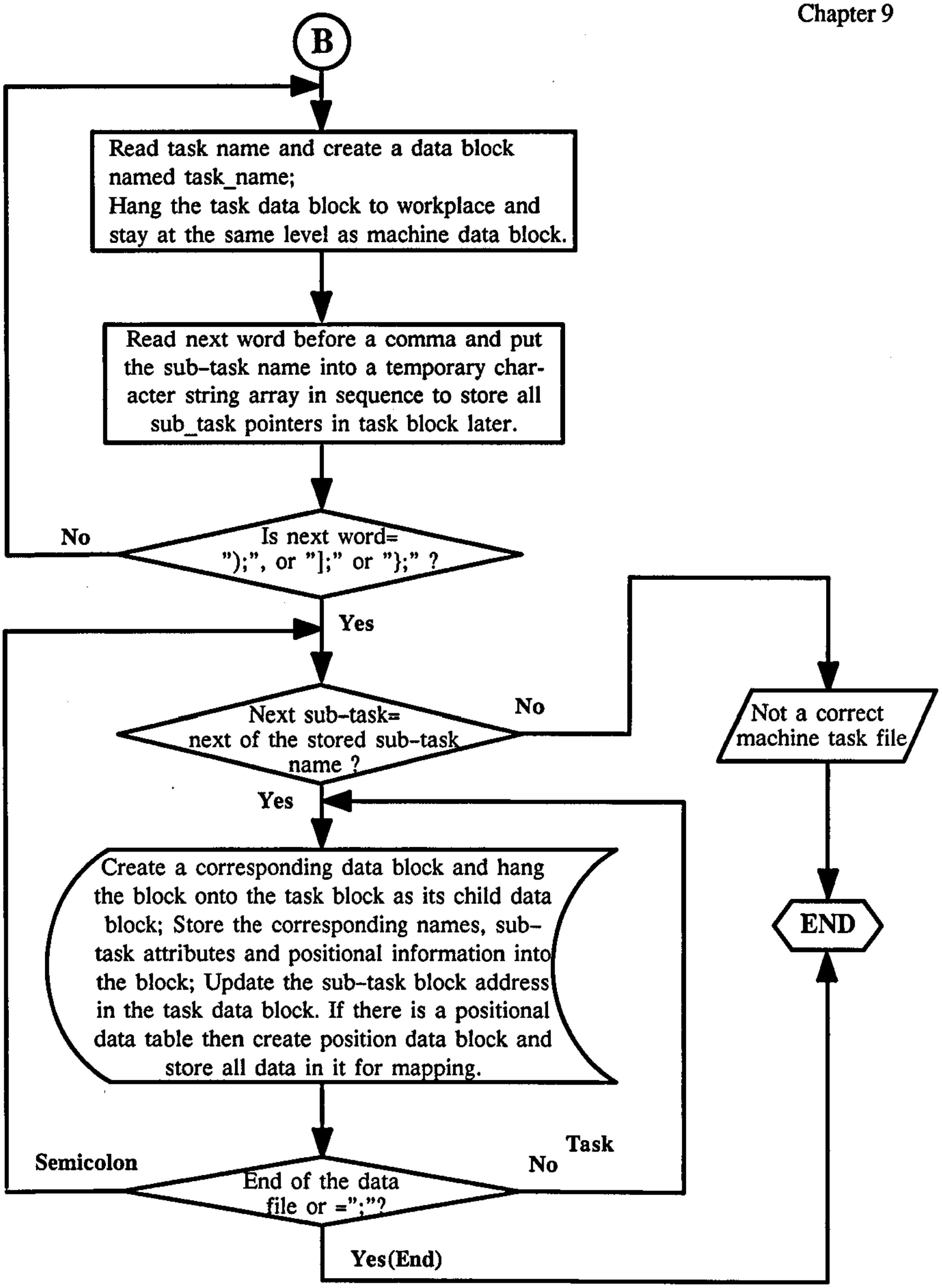

B
Read task name and create a data block named task_name;
Hang the task data block to workplace and stay at the same level as machine data block.
Read next word before a comma and put the sub-task name into a temporary character string array in sequence to store all sub_task pointers in task block later.
Is next word= ");", or "];" or "};" ?
No
Yes
Next sub-task= next of the stored sub-task name ?
No
Yes
Not a correct machine task file
END
Create a corresponding data block and hang the block onto the task block as its child data block; Store the corresponding names, sub-task attributes and positional information into the block; Update the sub-task block address in the task data block. If there is a positional data table then create position data block and store all data in it for mapping.
Semicolon
Task
No
End of the data file or =";"?
Yes(End)

Figure 9.3 The flowchart of post-processing of a machine data file with common data format

while the Entity Ring Start Pointers (ERSP) are used to identify the lower level descendants (see Figure 4.7 and Figure 5.5). The entity's name and its pointer are treated as the main connection to find the correct data block and this logical connection is maintained throughout the simulation data structure.

The searching pointer traverses the data structure from the top level of the hierarchy to the bottom of one of its constituents (device). Once it finishes the searching of one branch of the tree, it traverses to the next constituents of the machine and searches through its descendants again. Figure 9.4 describes the top-down searching approach. The same procedure is used for the search of different devices with the accommodation of some diverse properties for different device types. The end of each low level or next data structure group searching means the completion of each common data block format, i.e, the output of beginning from the data output block in the form of starting symbol "{" or "[" or "(" to the concluding symbols "};" or "];" or ");". Looping techniques were used to improve the program quality and it proves to be efficient in data searching. The program was coded in Fortran and the common data format file is created as an ASCII file for easy transfer. The whole procedure is illustrated in Figure 9.5.

9.5 The interface between the simulation environment and other CAD based systems

9.5.1 Introduction

The simulation environment has been designed to provide machine design and evaluation tools which are user friendly and versatile in functionality. The motion primitive library covers a wide range of motion elements and appropriate selection and combination enables

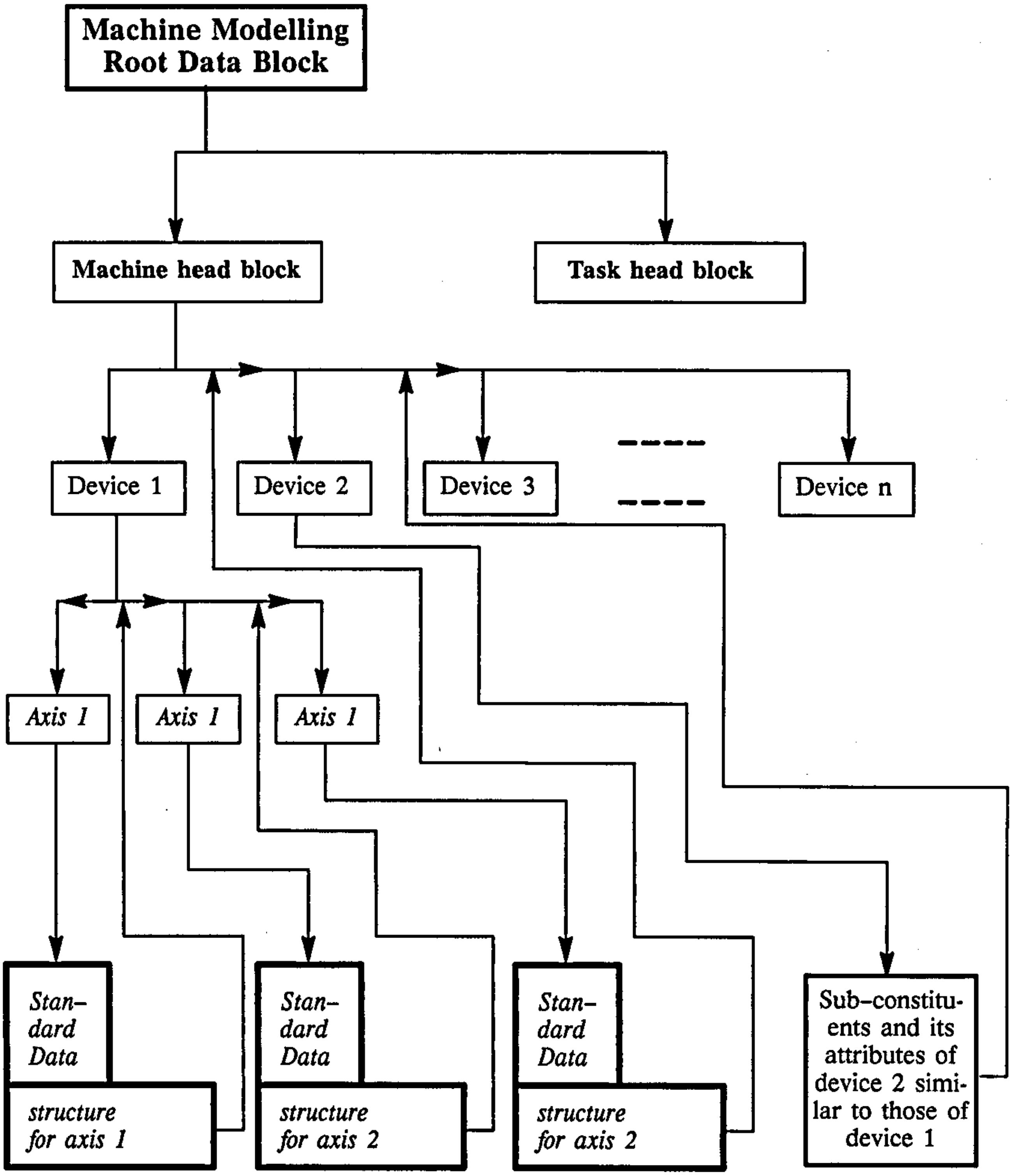


Figure 9.4 Top-down approach in data searching

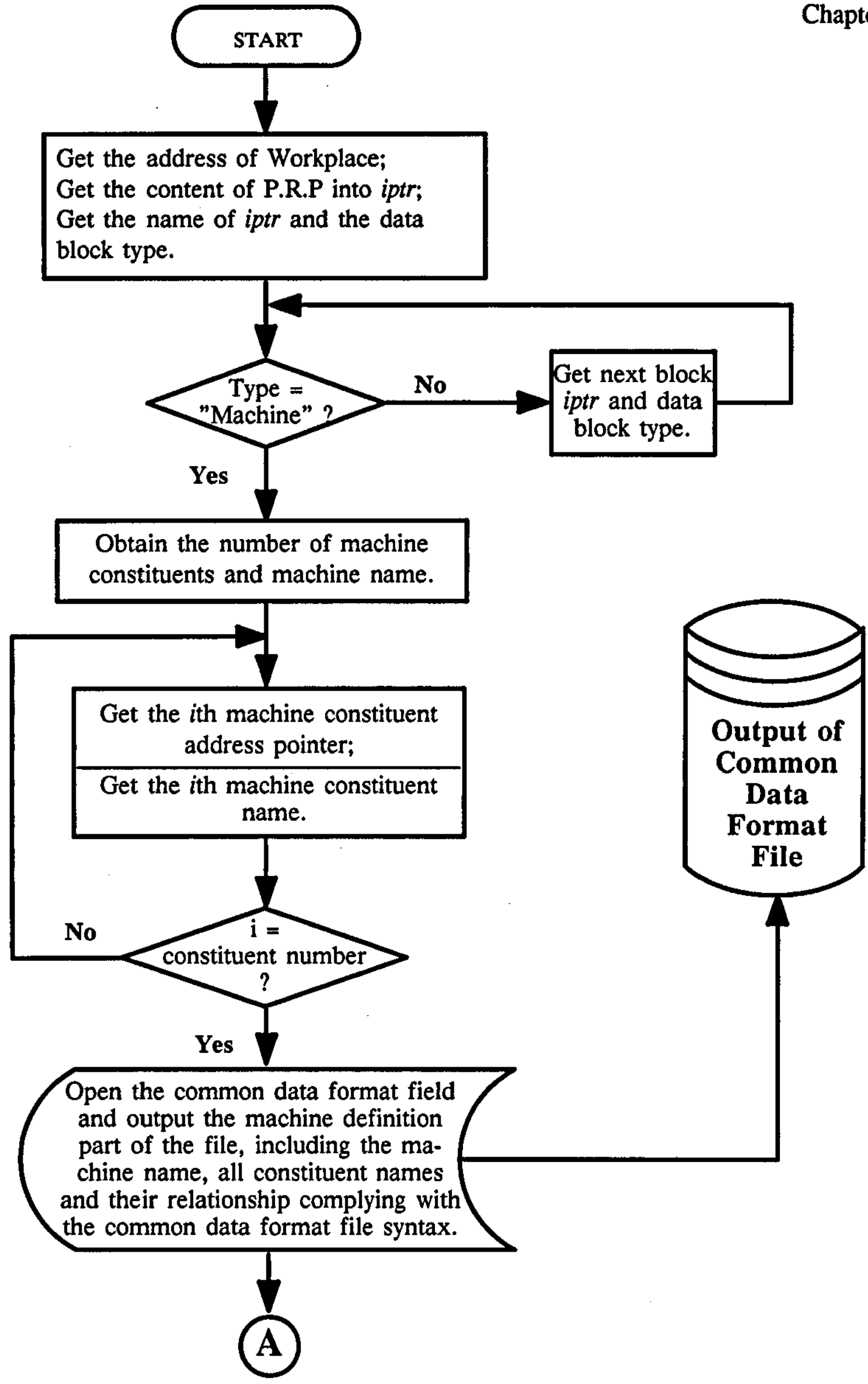

START
Get the address of Workplace;
Get the content of P.R.P into *iptr*;
Get the name of *iptr* and the data block type.
Type = "Machine" ?
No
Get next block *iptr* and data block type.
Yes
Obtain the number of machine constituents and machine name.
Get the *i*th machine constituent address pointer;
Get the *i*th machine constituent name.
i = constituent number ?
No
Yes
Open the common data format field and output the machine definition part of the file, including the machine name, all constituent names and their relationship complying with the common data format file syntax.
Output of Common Data Format File
A

Figure 9.5 The flowchart of pre-processing of a machine data structure into its common data format(to be continued)

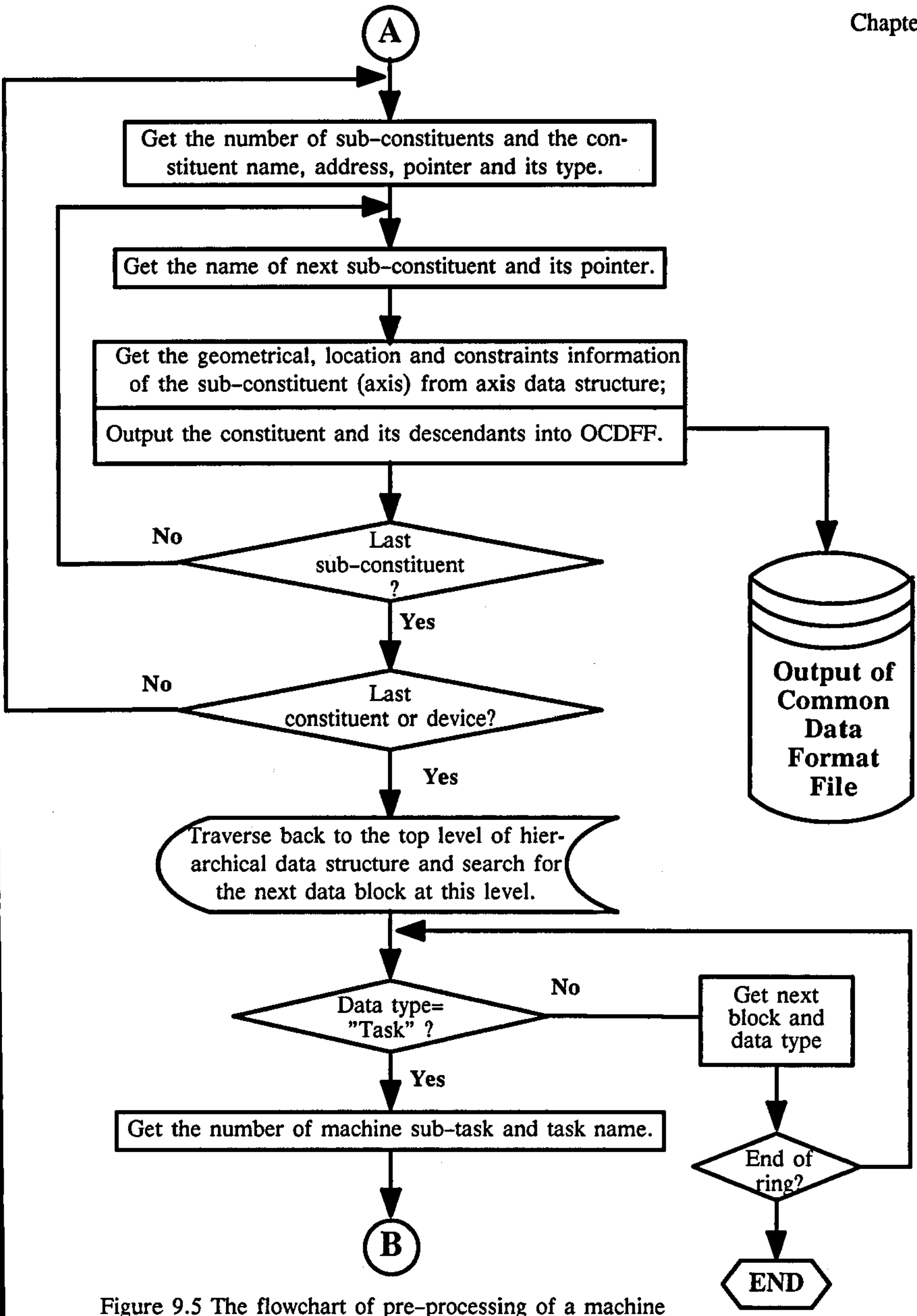

A
Get the number of sub-constituents and the constituent name, address, pointer and its type.
Get the name of next sub-constituent and its pointer.
Get the geometrical, location and constraints information of the sub-constituent (axis) from axis data structure;
Output the constituent and its descendants into OCDFF.
No
Last sub-constituent ?
Yes
No
Last constituent or device?
Yes
Output of Common Data Format File
Traverse back to the top level of hierarchical data structure and search for the next data block at this level.
Data type= "Task" ?
No
Get next block and data type
Yes
Get the number of machine sub-task and task name.
End of ring?
B
END

Figure 9.5 The flowchart of pre-processing of a machine data structure into its common data format(to be continued)

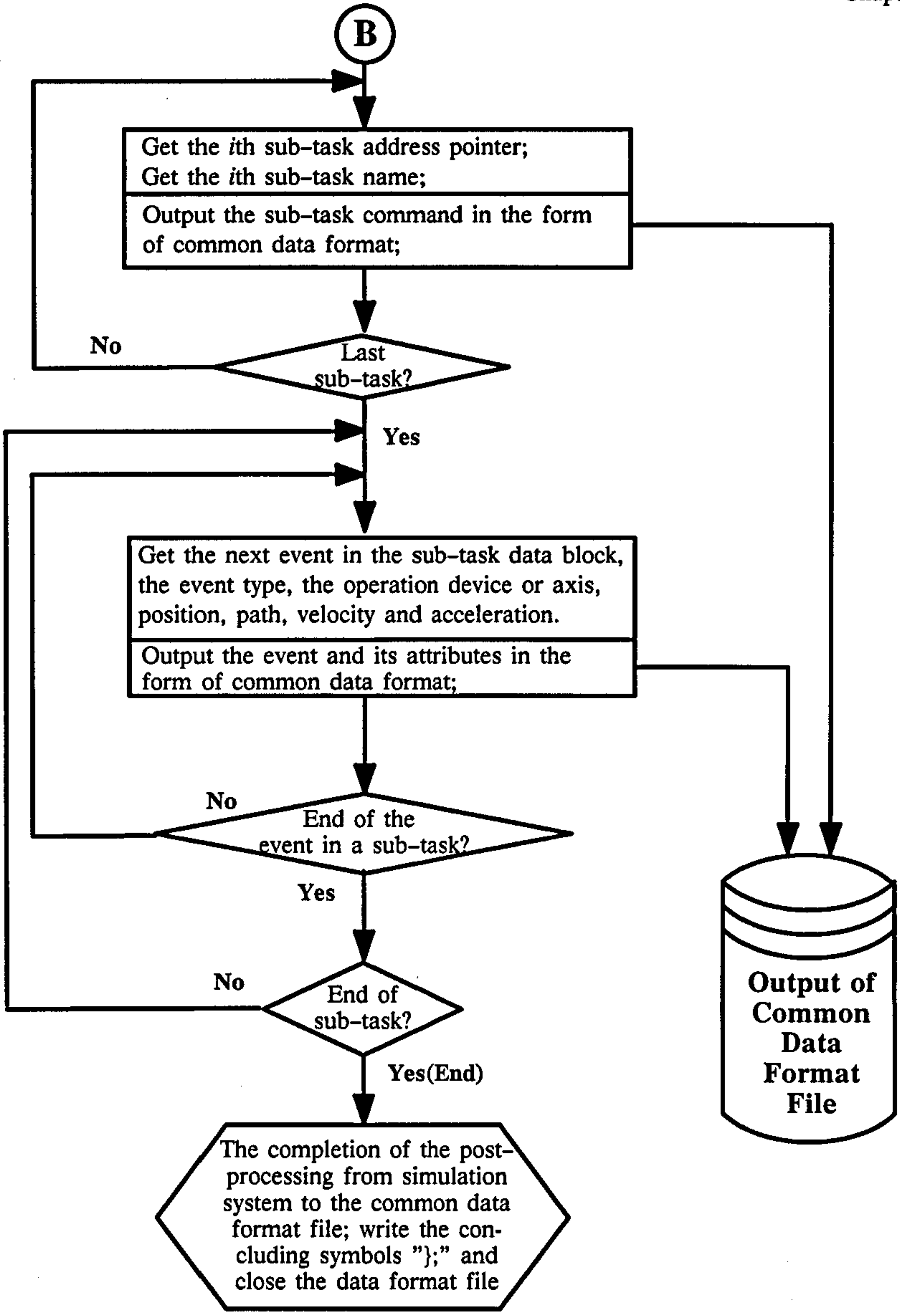

B
Get the *i*th sub-task address pointer;
Get the *i*th sub-task name;
Output the sub-task command in the form of common data format;
Last sub-task?
No
Yes
Get the next event in the sub-task data block, the event type, the operation device or axis, position, path, velocity and acceleration.
Output the event and its attributes in the form of common data format;
End of the event in a sub-task?
No
Yes
End of sub-task?
No
Yes(End)
The completion of the post-processing from simulation system to the common data format file; write the concluding symbols "};" and close the data format file
Output of Common Data Format File

Figure 9.5 The flowchart of pre-processing of a machine data structure into its common data format

the creation of more complex mechanisms. However, as the complexity of manufacturing methods increases and further integration requirements with other CAD-based special machine design systems arise, it is of great importance to evaluate the method and approach to interfacing with these systems. This is because these tools provide in-depth information for a special area of machine design which is not considered substantially in the simulation environment. Usually only this special part of information from a particular tool is desired in the data utilisation at the simulation phase and physical machine control. For example, the CAD geometry information of an assembled part, such as the shape, dimensions etc. can be used to derive position information for task planning and execution of a machine assembly operation. Due to this "specialized information" feature, the approach of writing a post-processor targeted at providing one kind of "special" information is adopted, and for each of these specialized tools one post-processor is written to interpret the information into part of the common data format. In this section, the method and implementation of a post-processor between a specialized mechanism design tool CAMLINKS and the simulation environment is given below.

9.5.2 The approaches to interfacing CAMLINKS and the simulation environment

CAMLINKS is a program for mechanism design and analysis. The MOTION program in the package enables the design of motions for either linear or angular coordinates with reference of time or an input motion. The output of the program is a data file describing the designed motion. Since the current simulation environment does not provide a cam motion design facility, it is desirable and feasible to interface the CAMLINKS motion design with the machine simulation environment. This can improve the functionality of machine design tool and provide a test-bed for cam software design and application. There are ten types of mathematical law which specify the motion type within a segment of a motion cycle (usually a cycle is divided into several segments). The user can select one of ten motion types to apply it to a segment. Therefore the output of a complete motion cycle always has the same number of data description sections as that of a segment division in motion cycle design.

The output of a CAMLINKS motion profile has two basic parts, namely the initial conditions and the parameters to specify an equation, and the calculated displacement, velocity and acceleration output with reference to a user specified input interval (see Appendix E for a CAMLINKS motion output). Based on this format two approaches were adopted to use the result of a CAMLINKS motion design.

9.5.2.1 Abstracting the displacement output to produce a designed motion profile

The first and easier approach adopted in this study was to save the data file of a designed motion profile on a data disk, place the disk on another PC machine which is connected to the Sun workstations via RS232, initiate the *ftp* (file transfer protocol) on the PC machine, and transfer the designed motion profile to a workstation through *ftp* using *put* command. This motion data file in ASCII format can then be read into the simulation environment. Figure 9.6 illustrates the above file transfer approach used.

When the motion file is read into the simulation environment, a function program called "filter" was used to abstract the motion profile data file. Basically the data file consists of following information:

Initial condition data block which defines the type of motion outputs (linear or

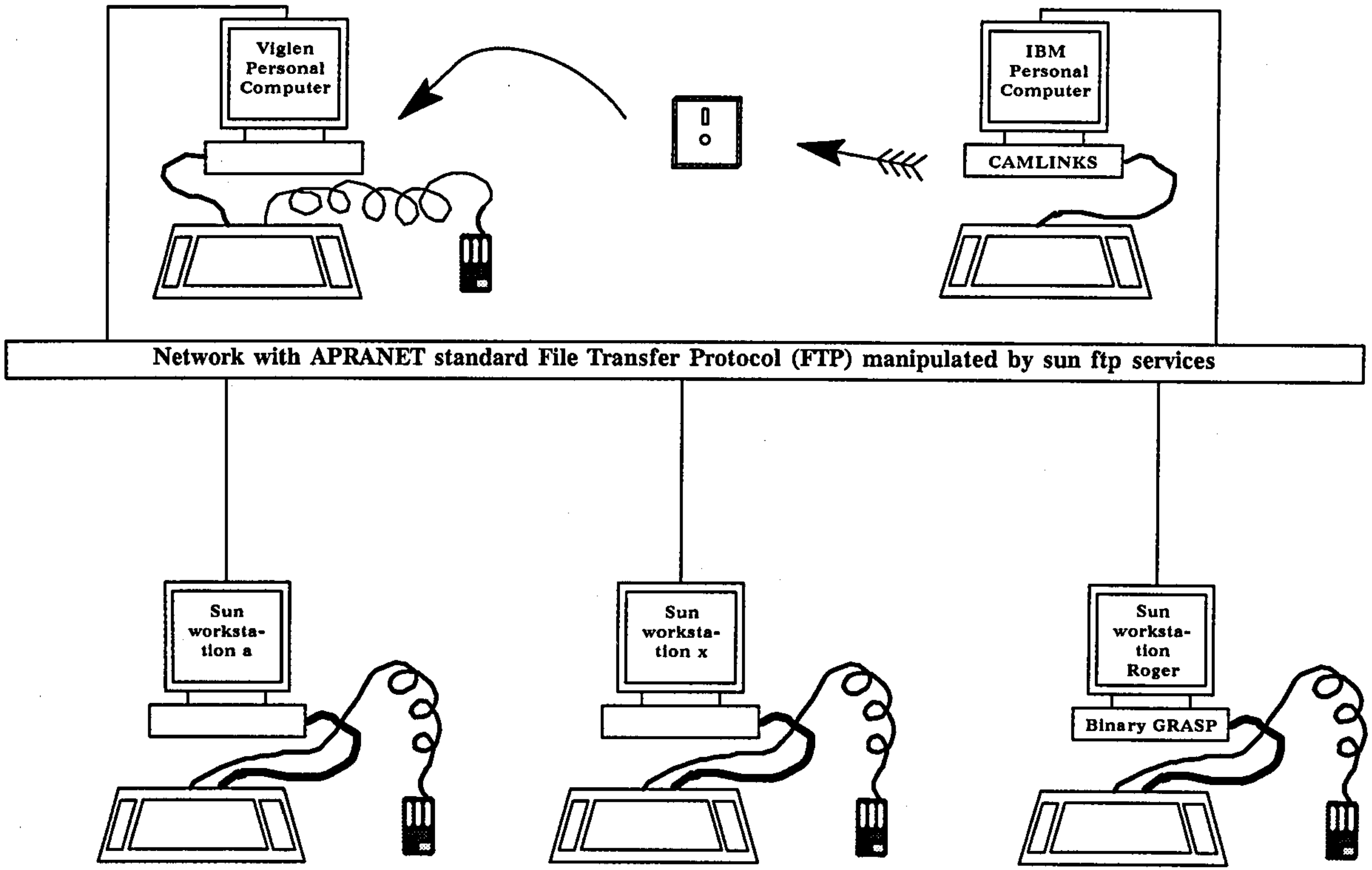


Figure 9.6 The network installation for a motion file transferring from PCs to Sun workstations

angular), the input and output value at the start of motion profile, and the speed of the master crank rotation;

Within each section there are possibly three parts to the data description. The first part specifies the segment number, segment type, input start point, end point and the increment, and the output start point, end point and output increment. Depending on the segment type, there is a further part of description to define the segment profile precisely for some motion type. The third part of the segment description gives the output displacement, velocity and acceleration with reference to each input value which are determined by the master crank rotational speed and the number of total revolute steps in one cycle based on the following equation:

$Iv(i) = Iv(i-1) + \frac{Sm}{60} \times \frac{Ic}{Nc}$ if the motion is linear, or

$Iv(i) = Iv(i-1) + \frac{Sm}{60} \times \frac{Ic}{Nc} \times 360 = Iv(i-1) + 60Sm \times \frac{Ic}{Nc}$ if the motion is angular, where ***Iv(i)*** is the input value at the *i*th step in a cycle, ***Sm*** is the master crank rotational speed, ***Ic*** is the total increment of input in a cycle and ***Ns*** is the total step number in a cycle.

The above section information repeats for each segment in a profile cycle until the profile comes to the end of a cycle. For an example of the data format for a motion profile, see Appendix E. Based on the above data information analysis, the corresponding algorithm to abstract the input and output displacement data is illustrated in Figure 9.7. Since the last input and output value set in one segment re-appears at the first line of the following segment, the last value set of input and output in the previous segment is ignored to avoid the repetition of the same value set. There should be no concern about the segment profile type due to the already available output values of the motion profile. However to provide clear information, the segment type value is still stored in a word and all output values in one segment are also stored in the same data block to be used as position data block to drive

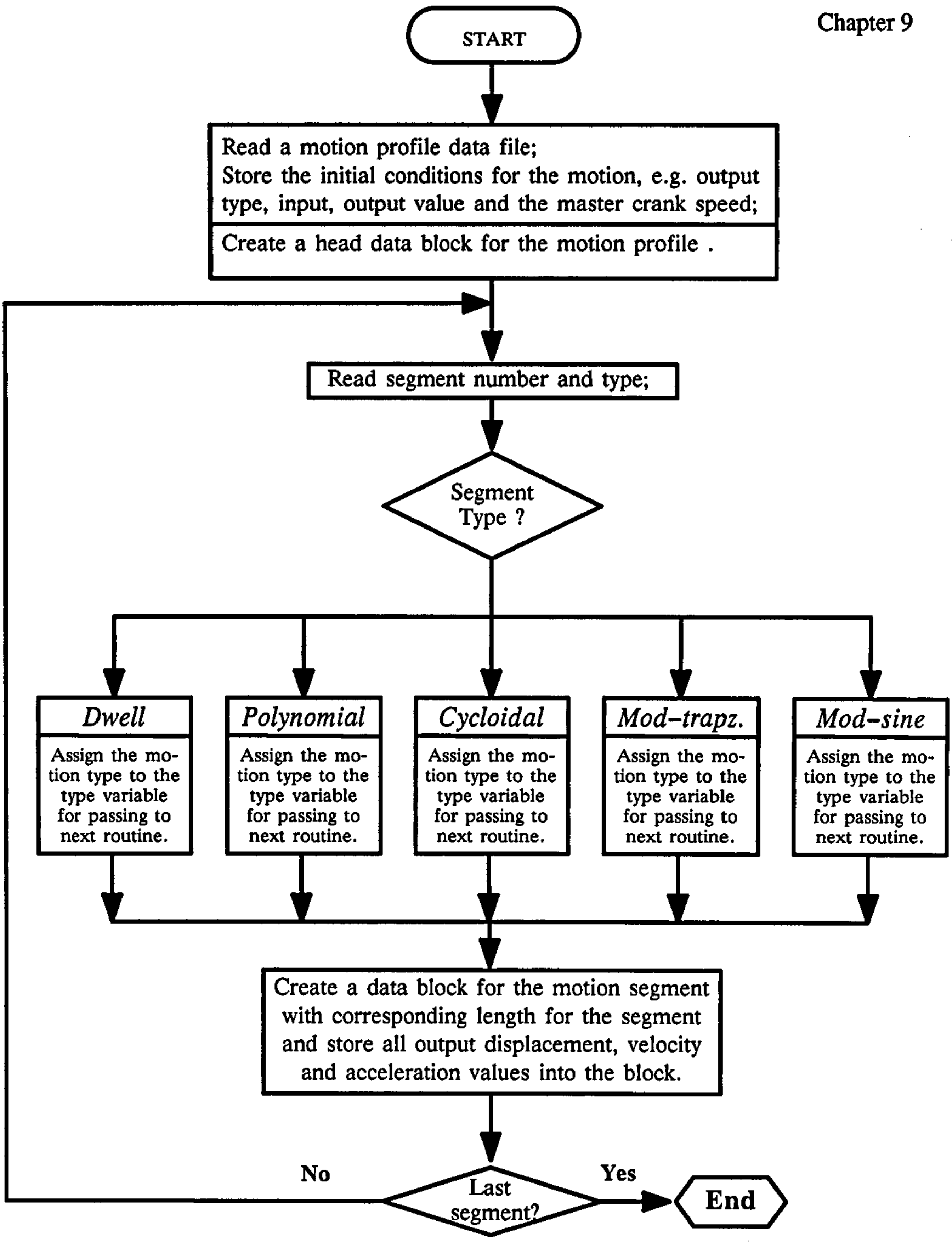

START
Read a motion profile data file;
Store the initial conditions for the motion, e.g. output type, input, output value and the master crank speed;
Create a head data block for the motion profile .
Read segment number and type;
Segment Type ?
Dwell
Assign the motion type to the type variable for passing to next routine.
Polynomial
Assign the motion type to the type variable for passing to next routine.
Cycloidal
Assign the motion type to the type variable for passing to next routine.
Mod-trapz.
Assign the motion type to the type variable for passing to next routine.
Mod-sine
Assign the motion type to the type variable for passing to next routine.
Create a data block for the motion segment with corresponding length for the segment and store all output displacement, velocity and acceleration values into the block.
No
Last segment?
Yes
End

Figure 9.7 The flowchart of transferring the data file from Camlinks to simulation system

an axis.

9.5.2.2 Deriving equations of each segment to calculate the output at the simulation stage

An advanced approach was also adopted in this study to facilitate the application of these motion profiles designed on CAMLINKS. Since CAMLINKS uses standard mathematical laws to describe each segment profile, it is desirable and possible to derive the mathematical equations for each segment along with the boundary conditions. This approach has following advantages.

(1) Large amount of data storage memory are saved. Due to the considerable number of motion profiles requirement for complex motion control, the output data value set can consume an extremely large amount of memory space. This is specially true when the number of calculated input and output value in the set increases, due to a time interval decrease. This can potentially slow down the searching and calculation of simulation activity;

(2) The user can specify the time interval at the simulation stage rather than go back to CAMLINKS to modify the number of motion steps in a cycle and recalculate the whole value set again;

(3) It is desirable to obtain the motion displacement versus time relationship in order to fully integrate the CAMLINKS design results into the simulation environment. In CAMLINKS, the simulation clock (or timing mechanism) is a very simple one. Because it does not cater for the multi-device machine design situation, the timing coordination among the distributed devices is not considered in the package. It is therefore very important to obtain the displacement versus time relationship to coordinate the motion with other device motions in the simulation. One of the

example is that CAMLINKS can not calculate the displacement with a different time interval for the same motion profile. This can happen in the simulation of multi-device machines.

To derive the equation expressing the displacement versus time relationship, the above motion profile data file is re-used. Since all initial conditions are defined in the CAMLINKS output motion file, they can be read into their corresponding data block as the boundary conditions for each motion segment. The motion profile definition in CAMLINKS is based on standard mathematical laws. The following motion laws are considered in the derivation of equation expression:

(1) a dwell law which provides a segment with a zero change in the position of the output (see Figure 9.8.1);

(2) a polynomial law which provides a general purpose motion profile based on the following expression:

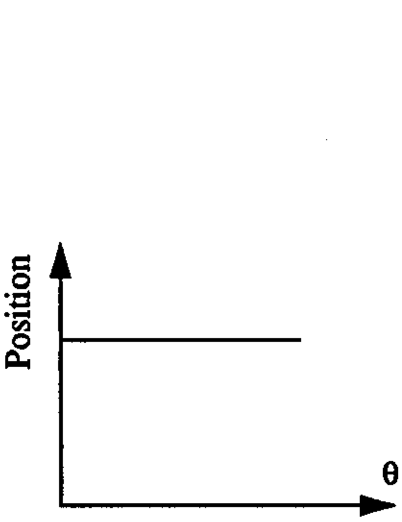
$$y=A_0+A_1x+A_2x^2+A_3x^3+A_4x^4+A_5x^5+A_6x^6+A_7x^7+A_8x^8+A_9x^9+A_{10}x^{10}+A_{11}x^{11}$$

Up to twelve of the above coefficients are determined by the boundary and via-point conditions (see Figure 9.8.2);

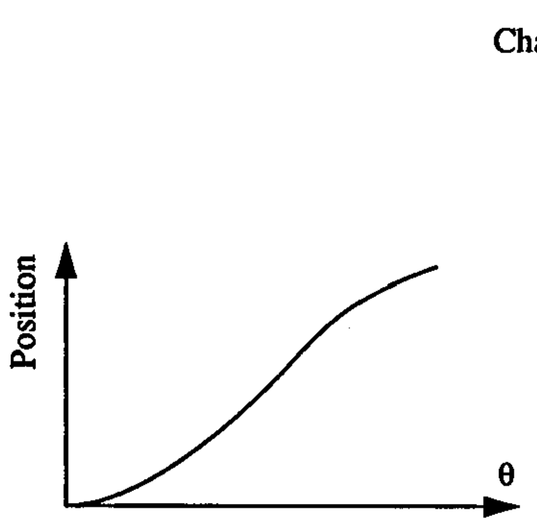
(3) a modified-sine law which provides dwell-rise-dwell or DRD motion. The only data requirement is the start value, end value and increments of the input and output displacements. The whole segment is divided into three sections in the range of $0 \le \theta \le \frac{1}{8}\beta$, $\frac{1}{8}\beta \le \theta \le \frac{7}{8}\beta$, $\frac{7}{8}\beta \le \theta \le \beta$

where β is the rotation cycle of a master crank (see Figure 9.8.3);

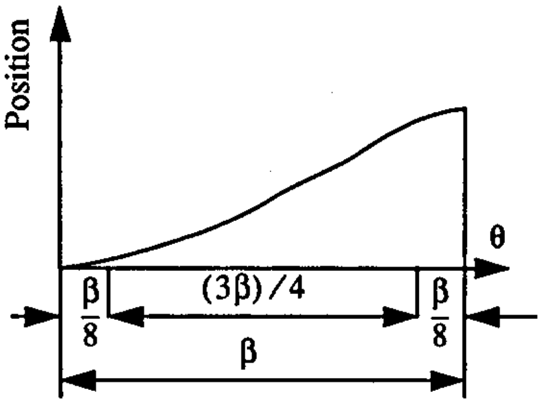
(4) a modified-trapezoidal law which also provides dwell-rise-dwell (DRD) motion



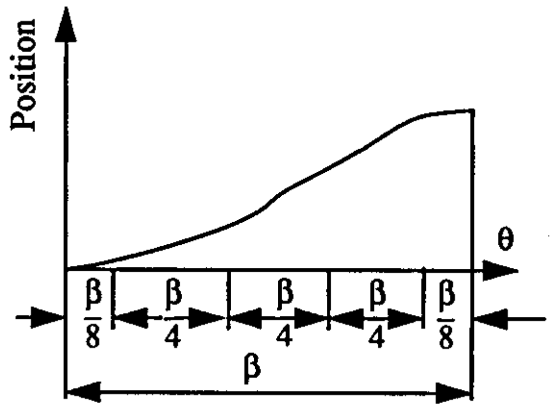
(1) A dwell motion profile



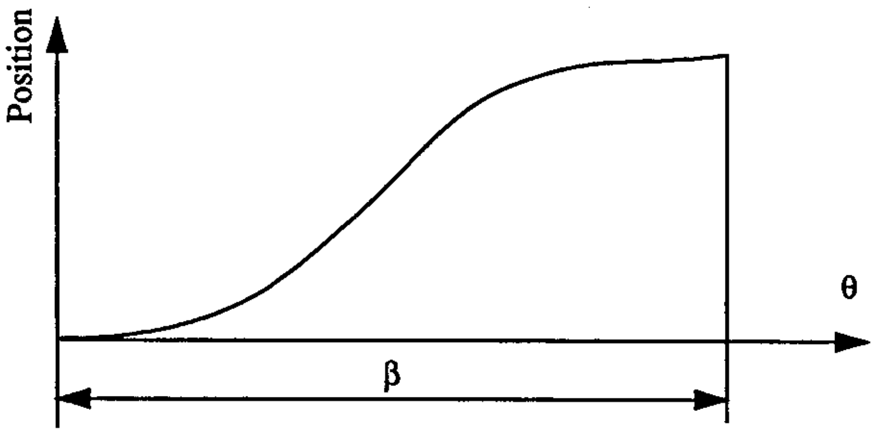
(2) A 3-4-5- polynomial motion profile



(3) A modified sine motion profile



(4) A modified trapezoidal motion profile



(5) A cycloidal motion profile

Figure 9.8 Some of motion laws and their motion profiles

which leads to relatively low dynamic forces [CAMLINKS 1991]. The data required is the start value, end value and increments of the input and output displacement since it is a standard law. The whole segment is divided into following sub-segments, $0 \le \theta \le \frac{1}{8}\beta$, $\frac{1}{8}\beta \le \theta \le \frac{3}{8}\beta$, $\frac{3}{8}\beta \le \theta \le \frac{5}{8}\beta$, $\frac{5}{8}\beta \le \theta \le \frac{7}{8}\beta$, $\frac{7}{8}\beta \le \theta \le \beta$,

where β denotes the rotational cycle of a master crank (see Figure 9.8.4);

(5) a cycloidal law which provides the simplest DRD standard law to apply in the case of high speed applications [Chen 1982]. There is no further division for cycloidal motion curve since the displacement can be expressed in one single equation. The input and output range are required to exactly specify the size of the curve (see Figure 9.8.5).

All equations for the above motion laws along with their curve diagrams can be found in Chen [1982].

As can be seen from the above description, the laws are fully determined by their boundary conditions apart from the polynomial law. With the substitution of these boundary conditions into the corresponding equations in each case, the intermediate displacement values can be calculated with the specified time interval. The equations for the polynomial law can be determined by using the parameter values provided by the polynomial motion data file. There are twelve parameters which are for the twelve parameters in $y = f(x)$ equation. Thus the polynomial equation is also fully determined and can be used to calculate the displacement values at a specific time interval. Figure 9.9 illustrates the schematic of equation abstraction, boundary initialisation and calculation implemented in this study at the simulation stage.

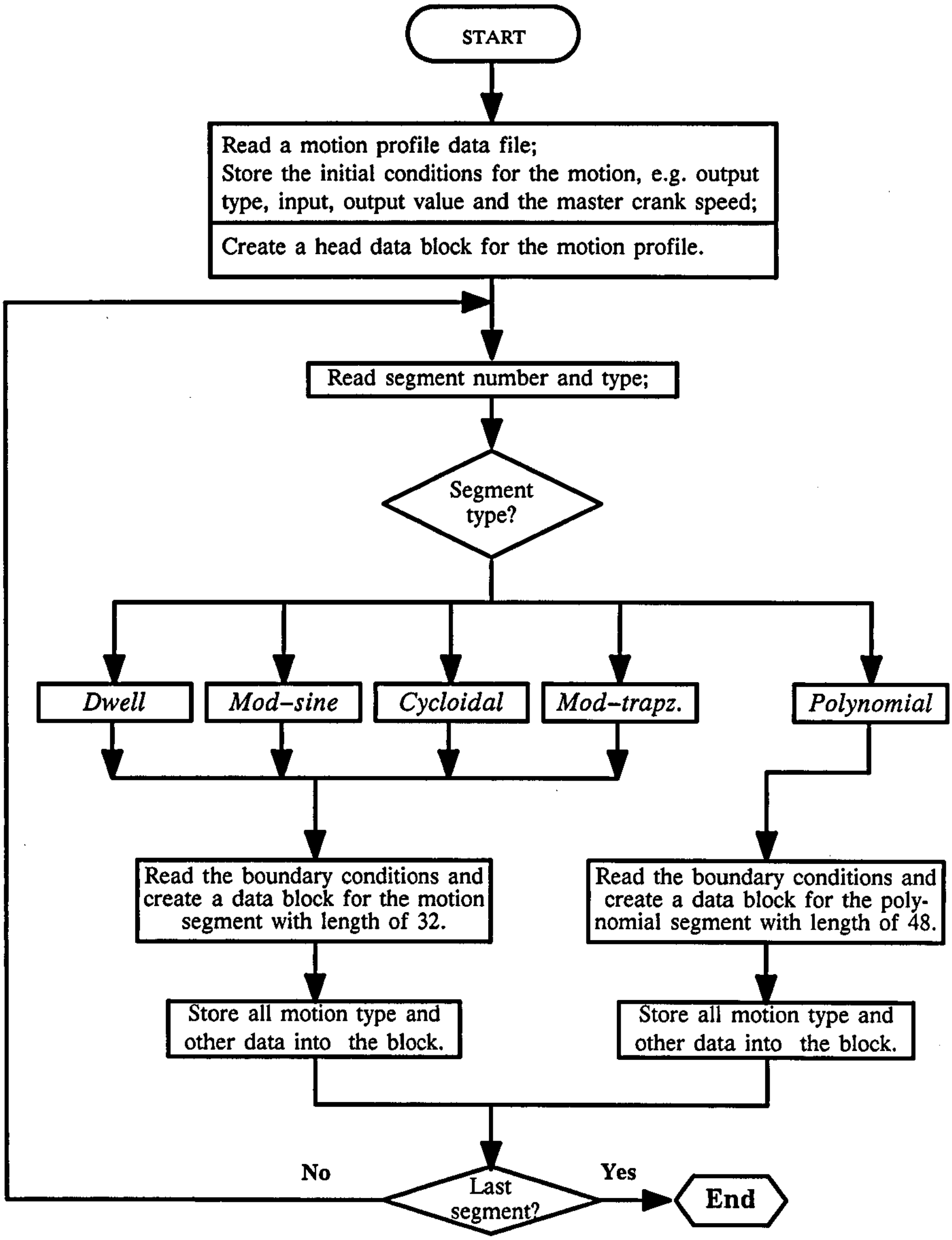

START
Read a motion profile data file;
Store the initial conditions for the motion, e.g. output type, input, output value and the master crank speed;
Create a head data block for the motion profile.
Read segment number and type;
Segment type?
Dwell
Mod-sine
Cycloidal
Mod-trapz.
Polynomial
Read the boundary conditions and create a data block for the motion segment with length of 32.
Read the boundary conditions and create a data block for the polynomial segment with length of 48.
Store all motion type and other data into the block.
Store all motion type and other data into the block.
No
Last segment?
Yes
End

Figure 9.9 The flowchart of transferring the initial conditions of a data file from Camlinks to simulation system

In the simulation phase, with the identification of segment motion type, the corresponding equation set is called and the parameters can be passed onto the equation. In order to determine the output displacement value at a specific time, the corresponding input time is required to achieve the calculation. The input value can be obtained through the following equation

$Iv(t) = \frac{Sm}{60} \times Di \times t$ if the output is linear, or

$Iv(t) = \frac{Sm}{60} \times Di \times t \times 360 = 60Sm \times Di \times t$ if the output is angular,

where ***Di*** is the difference between the end input value of the last segment of a motion cycle and the start input value of the first segment in a cycle. With this time value, the segment can be easily identified and the corresponding equation set can be found. The output displacement, velocity and acceleration are then calculated with the consideration of each segment boundary conditions.

9.6 Conclusion

In this chapter, a neutral common data format has been proposed in the form of a hierarchical text file format. The requirements for both physical machines and the simulation have been discussed. The interface between the simulation system and the common data format is described in terms of its pre- and post- processing. The algorithms for the implementation have been outlined.

As an example, the integration of a specialised motion design package CAMLINKS and the common data format has illustrated two aspects. The direct data format transfer enables the user to use the data output from other systems quickly at the possible inconvenience of inflexibility and huge amounts of data. The analytical approach to the integration of these systems provides flexible integration due to the knowledge exchange between systems. However this needs very close collaboration between system designers and this approach can prove difficult among commercialised systems.

The full integration of the simulation and the physical machine (UMC demonstration rig) has not been tackled thoroughly by the author due to time limitations and the large amount of work involved. Other researchers [Goh 1991] have used a relational database to abstract the different data requirements for each system from the common data format file. The integration of the common data format and the UMC physical machine has been partially implemented in this way. The data requirements of the UMC machine can be satisfactorily derived from the common data format file.

Chapter 10 Contribution to knowledge and future recommendations

10.1 Contribution to knowledge

This study has concentrated on the derivation of a modular machine design and simulation environment which aims at producing a flexible design environment for appraising the use of modular machines. The methodologies of creating such an environment have been studied and illustrated in this thesis and provide an essential basis for the derivation of a new type of machine design and simulation system for modular automated machines. In contrast to the conventional approach to modelling and simulating the industrial robot type of automated machine by using a fixed method of representation, a standard modular approach to describing the constituent elements of a machine has been proposed and the related data structures used to standardise modelling of each of the machine's constituent parts has also been created. A machine designer can select the necessary machine building primitives and aggregate them into a machine model in a modular manner by using the configuration tools derived in this Ph.D research. The machine model can then be associated with kinematic features and execution tasks for the purpose of simulating and animating the performance of the modelled machine. Articulated and distributed configurations of mechanisms have been demonstrated and show that the proposed methodologies in modelling a modular machine are well established and sufficient for the task.

In addition, the integration of a modelling system and other CAD based design or real time systems has been considered through the adoption of common data format structures.

Collectively these aspects represent an important contribution to the literature in the areas of enhancing flexibility in the modelling of modular machines, a reduction in the lead-time to design a machine, the derivation of configuration tools for modular machine design and the application of modelling results to life cycle engineering. In facilitating a proof of concept demonstration of modular machine design environment, more specific contributions to knowledge can be summarised as follows:-

(1)A library of machine primitive modules has been created based on an analysis of six types of motion pair. These machine primitives (which comprise two basic types - prismatic and revolute axis) can be used to generate the other four types of motion pair commonly found in traditional mechanisms. The establishment of such a library of machine primitives greatly facilitates the design activities for modular machines and provides a fundamental primitive source for the design and simulation of a modular machine. Each machine primitive has been parameterised by its geometric features, motion type and kinematic features. The data structure of a geometric primitive stored in the primitive library enables a user to fill in necessary parameters in order to create a user required motion axis. The methodology used in the establishment of the primitive library is based on the concept that a manufacturing machine can be decomposed into many elementary building elements both in terms of mechanical and control constituents. Data modelling techniques were used to derive a generalised data structure to include the necessary information for a motion axis and the generalised data structure for all motion axis types has been defined as a standard ring and tree structure in this study. This structure is used frequently as a module to describe every new motion axis. Through the study of these motion pairs and their data structural representation, it is concluded that this data structure generalisation greatly increases the modularity of the whole modelling environment and provides a fundamental basis for the modelling of modular machines which can thereafter be aggregated by selecting motion axis primitives from the primitive library. This general purpose ring and tree structure can also be used as a standard motion module to model other automation devices in a modular manner, and this approach therefore overcomes the disadvantages which result from most current approaches towards modelling of manufacturing equipment, where different or customised data structures may be required for each class of entity modelled. The establishment of a library of machine primitives represents a major part of this study.

(2) A general multi-level ring and tree structure was designed to model various aspects of modular machines, including geometry modelling, kinematic modelling and task description with respect to its working environment. The methodology of modelling a modular machine and its environment is based on the hierarchical concept of a modular ring and tree data structure.

(a) Modular machine's geometry was modelled in a hierarchical manner to facilitate the search for model elements and clarify the structural relationships within a modelled machine.

(b) Kinematic modelling is another aspect of modular machine modelling, and position, motion path and velocity and acceleration were modelled as a kinematic sub-structure. A desired kinematic feature of a motion for a modular machine or a device within the machine needs to be specified clearly before the motion is actually activated. Separation of the kinematic and geometric aspects improves the modelling system clarity and ease of manipulation, and furthermore this decomposition also parallels that in the physical modular machine.

(c) Task specification is a further aspect of modular machine modelling that is also a separate issue in the physical modular machine. A hierarchical structure of task specification was adopted and corresponding task information was interpreted into data contents in a task description data block which is arranged in a ring and tree structure. The hierarchical modelling ring and tree adopted and implemented in this study is based on the result of a modular machine decomposition to maintain the modularity concept.

The use of a hierarchical modelling tree and a variety of data blocks has been shown to improve the clarity of representing various aspects of a modular machine's information; it also facilitates the manipulation of the data contents; it arranges the various data in a consistent and organised way; and it also improves the efficiency of model element searching although a multi-level hierarchy may appear to be more complicated when simply searching along one branch. This data structure approach has been favoured for modular machine modelling in this study and can be potentially beneficial for other types of machine modelling. The implementation of the above data structure for the modelling of modular machines is another major part of the research.

(3) A set of configuration tools has been created for the manipulation of created motion axes, including locating an axis, forming a device, deforming a device, geometric manipulation of an axis etc. This set of configuration tools provides a very useful aid for modular machine designers by allowing them to select some motion primitives, locate them at a desired position and orientation, and aggregate them into a functional device to achieve a certain task. A designer of a machine can also use these tools to change selected dimensions of an axis, to scale an axis to a user-defined scale, and to break down a formed device into its original machine building primitives for either re-use or deletion. These tools are of vital importance in improving the system's user friendliness and can provide users with more manoeuvrability over the modelled machine arrangement and reconfiguration. The configurability of a modelled modular

machine is very much dependent on these tools. A SunView window facility was also studied and implemented to further facilitate the specification of motion primitive creation and modification. This facility can be more extensively used for the specification of all parameters within the design and simulation environment based on the study of motion primitives. However due to the time limitations this work is left for future implementation. An open end is also left for modular machine designers or future implementations to add more configurations of modular mechanisms. This method was adopted in this study because the potential complexity of a modular machine's configuration is beyond current implementation and this method can be helpful for new configuration users. Configuration tools are provided both for general geometries and machine primitives and only tools for the configuration of modular machines was devised in this study as the former already existed in Grasp.

(4) Four types of robot configuration (i.e. cartesian, cylindrical, revolute and spherical configurations) [Moore 1986] have dominated articulated mechanisms in automated machines although other articulated configurations have been proposed by some researchers. A comprehensive study of the kinematic synthesis and analysis of configurations with up to three degrees of freedom has been carried out to provide a theoretical basis for the use of different configurations. The well known representation - Denavit-Hartenberg notion of transformation matrices was adopted to deal with most of these configurations. It has been found that D-H representations are not adequate to handle every possible configuration of up to three degrees of freedom. In these situations, a modified 4×4 matrix representation was derived analytically by the author to compensate for the inadequacies of D-H representation. Seven sets of forward and inverse kinematic solutions were derived for all seven possible configurations of two axis groups. Twelve sets of forward kinematic solutions were also derived for all twelve possible configurations of the three axis

groups and three sets of inverse kinematic solutions were also derived for three of the twelve configurations. The remaining inverse solutions for the other configurations can be obtained in the same way. Kinematic analysis has to be done as no solutions to all configurations studied are readily available to anyone.

(5) A set of machine task related processors has been designed for simulation and animation purposes. Through the use of this set of processors it has been shown that they are the key elements of a modular machine simulation environment and they are also devices which can be used to generate data to drive a machine model during animation. Due to the complexity of machine configuration, it is essential to understand that a variety of processors are required to cater for the variation of processing requirements of different functional devices which form modular machines. This has been illustrated by creating processors for articulated, and distributed configurations of mechanisms and sensory devices. A simulation clock has also proven to be an important requirement for the simulation of modular machines, providing a time reference to co-ordinate the processing of different events. A requirement for an event manager (or controller) within a simulation environment has also been demonstrated, thus with reference to simulation time it has been shown that the event controller can co-ordinate the various machine processors. Since these simulation mechanisms and processors are key elements of a modular machine simulation environment, they represent another major part of contribution.

(6) It has been shown that the manufacturing operations performed by modular machines can be generalised into hierarchical levels, this leading naturally to a three level operation-oriented methodology for flexible modular machine programming. It can be concluded that programming at the lowest level (i.e. at the level of manipulating single machine primitives) allows highly trained staff to have more capability to

describe a specific manufacturing task in detail, whereas second level (i.e. at device operation level) programming enables the user to program a machine as a whole in a more abstract form rather than be concerned with details of its individual primitives. At an object level, programming of a modular machine is an even higher level of abstraction. However from the viewpoint of task programming the lowest level programming is the most complicated and difficult, whereas the second level programming is less complicated and the object level is the easiest among the three levels. It has also been illustrated that the use of a hybrid of second level and object level programming can lead to a flexible programming environment. The description of both generic and application dependent operations are required to form a complete programming facility. The implementation of the programming capability necessarily was limited to the assembly related manufacturing tasks.

(7) Aspects of systems integration were addressed in this study with interfacing achieved between the proof of concept modular machine simulation system, other proprietary CAD systems and the physical machine. Here a use of a common data format approach was used and generalised by the author. Pre- and post- processors were implemented and used between the common data format and the simulation environment: this having been the first phase of work required to enable the sharing of information created at various life cycle stages. Interfacing between the simulation environment and a proprietary CAD based systems was also considered in a second phase of the authors' integration study; this being to enable the simulation environment to access information created during other design processes. The third phase has been the interfacing between the simulation environment and physical machines and a method has also been proposed in this regard.

10.2 Implications and future recommendations

A prototype of a simulation environment for modular machines has been proposed and implemented in this PhD research study. The following suggestions and recommendations can be made based on the findings of the research.

10.2.1 Modelling of modular machines

Articulated and distributed configurations have been the main focus of this study since they represent the majority of motion mechanisms. However, it is also necessary to model other mechanisms such as parallel mechanisms which form a motion mechanism in a closed form [Khalil and Kleinfinger 1986] thereby having redundant degrees of freedom [Cleary and Tesar 1990, Karlen et al. 1990], mobile mechanisms (e.g. walking mechanisms) and hybrid systems of parallel and articulated mechanisms. Generally parallel mechanisms will have the advantage of a large payload bearing capability: by using parallel link structures high-precision can be realised when positioning its end-point. A mobile mechanism is used to change the base location within the working environment and is very useful for transporting components to some other location away from the current operation site. A hybrid configuration of parallel and serially chained mechanisms may achieve the advantages associated with both types of mechanisms, i.e. having a large payload bearing capacity, rigidity, high-precision achieved by parallel structure in the hybrid system, and large working envelope and dexterity associated with serially chained configurations. The current simulation environment provides the capability of graphically describing all the above mechanisms. However, the internal data association among the different data structures for each of the machine primitives needs to be realised according to a particular mechanical structure. It is very important to achieve this internal data association in order to make the modular machine simulation environment more versatile and powerful. It can be anticipated that it should be easy to incorporate the above mechanisms into the simulation environment by establishing the kinematic relationships and solutions for each mechanical structure in a newly created head data block.

Solid modelling was the only geometric modelling technique used within the current implementation. Other geometric modelling techniques have some particular advantages, and, it is necessary to investigate the possibility of using other techniques and incorporating them within one simulation environment. Thus different techniques could be used to model the aspect of a machine in which the technique offers advantage. For example, surface modelling allows curve of surface descriptions which is vital to some applications such as arc cutting for turbine blades of aircraft engines.

10.2.2 Kinematic and dynamic modelling

The kinematics of certain classes of robot manipulator has been well established, but there are no general closed form solutions. Numerical methods have to be used to solve the inverse kinematics. Due to the wide diversity of mechanical configurations possible with modular machines, more complex kinematic problems will be encountered in the situation where branching of a new mechanical chain starts or where a hybrid mechanism of parallel and serially chained configuration is employed. Since there is a discrepancy in the parameter representation between an articulated configuration (usually represented by the D-H parameters) and a hybrid mechanism of parallel configuration and articulation, a new method of kinematically describing them needs to be devised. Benhabib et al. [1989] proposed a method to describe a kinematic chain with some branches of articulation, and this method needs evaluating and enhancement for a general modular configuration within this implementation. There are often redundant degrees of freedom for these modular configurations, thus a general approach towards representing these kinematic redundant degree of freedom and the criteria used to determine which joint paths are best for a given situation should be exploited for a better understanding of the configuration.

Dynamic modelling of a modular machine incorporates its mass, compliance and damping to simulate the machine's response to the force and external loads of each machine motion primitive. Dynamic modelling of robotic systems has been addressed by researchers [Kang and Freeman 1990, Cox and Tesar 1989, and Cho et al. 1989]. A dynamic model can be used to evaluate the dynamic conditions, response and performance of a robot when it executes an assigned task. A powerful capability of simulating the dynamic features of a robot can facilitate the optimization of a robot design and provide invaluable assistance for robot designers. A modular machine can also benefit from the dynamic modelling and simulation by graphically evaluating the influence of some dynamics-related parameters and by finally optimising them with respect to a particular application. The dynamic performance measure, and criteria for optimization of dynamic features of a modular machine needs to be evaluated so that it can complement the simulation environment.

10.2.3 System integration

System integration has recently been a very attractive research area aimed at gaining a better understanding of integration requirements and benefits within various sections of a manufacturing enterprise. Weston et al. [1989c, 1992] realises that in addition to an established interface which enables an inter-communication between manufacturing sub-systems, applications control and information support services are also required to achieve true system integration.

Various system integration issues were discussed in Chapter 9 with particular reference to establishing interfaces with a CAD based motion design software package and the UMC physical machine: thus interface capabilities were implemented and requirements of some basic information support tools were considered. Essentially an advancement of this integrated facility could be realised with two application foci in mind. The first focus could centre on providing an integrated set of tools aimed at covering the various life cycle phases of machines, ranging from tools to assist machine design, through building and run time, to enabling and supporting change. Thus this first integrated toolset would be specific to the need of shop-floor manufacturing machines. A second, much broader focus could be to provide tools to support enterprise-wide integration, where the machine toolset would be just one sub-system but could access, or provide, information on a much wider basis. This wider integration scope can be viewed conceptually as horizontal and/or vertical integration (see Figure 10.1) [Weston and Davies 1992], where in general benefits of

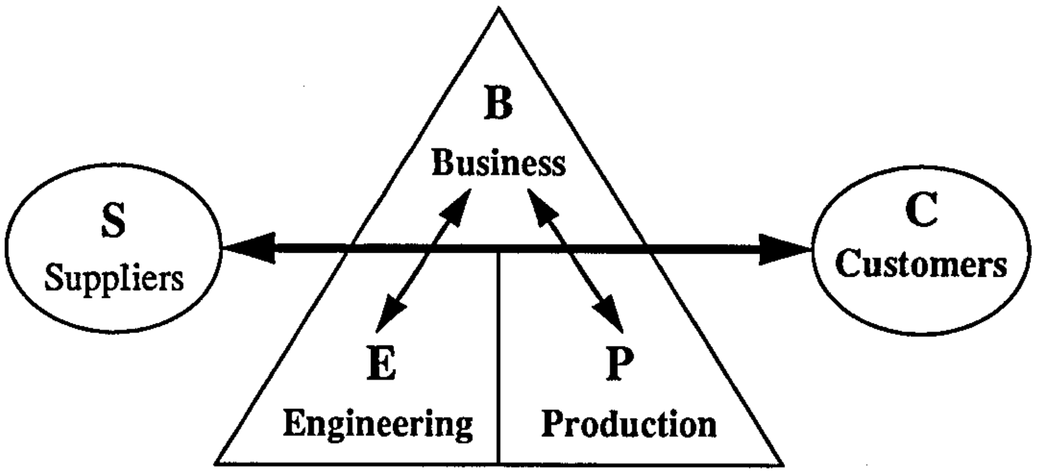


Figure 10.1 A Functional decomposition of an enterprise

improved and faster decision making can lead onto a reduction in product leadtimes and better products.

In the case of either focus it will be necessary to bridge existing gaps between the various methods and tools used through the engineering life cycle. The proof of concept, integrated machine design and simulation environment described in this thesis can provide a useful stepping stone towards this aim. Figure 10.2 illustrates conceptual requirements to achieve wider scope integration between system design and system application: where integration tools can deal with information exchange, communication establishment and application support. Such approach can offer the following benefits compared with existing manufacturing systems:

- partial evaluation of a manufacturing system before they are physically realised; leading to improve system design;
- opportunities to achieve a measure of optimization of the system design and automation; through system evaluation;
- reduction in product leadtime through improved decision making;
- provision of consistent models for various phases of engineering life cycle; to systemise design processes and avoid duplicated design effort;

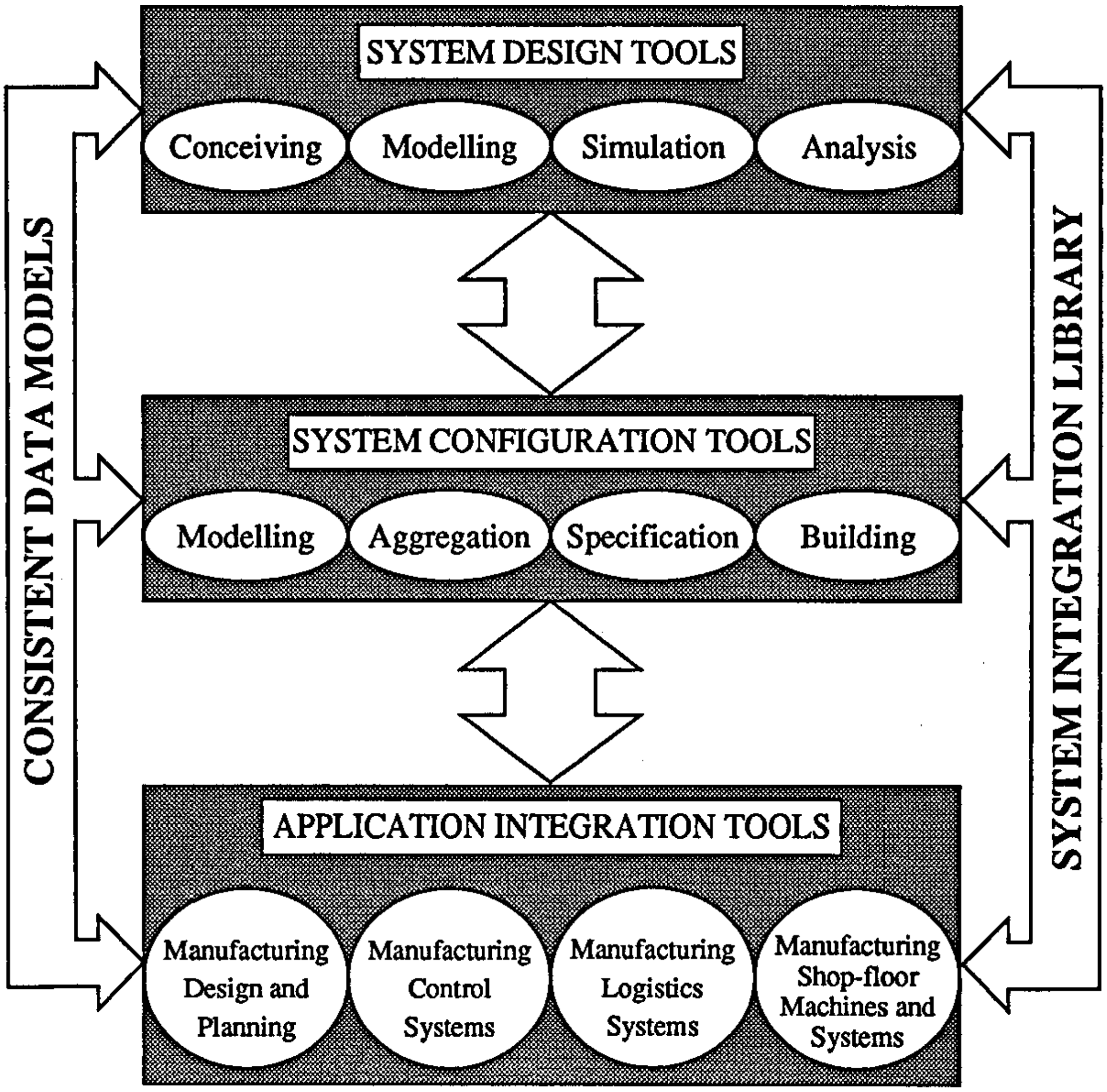

SYSTEM DESIGN TOOLS
Conceiving
Modelling
Simulation
Analysis
SYSTEM CONFIGURATION TOOLS
Modelling
Aggregation
Specification
Building
APPLICATION INTEGRATION TOOLS
Manufacturing Design and Planning
Manufacturing Control Systems
Manufacturing Logistics Systems
Manufacturing Shop-floor Machines and Systems
CONSISTENT DATA MODELS
SYSTEM INTEGRATION LIBRARY

Figure 10.2 An enterprise wide system integration approach

References:

Agapakis, J.E., Katz, J.M. and Pieper, D.L., "Programming and Control of Multiple Robotic Devices in Coordinated Motion", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 362-367, May 13-18, 1990.

Allen, D.K., "Architecture for Computer-Integrated Manufacturing", Annals of the CIRP, Vol. 36, No. 1, pp. 351-354, 1987.

Ambler, A. P., Popplestone, R.J., and Kempf, K.G., "An Experiment in the Off-Line Programming of Robots", Proceedings of 12th ISIR Paris, France, June, 1982.

Anderl, R. et al., "IGES review and proposed extensions", TAP Position Paper, DIN-NAM 961.412-84, 1984

Ang, Jr., M. H. and Tourassis, V. D.,"General- Purpose Inverse Kinematics Transformation for Robotic Manipulators", Journal of Robotic Systems,Vol. 4, No. 4, pp. 527-549, 1987.

Ang, Jr., M. H. and Tourassis, V. D.,"Flexible Manufacturing Using Modular Robotic Wrist", International Conference on CAD/CAM, Robotics and Factories of the Future, pp. 166-170, 1988.

Arai, T., Takashima, S., Hirai, S. and Sata, T., "Standardization of Robot Software in Japan," 15th International Symposium on Industrial Robots, pp. 995-1002, 11-15 September, 1985, Tokyo, Japan.

Armstrong, W.M., "Recursive Solution to the Equations of Motion of an N-Link Manipulator," Proceedings of the 5th World Congress on the Theory of Machines and Mechanisms,Vol. 2, July, pp. 1343-1346, 1979.

Armstrong, B., Khatib, O. and Burdick, J., "The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm," Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, pp. 510-518, April 1986.

Ball, D.A. and Smith, P., "The Practical Use of 3D Simulation in Industry", Proceedings of the 4th International Conference Simulation in Manufacturing, Edited by Browne, J and Rathmill, K., pp. 83-91, London, 1988.

Bartel, D.L. and Marks, R. W. ASME Journal of Engineering Industry, pp. 171-178, 1974.

Batory, D. S. and Kim, W., "Modelling concepts for VLSI CAD objects" ACM Transactions of Database system,Vol. 10, No. 3, pp. 322-346, 1985.

Benhabib, B., Zak, G., and Lipton, M.G., "A Generalised Kinematic Modelling Method for Modular Robots", Journal of Robotic Systems, Vol. 6, No. 5, pp. 545-571, 1989.

Benhabib, B., Cohen, R., Lipton, M.G. and Dai, M.Q., "Design of a Rotary-Joint-Based Modular Robot", in Cams, Gears, Robot and Mechanism Design, (Presented at the 1990 ASME Design Technical Conference), Proceedings of 21st Biennial Mechanisms COnference, pp. 239-243, Chicago, Illinois, September 1990.

Blaha, J.R., Lamoureux, J.P. and McKee, K.E., "Currently Available Robot Programming Languages," In Control and Programming in Advanced Manufacturing, Edited by Rathmill, K., IFS Publication, Springer-Verlag, 1988.

Booth, A., "An Introduction to UMC" Internal report, MS Research Group, The Department of Manufacturing Engineering, University of Technology, Loughborough, 1990.

Brantmark, H. and Ramstrom, K.G.,"ASEA Off-line Programming System - A User Friendly Implement", Proceedings of 15th International Symposium on Industrial Robots, Tokyo, Japan, pp. 741-749, Vol. 2, September 1985.

CAMLINKS manual, "CAMLINKS", by Limacon Consultants in Machine Design and Control, 1991.

Case, K., "Configuration Tools and UMC", Internal Technical Report within the Modular Research System Group in the Department of Manufacturing, Loughborough University of

Technology, December, 1990.

Chan, S.F., Weston, R.H. and Case, K.,"Robot Simulation and Off-line Programming", Computer-Aided Engineering Journal, pp. 157-162, August 1988.

Chan, S.F., "Advancement in Robot Programming with Specific Reference to Graphic Methods," Ph. D. Thesis of Department of Manufacturing Engineering, Loughborough University of Technology, 1989.

Chatila, R. and Giralt, G., "Task and Path Planning for Mobile Robots", in "Machine Intelligence and Knowledge Engineering for Robotic Applications", Edited by Wong, A.K.C. and Pugh, A., NATO ASI Series Vol. F33, Springer-Verlag, pp. 299-330, 1987.

Chen, Fan Yu, "Mechanics and Design of CAM Mechanisms", Pergamon Press Inc. 1982.

Cleary, K. and Tesar, D., "Incorporating Multiple Criteria in the Operation of Redundant Manipulators", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 618-624, May 13-18, 1990.

Craig, J.J., "Anatomy of an Off-line Programming System," Robotics Today, pp. 45-47, February 1985.

Craig, J.J., "Issues in the Design of Off-Line Programming Systems," in International Symposium of Robotics Research, Edited by Bolles, R. and Roth, B., MIT Press, Cambridge, MA, 1988.

Craig, J. J.,"Introduction to Robotics- Mechanism and Control", Second Edition, Published by Addison-Wesley Publishing Company, Inc., 1989.

Crookall, J.R., "Education for CIM", Annals of the CIRP, Vol. 36, No. 2, pp. 479-497, 1987.

Crosnier, A. and Fournier, A.,"Simulation of Cameras and Proximity Sensors for Computer Aided Design for Robotics and the Off-Line Robot Programming," Proceedings of the

International Workshop on Industrial Applications of Machine Vision and Machine Intelligence, Seilgen Symposium, pp. 316-323, Tokyo, Japan, 1987.

Cutkosky, M.R. and Howe, R.D., "Human Grasp Choice and Robotic Grasp Analysis," Chapter 1 in Dextrous Robot Hands Edited by Venkataraman, S.T. and Iberall, T. Published by Springer-Verlag New York Inc., 1990.

Dadam, P., Kospert, K., Anderson, F., Blanken, H., Erbe, R., Gunauer, J., Lum, V., Pistor, P., and Walch, G., "A DBMS Prototype to Support Extended NF2 Relations: An Integrated View on Flat Tables and Hierarchies", Proceedings of the ACM SIGMOD Conference, ACM, New York, pp. 376-387, 1986.

Denavit, J. and Hartenberg, R.S., "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices", ASME Journal of Applied Mathematics, Vol. 77, pp. 215-221, 1955.

Derby, S., "Off-Line Programming of Two Industrial Robots," Robots 8th Conference Proceedings, Detroit, MI, USA, Vol. 2, pp. 20/65-20/76, 4-7 June, 1984.

Dillman, R. and Huck, M., "A Software System for the Simulation of Robot Based Manufacturing Processes", Robotics,Vol. 2, No. 1. Elsevier Science, Publishers (North-Holland), March, 1986.

Dillmann, R., "The Use of Computer-Aided Design Methods in Robotics", Welding and Cutting, Vol. 39, Part 4, pp. E63-E66, April 1987.

Dillman, R. and Huck, M., "A Relational Data Base Supporting CAD-Oriented Robot Programming", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, Il Ciocco, Italy, July 4-6, 1988.

Dimarogonas, A. D., "Computer Aided Machine Design," published by Prentice Hall International (UK) Ltd., 1988

Dooner, M., "Robotics Software and CADCAM", Computer-aided Engineering Journal, pp. 217-220, December 1984.

Doyle, R. and Case, K., "The Logical and Geometrical Modelling of a Universal Machine Control Architecture", European Technology Congress and Exhibition, Birmingham, C415/019, pp.189-197, July 1991.

Duelen, G., Bernhardt, R. and Schreck, G., "Use of CAD-Data for the Off-Line Programming of Industrial Robots", Robotics, Vol. 3, Part 324, pp. 389-397, 1987.

Duffey, M. R. and Dixon, J.R., "A Program of Research in Mechanical Design: Computer-Based Models and Representations", Mechanism, Machine, and Theory,Vol. 25, No. 3, pp. 383-395, 1990.

Duffey, N. D., Herd, J.T. and Philip, G.P., "A Structural Approach to the Control of Parallel Acting Co-operating Robots", Proceedings of International Symposium on Industrial Robots, pp. 323-332, April 1988.

Dupourque, V., "Using Abstraction Mechanisms to Solve Complex Tasks Programming in Robotics", Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, April 1986.

Durr, M., Huck, M., Kemper, A., Mohrholz, p. and Wallrath, M., "Using Conventional and Nested Relational Database Systems for Modelling CIM Data", Computer-Aided Design, Vol. 21, No. 6, pp. 379-392, July/August 1989.

Eastman, C.M., "The Use of Object-oriented Database to Model Engineering Systems", Proceedings of the International Workshop on Object-oriented Database Systems, Pacific Grove, California. IEEE Computer.

Edwards, J., "Machine Vision and Its Integration with CIM Systems in the Electronics Manufacturing Industry", Computer-Aided Engineering Journal, pp. 12-18, February 1990.

Fanghella, P.,Galletti, G. and Giannotti,E., "Computer-aided Modelling and Simulation of Mechanisms and Manipulators", Computer Aided Design, Vol. 21, No 9, pp. 577-583, November 1989.

Featherstone, R., "Robot Dynamics Algorithms" by Kluwer Academic Publishers, 1987.

Ferrandez, K., "The Use of Computer Graphics Simulation in the Development of Robotics Systems," ACTA Astronaut (UK), Vol. 17, No. 1, pp. 115-122, January 1988.

Floriani, L.D. and Bruzzone, E., "Building a Feature-Based Object Description from a Boundary Model", Computer-aided Design, Vol. 21, No. 10, December, 1989.

Forestier, P., "The CATIA Integrated Geometry Modeller," Computer Graphics, Proceedings of the International Conference, London, pp. 381-391, 1985.

Fougere, T. J., Chawla, S. D. and Karneva, J. J.,"ROBOT-SIM: A CAD Based Workcell Design and Off-line Programming System", ASME Winter Annual Meeting, Robotics and Manufacturing Automation, PED Vol. 15, 1985.

Fu, K. S., Gonalez, R. C. and Lee, C. S. G., "Robotics: Control, Sensing, Vision, and Intelligence", Published by McGraw - Hill Inc., New York, 1987.

Fukuda, T. and Kawauchi, Y., "Cellular Robotic System (CEBOT) as One of the Realization of Self-Organising Intelligent Universal Manipulator," Proceedings of 1990 IEEE International Conference on Robotics and Automation, pp. 662-667, Cincinnati, Ohio, 13-18 May, 1990.

Gini, Maria, "The Future of Robot Programming", Robotica, Vol.5, pp. 235-246, 1987.

Gleason, G.J. and Agin, G.J., "A Modular Vision System for Sensor-Controlled Manipulation and Inspection",Proceedingsof9thInternationalSymposiumandExpositiononIndustrialRobots, Washington. D. C., March, 1979.

Glib, "Grasp Library Subroutine Menual", BYG System Limited, Nittingham, 1989.

Goh, A.S., "Integration of UMC and common data format based on step", Internal miniute, MS Research Group, The Department of Manufacturing Engineering, University of Technology, Loughborough, 1991.

Goh, K. and Middle, J.E., "The WRAPS System -A Tool for Welding Robot Adaptive Programming and Simulation," Proceedings of the First National Conference on Production Research, Nottingham University, U.K. 9-10 September 1985.

Goldenberg, A.A., Benhabib, B. and Fenton, R.G., "A Complete Generalized solution to the Inverse Kinematics of Robots," IEEE Journal of Robotics and Automation, Vol. RA-1, No. 1, pp. 14-20, March 1985.

Gupta, K. C.,"Kinematic analysis of Manipulators Using the Zero Reference Position Description", International Journal of Robotics Research, Vol. 5, No. 2, pp. 5-13, Summer, 1986.

Gupta, K.C. and Kazerounian, K., "Improved Numerical Solutions of Inverse Kinematics of Robots", Proceedings of IEEE International Conference on Robotics and Automation, pp. 743-748, 25-28 March, 1985.

Gupta, K. C.,"Kinematics of Robot with Continuous Roll Wrist", IEEE Journal of Robotics and Automation, Vol. 4, No. 4, pp. 440-443, August 1988.

Han, C.S., Traver, A.E. and Tesar, D., "Using CAD/CAM in the Design of a Robotic Micromanipulator", Computer-Aided Engineering Journal, pp. 43-48, April 1989.

Harrison, R., "A description of UMC", Internal report, MS Research Group, The Department of Manufacturing Engineering, University of Technology, Loughborough, 1989.

Harrison, R., "A Generalized Approach to Machine Control", Ph.D Thesis of the Department of Manufacturing Engineering, Loughborough University of Technology, 1991.

Harrison, J.P. and Mahajan, R., "The IGRIP Approach to Off-Line Programming and Workcell Design," Robotics Today, pp. 25-26, August 1986.

Hasegawa, Masaki, Takata, Masayuki, Temmyo, T and Matsuka,H., "Modelling of Exception Handling in Manufacturing Cell Control and Its Application to PLC Programming", Proceedings of 1990 IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, pp. 514-519, May 13-18, 1990.

Haurat, A and Perrard, J., "ADAR: A New Vision of Tasks Programming for Robotized Industrial Workcells", Proceedings of International Symposium on Industrial Robots,. pp. 431-442, April 1988.

Heginbotham, W. B., Dooner, M. and Case, K., "Rapid Assessment of Industrial Robot's Performances by Interactive Computer Graphics", 9th International Symposium on Industrial Robots, Washington D,C., pp. 563-574, March, 1979.

Hemami, H., Jaswa, V. C., McGee, R. B., "Some Alternative Formulations of Manipulator Dynamics for Computer Simulation Studies", Proceedings of the 13th Allerton Conference on Circuit and System Theory, University of Illinois, pp. 124-140, October 1975.

Hornick, M. L. and Ravani, B., "Computer-aided Off-line Planning and Programming of Robot motion", International Journal of Robotics Research, Vol. 4, No 4, pp. 18-31, Winter 1986.

Hornick, M. L. and Ravani, B., "A Data Structure and Data Base Design for Model Driven Robot Programming", Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, pp. 1082-1086, April 1986b.

Howie, P., "Graphical Simulation for Off-Line Robot Programming," Robotics Today, Vol. 6, Part 1, pp. 63-66, 1984.

Huang, P.Y. and Houck, B.L.W., "Cellular Manufacturing: An Overview and Bibliography", Production and Inventory Management, Fourth Quarter, 1985.

Hunt, K. H., "Kinematic Geometry of Mechanisms", Published By Oxford University Press, 1978.

Hunt, K. H., "Kinematic Geometry of Mechanisms", Published By Oxford University Press, 1990.

Hutchinson, S.A. and Kak, A.C., "FProlog: A Language to Integrate Logic and Functional Programming for Automated Assembly", Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, pp. 904-909, April 1986.

Industrial Robot, "Cambridge Control Package; its Dynamic Approach," Vol. 14, No. 2, pp. 93-94, February 1987.

Imam, I and Davis, J.E., "Robot Simulation and Off-Line Programming - An Integrated CAE-CAD approach" Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, Il Ciocco, Italy, pp. 189-201, July 4-6, 1988.

Ishii, M., Sakane, S., Mikami, Y., and Kakikura, M., "A 3-D Sensor System for Teaching Robot Paths and Environment," International Journal of Robotics Research,Vol. 6, No. 2, pp. 45-59, 1987.

Izagnirre, A. and Paul, R.P., "Computation of the Inertial and Gravitational Coefficients of the Dynamic Equations for a Robot Manipulator with a Load," Proceedings of the 1985 International Conference on Robotics and Automation, St. Louis, pp. 1024-1032 March 1985.

Jacobs, M.P., "Off-Line Robot Programming: A Current Practical Approach," Robots 8 Conference Proceedings, Detroit, MI, USA, Vol. 1, pp. 4/1-4/11, 4-7 June, 1985.

Jafari, M. A., "Petri Net Based Shop-floor Controller and Recovery Analysis", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 532-537, May 13-18, 1990.

Juyaraman, R. and Levas, A., "A Workcell Application Design Environment (WADE)", in CAD Based Programming for Sensory Robots, Edited by B, Ravani, Published by Springer-

Verlag, NATO ASI Series, Vol. F50, pp. 91-120, 1988.

Jones, A. and Saleh, A., "A Multi-level/Multi-layer Architecture for Intelligent Shop-floor Control", International Journal of Computer Integrated Manufacturing Vol. 3, No. 1, pp. 60-70, 1990.

Jouaneh, M.K., Wang, Z. and Domfeld, D.A., "Trajectory Planning for Coordinated Motion of a Robot and a Positioning Table: Part1 - Path Specification", IEEE Transactions on Robotics and Automation, Vol. 6, No 6, pp. 735-745, December 1990.

Kacala, J., "Robot Programming Goes Off-Line," Machine Design, pp. 89-92, November 1985.

Kamm, L.J., "Recent Applications of Modular Technology Robots", SME.13th International Symposium on Industrial Robots and Robot7, (U.S.A), pp. 11.66-11.74, 1983.

Kane, T.R. and Levinson, D.A., "The Use of Kane's Dynamical Equations in Robotics," The International Journal of Robotics Research, Vol. 2, No.3, pp.3-20, Fall 1983.

Karlen, J.P., Thompson, J.M. and Vold, H.I., "A Dual-Arm Dexterous Manipulator System with Anthropomorphic Kinematics", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 368-373, May13-18, 1990.

Kato, Ichiro, "Mechanical Hands Illustrated", Survey Japan, Japan, 1980.

Kemper, A., "CAD Database: Requirements and Survey," Proceedings of 19th Annual Hawaii International Conference on System Sciences, Honolulu, HI,USA, pp. 363-378, January 1986.

Kemper, A. and Wallrath, M., "An Analysis of Geometric Modelling in Database Systems," ACM Computer Surrey, Vol. 19, No. 1, pp. 47-91, March 1987.

Khalil, W. and Kleinfinger, J.F., "A New Geometric Notation for Open and Closed-Loop Robots", Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San

Francisco, pp.1174-1179, April 1986.

Kitajima, K., "Interactive Robot Simulator for High-Level Tasks", Computer-Aided Design, Vol. 20, No. 2, pp. 93-99, March 1988.

Kiedrzynski, A and Becquet, M., "Light Structure Modular Design Using Honeycomb Links". Proceedings of 18th International Symposium on Industrial Robots, pp. 101-110, April, 1988.

Kota, S. and Chuenchom, T., "Adjustable Robotic Mechanisms for Low-Cost Automation", in Cams, Gears, Robot and Mechanism Design, (Presented at the 1990 ASME Design Technical Conference), Proceedings of 21st Biennial Mechanisms Conference, pp. 297-322, Chicago, Illinois, September 1990.

Kusiak, A. and Heragu, S.S., "Computer Integrated Manufacturing: A Structural Perspective", IEEE Network, Vol. 2, No. 3,pp14-21, May 1988.

Kusiak, A. and Park, K., "Concurrent Engineering: Decomposition and Scheduling of Design Activities," International Journal of Production Research, Vol. 28, No. 10, pp.1883-1920, 1990.

Larson, R. and Donath, M., "Animated SImulation of Intelligent Robot Workcell", Proceedings of Robots 9 SME Ref. MS85-622, pp. 19-54-19-69, Detroit, Michigan, USA, June 2-6, 1985.

Langrana, N. A. and Bartel, D. L., "An Automated Method for Dynamic Analysis of Spatial Linkages for Biomechanical Applications", Transaction ASME, Journal of Engineering Industry, Vol. 97, pp. 568-574, 1975.

Laugier, C., "Planning robot motions in the SHARP system," in "CAD Based Programming for Sensory Robots", Edited by Bahram Ravani, Published by Springer-Verlag Berlin Heidelberg, NOTO ASI Series, Vol. F50, pp. . 151-187, 1988.

Langier, C. and Pertin, J., "Automatic Grasping: A Case Study in Accessibility Analysis", Published in "Advanced Software in Robotics", Edited by Danthine, A and Geradin, M, North Holland, 1984.

Lee, C.S.G., "On the control of Robot Manipulators," Proceedings of 27th Soc. Photo Optical Instrumentation Engineers, Vol.442, San Diego, California, pp. 58-83, 1983.

Leu, M.C., "Robotics Software Systems", Robotics and Computer-Integrated Manufacturing, Vol. 2, No. 2, pp. 1-12, 1985.

Levas, A. and Jayaraman, R, "WADE: An Object-Oriented Environment for Modelling and simulation of Workcell Applications," IEEE Transactions on Robotics and Automation, Vol. 5, No 3, pp. 324-335, June 1989.

Li, Z. and Sastry, S., "Task-Oriented Optimal Grasping by Multifingered Robot Hands", IEEE Journal of Robotics and Automation, Vol. 4, No. 1, pp. 32-44, February 1988.

Lorie, R., "Issues in Databases for Design Applications", in File Structure and Databases for CAD, Edited by Encarnacao, J. and Krause, F. L., North-Holland, Amsterdam, 1982.

Lorie, R. and Plouffe, W, "Complex Objects and Their Use in Design Transactions". Proceedings of ACM SIGMOD Conference on Engineering Design Applications. San Jose, California, pp. 115-121, May 1983.

Luby, S. C., Dixon, R. R. and Simmon, M. K., "Creating and Using a Features Database", Computer. Mech. Eng., Nov. 1986.

Luh, J.Y.S., Walker, M.W., and Paul, R.P., "On-Line Computational Scheme for Mechanical Manipulators," Transaction ASME, Journal of Dynamic Systems, Measurement and control, Vol. 120(2), pp. 69-76, 1980.

Lyons, D.M. and Arbib, M.A., "A Formal Model of Computation for Sensory-Based Robotics", IEEE Transactions on Robotics and Automation, Vol. 5, No 3, pp. 280-293, June 1989.

Muhieddine, F. and Webb, D.C., "BCL -the Industrial CNC Standard", Computer-Aided Engineering Journal, pp. 54-56, April 1990.

Maier, D. Otis, A. and Purdy, A., "Object-oriented database development at Servio Logic", IEEE Database Engineering,Vol. 8, No. 4, pp. 58-65, 1985.

Maimon, D., "The Robot Task - Sequencing Planning Problems", IEEE Transactions on Robotics and Automation, Vol. 6, No 6, pp. 760-765, December 1990.

Mattis, A. and Gill, K.D., "The Best Robot for the Job: Simulation can Help Decide," The Industrial Robot, Vol. 15, No. 1, pp. 32-34, 1988.

McCloy, D and Harris, M., "Robotics - An Introduction," Published by Open University Press, Milton Keynes, England.

Milenkovic, V. and Huang, B., "Kinematics of Major Robot Linkages", Proceedings of 13th International Symposium on Industrial Robots and Robots 7, Chicago, Illinois, Vol. 2, 1983.

Middle, J.E. and Goh, J., "WRAPS - Welding Robot Adaptive Off-Line Programming and Expert Control System," Second International Conference Developments in Automated and Robot Welding, London, 17-19 November, 1987.

Milberg, J., Schrufer, N. and Tanber, A., "Requirements for Advanced Graphic Robot Programming Systems," IFAC, Robot Control, Karlsruhe, FRG, 1988.

Miller, R.K., "Manufacturing Simulation," SEAT Technical Publication and Technical Insight Publication, 1987.

Miller, D. J. and Lennox, R. C., "An Object-Oriented Environment for Robot System Architectures", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 352-361, May 13-18, 1990.

Moore, P. R., Weston, R.H., Thatcher, T.W. and Gascoigne, J.P., "Modular Robot Systems", Proceedings of 2nd IASTED International Symposium on Robotics and Automation, Lugano, Switzerland, pp. 165-169 June 1983.

Moore, P. R., "Pneumatic Motion Control Systems for Modular Robots", Ph. D.Thesis, Department of Engineering Production, Loughborough University of Technology, April 1986.

Moore, P. R., Weston, R.H., Booth, A. and Harrison, R., "An Open Control Architecture for Assembly Automation", The Proceedings of 11th International Conference on Assembly Automation, Detroit, Michigan, pp. MS90-838-1-MS90-838-17, November 11-14, 1990.

Orin, D.E., McGhee, R.B., Vukobratovic, M., and Hartoch, G., "Kinematic and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler Methods," Math-Biosciences Vol.43, pp.107-130, 1979.

Pai, D. K. and Leu, M. C., "INEFFABELLE - An Environment for Interactive Computer Graphics Simulation of Robotic Applications". Proceedings of IEEE International Conference on Robotic and Automation, San Francisco, California, April, 1986.

Paul, R. P.,"Robot Manipulators: Mathematics; Programming, and Control", The MIT Press, Cambridge, Massachusetts. and London, England, 1981.

Paul, B. and Rosa, J.,"Kinematics Simulation of Serial Manipulators", The International Journal of Robotics Research Vol. 5, No. 2, pp. 14-31, 1986.

Paul, R. P. and Zhang, H.,"Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation", The International Journal of Robotics Research, Vol. 5, No. 2, pp. 32-44, 1986.

Pieper, D.,"The Kinematics of Manipulators Under Computer Control", Ph.D. Dissertation, Computer Science Department, Standford University, Standford, CA, Oct. 1968.

Pinson, E., "A Simulation Environment for Robot Software Development". SPIE, Vol. 579, Intelligent Robots and Computer Vision, 1985.

Primrose, E. J. F., "On the Input-Output Equation of the General 7R Mechanism," Mechanism, Machine and Theory, Vol. 21, No. 6, pp. 509-510, November 1986.

Pratt, M. J., "Solid Modelling and the Interface Between Design and Manufacture", IEEE Computer Graphics and Applications, Vol.4, No 7, pp. 52-59, July 1984.

Pratt, M. J. and Wilson, P.R., "Requirements for Support of Form Features in A Solid Modelling System", Supplied to CAM-I as the Final Report of the Contract "Features Support in a Geometric Modelling SYstem (GMS) - Phase II", 1985.

Quin, "Intelligent Motor Control for Industrial Plant Automation", An Introduction to the Principles and Application of Intelligent Control Techniques to Solve Complex Velocity, Acceleration and Position Control Problems in Industrial Automation Projects, 1989.

Raczkowsky, J., Mittenbuhler, K. H. and Fohler, C., "Simulation of Vision in Robot Applications", In Robot Control, pp. 499-504, 1988.

Rajan, V. R. and Nof, S. Y., "A Game-Theoretic Approach for Co-operation Control in Multi-Machine Workstations," International Journal of Computer Integrated Manufacturing by Taylor & Francis Ltd., Vol. 3, No. 1, pp. 47-59, 1990.

Ravani, B., "World Modelling of CAD Based Robot Programming and Simulation", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, II Ciocco, Italy, pp. 67-89, July 4-6, 1988.

Ravani, B. and Hornick, M. L.,"STAR: A Simulation Tool for Automation and Robotics", in Control and Programming in Advanced Manufacturing, Rathmill, K.ed, International Trends in Manufacturing Technology, IFS Publish. pp. 269-294, 1988.

Requicha, A. A. G., "Representations for Rigid Solids: Theory, Methods, and Systems", ACM Computing Survey, Vol. 12, No. 4, pp.437-464, 1980.

Requicha, A. A. G. and Voelcher, H.B., "Solid Modelling: a Historical Summary and Contemporary Assessment", IEEE Computer Graphics and Applications, Vol. 2, No. 3, pp. 9-24, 1982.

Requicha, A. A. G., "Representation of Tolerance in Solid Modelling: Issues and Alternative Approaches in Solid Modelling", Edited by Pickett, M. S. and Boyse, J. W. Published by Computer Plenum Press, New York, USA,1984.

Requicha, Aristides, A.G., "Solid Modelling-A 1988 Update", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, II Ciocco, Italy, July 4-6, 1988.

Rieseler, H. and Wahl, F.M., "Fast Symbolic Computation of the Inverse Kinematics of Robotics", The Proceedings of 1990 IEEE International Conference on Automation and Robotics, Cincinnati, Ohio, pp. 462-467, May13-18, 1990.

Robotics World, "AutoSimulation," Vol. 13, No. 3, pp. 3, 1986.

Rock, S.T., "Intelligent Robot Programming: You Can't Get There From Here - A Viewpoint," Robotica, Vol. 6, pp. 333-338, 1988.

Rockey, K. E., et al., "The Finite Element Method", New York: Halsted Press/Witley, 1975.

Rogers, P., Williams, D.J., Wesley, P.S. and Clare, J. N., "On-Line Scheduling of Machining Cells Using Knowledge-Based Simulation", Proceedings of 4th International Conference of Simulation in Manufacturing, pp.151-163, November 1988.

Rogers, G.G. and Weston, R.H., "Modular Production Systems", Proceedings of Institution of Mechanical Engineers International Conference Mechatronics: Designing Intelligent

Machines, pp.31-35, September 1990.

Rui, A., Weston, R.H., Gascoigne, J.D., Hodgson, A. and Sumpter, C.M., "Automating Information Transfer in Manufacturing Systems," Computer-Aided Engineering Journal, pp. 113-121, June, 1988.

Sanderson, A.C. and Homem-de-Mello, L.S., "Task Planning and Control Synthesis for Flexible Assembly Systems", in "Machine Intelligence and Knowledge Engineering for Robotic Applications", Edited by Wong, A.K.C. and Pugh, A., NATO ASI Series Vol. F33, Springer-Verlag, pp. 331-353, 1987.

Sandgren, E., "A Multi-Objective Design Tree Approach for the Optimization of Mechanism", Mechanical Machine Theory, Vol. 25, No. 3, pp. 257-272, 1990.

Sandor, G. N., "The Seven Stages of Engineering Design", Mechanical Engineering, Vol. 86, No. 4, pp. 21-25, 1964.

Schek, H.J.and Pistor, P., "Data Structures for an Integrated Database Management and Retrieval System", Proceedings of the 8th International Conference on Very Large Database. Mexico City, VLDB.Endoment, Saratoga, California, 1982.

Schlechtendahl, E G (Editor) Specification of a CAD*I neutral file for solids. Published by Springer, 1986.

Schmitz, D., Khosla, P.K. and Kanade, T., "The CMU Reconfigurable Modular Manipulator System", Technical Report, Carnegie Mellon University, CMU-RI-TR-88-7, 16 pages, 1988.

Schreiber, R.R., "Robot System Simulation," Robotics Today, pp. 81-90, June 1984.

Shah, J.J and Rogers, M.T., "Expert Form Feature Modelling Shell", Computer-Aided Design, Vol. 20, No. 9, pp. 515-524, November 1988a.

Shah, J.J and Rogers, M.T., "Functional Requirements and Conceptual Design of the Feature-Based Modelling System", Computer-Aided Engineering Journal, pp. 9-15, February 1988b.

Shipman, D., "The Functional Data Model and Data Language DAPLEX", ACM Transactions on Database Systems, Vol. 6, No. 1, pp. 140-173, March 1981.

Siegler, A., Bathor, M. and Devi, G., "A 3-Dimensional Computer Animation System with Robotic Applications", Robotica, Vol. 5, pp. 281-290, 1987.

SILMA Inc., "Programming in SIL", Available from SILMA Inc., 1601 Saratoga - Sunnyrale Rd., Cupertino, Calif. 95014, 1989.

SILMA Inc., "Transmitting Power Intelligently", Professional Engineering, Mechanical Eng. Publications, pp. 52-54, April 1990.

SILMA Inc., "The CimStation User's Manual," Version 4.0. Available from SILMA Inc., 1601 Saratoga - Sunnyrale Rd., Cupertino, Calif. 95014, 1989.

Simkens,P., Van Brussel, H., Serrien, S. and Bryon, S.,"Generating Off-line Robot Programs with Geometrical Data From a CAD-Database", Proceedings of International Symposium on Industrial Robots, pp. 309-322, April 1988.

Sinha, P. K., "Transmitting Power Intelligently", Professional Engineering, Mechanical Eng. Publications, pp. 52-54, April 1990.

Smith, R. C. and Cazes, R., "Modularity in Robotics - Technical Aspects and Applications", IFS Proceedings of International Conference on Robots in the Automation Industry (U.K), pp. 115-122, 1982.

Spar Corporation, "Conceptual Design of the Payload Handling Manipulator System," for NASA Jonhson Space Centre, Houston, TX., 1975.

Stonebraker, M., Wong.E., Kreps, P. and Held, G, "The Design and implementation of INGRES", ACM. Transaction on Database Systems. 1.3 pp. 189-222, Sept. 1976.

Stonebraker, M., Anderson, E., Hanson, E. and Rubenstein, B., "QUEL as a Datatype", Memo. UCB/ERL M83/73, University of California, Berkeley, Dec. 1983a.

Stonebraker, M., Rubenstein,B.and Guttman, A., "Application of Abstract Data Types and Abstract Indices to CAD Database", Proceedings of ACM SIGMOD Conference on Engineering Design Applications, San. Jose, Calif., ACM. New York, May 1983b.

Taguchi, G. and Wu, U., "Central Japan Quality Control Association," Nagoya, Japan, 1980.

Takano, M.,"A New Effective Solution for Inverse Kinematics Problem (Synthesis) of a Robot With Any Type of Configuration", Journal of the Faculty of Engineering, The University of Tokyo, Vol. 38, No. 2, pp. 107-135, 1985.

Tan, H.F. and Chang, F.Y.,"A Flexible Robot Programming System - URUCS", Proceedings of 15th International Symposium on Industrial Robots, Tokyo, Japan, pp. 725-732, Vol. 2, September 1985.

Tao, J.M., Luh, J.Y.S. and Zheng, Y.F., "Compliant Coordination Control of Two Moving Industrial Robots", IEEE Transactions on Robotics and Automation, Vol. 6, No 3, pp. 322-330, December 1990.

Tesar, D, and Butler, M. S., "A Generalized Modular Architecture for Robot Structures", ASME, Journal of Manufacturing Review, Vol. 2, No2, pp. 91-117, 1989.

Theveneau, P. and Pasquier, M., "A Geometric Modeller for An Automatic Robot Programming System", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, Il Ciocco, Italy, July 4-6, 1988.

Tourassis, V.D., "Principles and Design of Model-Based Controllers," International Journal of

Control, Vol. 47, No. 5, pp. 1267-1275, May 1988.

Tourassis, V. and Ang , Jr., M., "Analysis and Design of Robotic Manipulators With Multiple Interchangeable Wrists", IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, pp. 223-230, April 1989a.

Tourassis, V. and Ang, Jr., M,"A Modular Architecture for Inverse Robot Kinematics", IEEE Transactions on Robotics and Automation, Vol. 5, No. 5, pp. 555-568, October 1989b.

Turney, J.L., Mudge, T.N., and Lee, C.S.G., "Equivalence of Two Formulations for Robot Arm Dynamics," SEL Report 142 ECE Department, University of Michigan, Ann Arbor, Michigan, 1980.

Uicker, J.J., "On the Dynamic Analysis of Spatial Linkages Using 4*4 Matrices", Ph.D dissertation, Northwestern University, Evanston, Ill. 1965.

Vaghul, M., Zinsmeister, G. E., Dixon J. R. and Simmon M.K., "Expert Systems in a CAD Environment: Injection Moulding Part Design as an Example" Proceedings 1985 ASME Conference, Computers in Engineering, Boston, August 1985.

Van Aken, L and Van Brussel, H.,"A Structured Geometric Database in An Off-line Robot Programming System", Robotica, Vol. 5, pp. 333-339, 1987.

Van Aken, L. and Van Brussel, H., "Robot Programming Languages: the Statement of a Problem", Robotica, Vol. 6, pp. 141-148, 1988.

Van Aken, L., Van Brussel, H. and Schutter, J.D., "Simplification of a Robot Task specification by Incorporating a Structured Geometric Database into an Off-line Robot Programming System", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, Proceedings of the NATO Advanced Research Workshop on CAD Based Programming for Sensory Robots, II Ciocco, Italy, pp.123-150, July 4-6, 1988.

Van Brussel, H., Winter, D.D., Valckenaers, P. and Claus, H., "A Universal Programming

Structure for Multi-Robot Assembly Systems", Proceedings of 8th International Conference on Assembly Automation, pp. 209-226, March 1987.

Volz, R.A., "Report of the Robot Programming Language Working Group: NATO Workshop on Robot Programming Languages," IEEE Journal of Robotics and Automation, Vol. 4, No. 1, pp. 86-90, February 1988.

Wagner, C.C., "The Implementations of Natural Language Structure on the Control of Intelligent Robotic Systems", International Journal of Computer Integrated Manufacturing, Vol. 3, No. 1, pp. 35-46, 1990.

Waldron, K.J., "A method of studying joint geometry Mechanism and Machine Theory" 7, 347-53 [1.4.3], 1972.

Walker, M. W., "Manipulator Kinematics and the Epsilon Algebra", IEEE Journal of Robotics and Automation, Vol. 4, No. 2, pp. 186-192, 1988.

Weck, M. and Clemens, R., "Experiences with Off-Line Robot Programming via Standardized Interfaces", CAD Based Programming for Sensory Robots, Edited by Ravani, B., NATO ASI Series F, Vol. 50, pp. 223-234, 1988.

Weston, R.H., Gascoine, J.D., Rui, A., Hodgson, A., Sumpter, C.M. and Coutts, I., "Steps towards information integration in Manufacturing", International Journal Computer Integrated Manufacturing, Vol. 3, pp. 140-153, 1988.

Weston, R.H., Harrison, R., Booth, A.H. and Moore, P.R., "A New Approach to Machine Control," Computer-Aided Engineering Journal, pp. 27-32, February 1989a.

Weston, R.H., Harrison, R.,Booth, A.H., and Moore, P.R., "Universal Machine Control System Primitives for Modular Distributed Manipulator Systems," International Journal of Production Research,Vol. 27, No.3, pp. 393-410, 1989b.

Weston, R.H., Gascoigne, J.D., Coutts, I., "The Need for a Generic Framework for System

Integration," in "Advanced Information Technologies for Materials Flow Systems", NATO ASI Series F53, pp. 279-309, 1989c.

Weston, R.H., Hodgson, A., Coutts, I., Murgatroyd, S., "Integration Tools Based on OSI Networks", AUTOFACT, 89, Detroit, Michigan, SME Ref. Ms89-708, October 1989d.

Weston, R.H., "New Concept in Programmable Automation", Proceedings Institute of Mechanical Enginerring Seminar on "Control in the Process Industries", Solihull, U.K. May 1990.

Weston, R.H. and Davies, B.J., "Boilding Wider Scope Integrated Manufacturing Systems", Tutorial for Advanced School on 'Mechtronics', Udine, Italy, 23-27 March 1992.

Woodwark, J., "Shape Models in Computer Integrated Manufacture - a Review," Computer-Aided Engineering Journal, pp. 103-113, June 1988.

Wozniak, A. and Warczynski, J., "Robot Simulation and Programming System", IFAC Robot Control, Karlsruhe, pp. 437-442, 1988.

Wurst, K. H., "The Concepts and Construction of a Modular Robot System", IFS, Proceeding International Symposium on Industrial Robotics (Belgium), pp. 37-44, 1986.

Yan, X. T., Weston, R.H. and Case, K., "An Emulation System for Machine Design and Control", The Proceedings of 11th International Conference on Assembly Automation, Detroit, Michigan, pp. MS90-840-1-MS90-838-18, November 11-14, 1990.

Yoffa, N. A., "Off-Line Programming for Automotive Spot Welding," Robotics World, pp. 24-25, April, 1988.

Yong, Y.F., Marshall, R.J. and Bonney, M. C., "Using CAD to Plan and Evaluate a Robotic Cell", in Control and Programming in Advanced Manufacturing ". Edited by Rathmill, K., Published by IFS Ltd, UK, Springer-Verlag, pp. 235-247, 1988.

Yoshimara, M., Yoshikawa, N. and Hitomi, K., "Design Optimization of Industrial Robots

Considering the Working Environment", International Journal of Production Research, Vol. 28, No. 5, pp. 805-820, 1990.

Zaniola, C., "The Database Language GEM", Proceedings of the International Conference on Management of Data, San. Jose, California, ACM, New York, pp. 207-218, May 23-26, 1983.

Zdonik, S. B. and Wegner, P., "Language and methodology for object-oriented database environments", Proceedings of 19th Hawaii International Conference on System Sciences, Honolulu, HI, USA, pp. 378-387, January 1986.

Zienkiewicz, O. C., "The Finite Element Method", London,: MCGraw-Hill, 1977.

Appendix A Contemporary robot simulation systems

In this appendix a brief description of the features of contemporary robot simulation systems is given to illustrate their variety.

<u>CimStation:</u> was developed by SILMA Inc. California, USA [SILMA 1989]. The CimStation is a CAD based system and offers facilities to import CAD models from external CAD systems, via the IGES (Initial Graphical Exchange Specification) interface or direct translators from certain CAD systems. It has capabilities for 3D CAD modelling with shading techniques, collision detection among objects, hierarchical world model structure, attachment of kinematics to structured objects and models some dynamic properties. A special robot programming language called SIL [SILMA 1988] was developed to strengthen its programming capability. Based on the use of its collision detection capabilities the operation of force sensors, limit switches and light-beam interrupt sensors can be simulated. The robot language KAREL is supported in post-processing [Craig 1988].

<u>ROSI:</u> ROSI was developed at the University of Karlsruhe. It is built on a centralized database architecture to support CAD and robotic simulation in a uniform manner [Dillmann and Huck 1986].

<u>SHARP:</u> SHARP is an automatic robot programming system under current development at the LIFIA laboratory in France. It has 3D modelling and motion planning capabilities. Two classes of functional reasoning are available in the system, namely functions aimed at computing collision free trajectories for a robot, and functions to generate contact guided motions under uncertainty constraints, i.e. motion in part-mating operations etc. [Laugier 1988].

<u>AutoSimulation Suite:</u> The AutoSimulation suite developed by AutoSimulation Inc. (ASI), is composed of: AutoBots for robot simulation and off-line programming;

AutoMod for numerical simulation; AutoGram for producing graphic representation; and InterFaSE modules for factory scheduler based simulation. [Robotics World 1986, Miller 1987]

GRASP: GRASP (General Robot Arm Simulation Program), was developed at the Rensselaer Polytechnic Institute in the United States. It has the capabilities of modelling up to six axis robots, describing tasks, cycle time estimation, and robot animation to facilitate evaluation. The VAL robot language is supported in post-processing [Chan 1989 and Derby 1984].

McDonnell Douglas Robotics Suite: The McDonnell Douglas robotic simulation system was designed for off-line robot programming [Howie 1984] and has four modules namely: BUILD for building robot models (with up to six degree of freedom manipulators); PLACE for designing and evaluating a robot workcell through animation; COMMAND for off-line program creation; and ADJUST for the calibration of errors between the built CAD model and physical robot workcell. Robot languages, VAL, VAL-II, KAREL and MCL for Unimation GMF and Cincinnati T3 are supported to enable off-line programming within the COMMAND module [Chan 1989].

ROBOGRAPHIX: The ROBOGRAPHIX simulation system was developed by Computervision [Mattis and Gill 1988]. Four major functions are covered in the system, which are workcell modelling, robot program creation, robot program verification, and post-processing and down loading to the target physical robots. CAD/CAM information can be used for robot simulation as the CAD/CAM system database and the ROBOGRAPHIX simulation system and its accessories library are integrated. Robot languages, such as VAL and RAIL are supported in post-processing.

GRASP: GRASP was developed at the University of Nottingham and further details can be found in Section 3.5.2.

ROBOT-SIM: ROBOT-SIM was developed by Calma R & D and offers the user with more than 20 popular industrial robot models [Miller 1987]. Providing many common robot simulation features (i.e. robot and workcell design, robot motion programming, cycle time estimation) ROBOT-SIM additionally provides capabilities for dynamic modelling and simulation of up to six degree of freedom robots generating information relating to velocities, accelerations, link inertias and motor torque characteristics.

<u>ROSI:</u> ROSI (RObot dynamic SImulator) was developed in the Department of Artificial Intelligence at the University of Edinburgh. The system claims to have no limitation on the number of degrees of freedom of a model. A dynamic's engine is included in ROSI for dynamic computation within an even larger software system. A user interface for communication between the dynamic engine and the program is also provided in graphical form. [Industrial Robot 1987]

<u>STAR:</u> STAR (or Simulation Tool for Automation and Robotics) was developed as an off-line robot motion planning and programming system with capabilities for dynamic modelling. A high level programming language is provided and therefore it has the abstraction of task description at high level. Common robot simulation facilities (e.g. input module for model building, kinematic modelling, trajectory planner and motion planing and programming) are provided. The animation of solid geometry models of objects generated on the GMOS solid modeller system is achieved through a CAD interface modeller system, although STAR is not a CAD package. [Hornick and Ravani 1986, Ravani and Hornick 1988]

<u>CATIA:</u> CATIA is the acronym for Computer Aided Design with a Three dimensional Interactive Application and was developed by Dassault Systems in France [Crosnier and Fournier 1987, Forestier 1985]. CATIA has capabilities for the 3D geometric modelling of complex kinematic closed loops. It can simulate cooperation between several robots, task definition in various forms and off-line robot programming.

<u>IGRIP:</u> IGRIP is the acronym of Interactive Graphics Robot Instruction Program and was

developed by Deneb Robotics Inc. It provides a software tool for workcell design, off-line programming, robot performance evaluation, collision detection and robot task description. It can also display the maximum and minimum reachable workplace of a robot. Elements of the workcell can be displayed in a choice of four forms [Yoffa 1988], viz: wire-frame, hidden line, simple shading, and sophisticated shading. IGRIP is computer system independent, being designed in modular form to provide portability and flexibility [Schreiber 1984, and Harrison and Mahajan 1986].

<u>HERON:</u> HERON was developed by Robcad Ltd., ISRAEL and is a stand-alone CAD/CAM workstation [Miller 1987]. It comprises six modules, namely:
ROBOSIM for workcell design, task description and simulation;
ROBOLOAD for downloading off-line robot programs;
ROBOGEO for geometric modelling and mechanism design;
ROBODOC for drafting and documentation;
ROBOPERT for providing project management tools;
ROBOLIB which provides libraries of available robots, accessories and peripherals.

<u>INTERGRAPH:</u> Intergraph robot simulation software was originally developed in conjunction with GMF Robotics. The system provides the tools to build libraries of end effectors and other peripherals for use with standard robot libraries. The system can define and simulate robot motion and perform production cost analysis. The off-line programming of a robot is further detailed in five phases [Kacala 1985] namely, operation planning and definition for robots and accessories; workplace composition; process simulation, editing and verification; program output for translating into a robot program and finally; process feedback and workcell calibration.

<u>GMF:</u> The GMF off-line programming system has a set of S and G codes to construct robot programs. English-like mnemonics may also be used as an alternative to indirectly specify the S and G codes. The data input for a point is required to comply with the joint axis form which makes programming difficult [Jacobs 1984].

<u>ROBCAM:</u> ROBCAM is a robot simulation and off-line programming system and was developed by Silma Corporation. It provides high-level robot programming capability based on the language RISE. The system requires a user to employ only one common language for all robots. The RISE program so created can be translated into a particular robot language after conversion through an intermediate language stage called RCODE. [Miller 1987, Craig 1985]

<u>WRAPS:</u> WRAPS stands for Welling Robot Adaptive off-line Programming and was developed in the Department of Manufacturing Engineering at Loughborough University of Technology [Goh and Middle 1985]. Being specialised for the off-line programming of robotic arc welding tasks, the system features functions such as procedure selection and optimization for welding via the use of an expert system [Middle and Goh 1987].

Appendix B The derivation of path control algorithms for two degree of freedom configurations

This appendix describes the derivation of forward and inverse kinematic solutions to articulated configurations with two degrees of freedom. In addition to the two situations discussed in chapter 4 the remaining five configurations depicted in Figure 4.5 are considered here.

B.1 One prismatic and one revolute axis articulated in the Z and Y directions respectively

For the case of (3) in Figure 4.5, the coordinate frame assignment of one prismatic and a revolute axes articulated in the Z and Y directions is shown in Figure A3.1.3. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	d_1	0	−90	0	$d_1+D_1+D_2$	0	−1
2	θ_2	$90+\theta_2$	0	D_4	D_3	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{0} to frame{2}are respectively

$$T_{31}(0,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_1+D_1+D_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{32}(1,2) = \begin{bmatrix} -\sin\theta_2 & -\cos\theta_2 & 0 & D_4\sin\theta_2 \\ \cos\theta_2 & -\sin\theta_2 & 0 & -D_4\cos\theta_2 \\ 0 & 0 & 1 & D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} T_3(0,2) &= T_{31}(0,1)\times T_{32}(1,2) \\ &= \begin{bmatrix} -\sin\theta_2 & -\cos\theta_2 & 0 & D_4\sin\theta_2 \\ 0 & 0 & 1 & D_3 \\ -\cos\theta_2 & \sin\theta_2 & 0 & D_4\cos\theta_2+d_1+D_1+D_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The inverse kinematic solutions for the above configuration can be analytically derived as follows based on the forward solutions:

$$\text{from}\quad \begin{aligned} P_{3x} &= D_4\sin\theta_2 \\ P_{3y} &= D_3 \\ P_{3z} &= D_4\cos\theta_2 + d_1 + D_1 + D_2 \end{aligned} \qquad \text{hence} \qquad \begin{aligned} \theta_2 &= \pm\text{asin}\frac{P_{3x}}{D_4} \\ d_1 &= P_{3z} - D_1 - D_2 - D_4 \end{aligned}$$

Clearly two solutions exist for a given position in manipulator cartesian space.

B.2 One axis revolute in Z and one axes prismatic in X direction

For the case of (4) in Figure 4.5, the coordinate frame assignment of one revolute and a prismatic axes articulated in the Z and X directions is shown in Figure A3.1.4. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	$90+\theta_1$	90	D_2	D_1	0	1
2	d_2	0	0	0	d_2+D_3	1	0

The transformation matrices from frame{0} to frame{1} can not be derived from the D-H representation based on the above four parameters in the table since there are six variables required to describe the transformation from frame {0} to {1}. Homogeneous transformation matrices were used to obtain the transformation as follows.

$$T_{41}(0,1) = Rot(Z,\theta_1)\,Trans(Z,D_1)\,Trans(X,D_2)\,Rot(Z,90)\,Rot(X,90)$$

$$= \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & D_2\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & D_2\sin\theta_1 \\ 0 & 1 & 0 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation matrices from frame{1} to frame{2}, and frame{0} to frame{2}are respectively

$$T_{42}(1,2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 + D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4(0,2) = T_{41}(0,1) \times T_{42}(1,2) = \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & (d_2+D_2+D_3)\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & (d_2+D_2+D_3)\sin\theta_1 \\ 0 & 1 & 0 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse kinematic solutions for the above configuration can be analytically derived as follows based on the forward solutions:

$$\text{from} \quad \begin{aligned} P_{4x} &= (d_2+D_2+D_3)\cos\theta_1 \\ P_{4y} &= (d_2+D_2+D_3)\sin\theta_1 \\ P_{4z} &= D_1 \end{aligned} \qquad \text{hence} \quad \begin{aligned} \theta_1 &= \text{atan2}\frac{P_{4y}}{P_{4x}} \\ d_2 &= \frac{P_{4y}}{\sin\theta_1} - D_2 - D_3 \end{aligned}$$

Clearly only one solution exists for a given position in manipulator cartesian space.

B.3 Two revolute axes articulated in the Z and Z directions

For the case of (5) in Figure 4.5, the coordinate frame assignment of two revolute axes articulated in the Z and Z directions is shown in Figure A3.1.5. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	0	D_2	D_1	1	0
2	θ_2	θ_2	0	D_3	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{0} to frame{2}are respectively

$$T_{51}(0,1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & D_2\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{52}(1,2) = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & D_3\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & D_3\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5(0,2) = T_{51}(0,1) \times T_{52}(1,2)$$

$$= \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & D_3\cos(\theta_1+\theta_2)+D_2\cos\theta_1 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & D_3\sin(\theta_1+\theta_2)+D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse kinematic solutions for the above configuration can be analytically derived as follows based on the forward solutions:

$$P_{5x} = D_3\cos(\theta_1+\theta_2)+D_2\cos\theta_1$$

from $P_{5y} = D_3\sin(\theta_1+\theta_2)+D_2\sin\theta_1$

$$P_{5z} = D_1$$

hence

$$\theta_1 = \operatorname{atan2}\left(\frac{P_{5x}\left(CP_{5y} \pm P_{5x}\sqrt{P_{5x}^2+P_{5y}^2-C^2}\right)}{C(P_{5x}^2+P_{5y}^2) - CP_{5y} \mp P_{5x}\sqrt{P_{5x}^2+P_{5y}^2-C^2}}\right)$$

$$\theta_2 = \operatorname{acos}(P_{5x}-D_2\cos\theta_1)/D_3-\theta_1$$

where $C = (D_3^2-D_2^2-P_{5x}^2-P_{5y}^2)/(2D_2)$

Only two valid solutions exist for a given position in manipulator cartesian space.

B.4 Two revolute axes articulated in the Z and X directions

For the case of (6) in Figure 4.5, the coordinate frame assignment of two revolute axes articulated in the Z and X directions is shown in Figure A3.1.6. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	90	D_2	D_1	0	1
2	θ_2	θ_2+90	0	D_3	0	1	0

The transformation matrices from frame{0} to frame{1} can not be derived from the D-H representation based on the above four parameters in the table since there are six variables required to describe the transformation from frame {0} to {1}. Homogeneous transformation matrices were used to obtain the transformation as follows.

$$T_{61}(0,1) = Rot(Z,\theta_1)Trans(Z,D_1)Trans(X,D_2)Rot(Z,90)Rot(X,90)$$

$$= \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & D_2\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & D_2\sin\theta_1 \\ 0 & 1 & 0 & D_1 \end{bmatrix}$$

The transformation matrices from frame{1} to frame{2}, and frame{0} to frame{2}are respectively

$$T_{62}(1,2) = \begin{bmatrix} -\sin\theta_2 & -\cos\theta_2 & 0 & -D_3\sin\theta_2 \\ \cos\theta_2 & -\sin\theta_2 & 0 & D_3\cos\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6(0,2) = T_{61}(0,1) \times T_{62}(1,2)$$

$$= \begin{bmatrix} \sin\theta_1\sin\theta_2 & \sin\theta_1\cos\theta_2 & \cos\theta_1 & D_3\sin\theta_1\sin\theta_2 + D_2\cos\theta_1 \\ -\sin\theta_2\cos\theta_1 & -\cos\theta_1\cos\theta_2 & \sin\theta_1 & -D_3\sin\theta_2\cos\theta_1 + D_2\sin\theta_1 \\ \cos\theta_2 & -\sin\theta_2 & 0 & D_3\cos\theta_2 + D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse kinematic solutions for the above configuration can be analytically derived as follows based on the forward solutions:

$$P_{6x} = D_3\sin\theta_1\sin\theta_2 + D_2\cos\theta_1$$

from $$P_{6y} = -D_3\sin\theta_2\cos\theta_1 + D_2\sin\theta_1$$

$$P_{6z} = D_3\cos\theta_2 + D_1$$

$$\theta_2 = \pm \mathrm{acos}\,(P_{6z} - D_1)/D_3$$

hence

$$\theta_1 = \mathrm{asin}\left(P_{6x}D_3\sin\theta_2 \pm D_2\sqrt{D_3^2\sin^2\theta_2 - P_{6x}^2 + D_2^2}\right) / (D_3^2\sin^2\theta_2 + D_2^2)$$

Clearly only one solution exists for a given position in manipulator cartesian space.

B.5 Two revolute axes articulated in the Z and Y directions

For the case of (7) in Figure 4.5, the coordinate frame assignment of two revolute axes articulated in the Z and Y directions is shown in Figure A3.1.7. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	-90	D_2	D_1	0	-1
2	θ_2	θ_2-90	0	D_3	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{0} to frame{2}are respectively

$$T_{71}(0,1)=\begin{bmatrix}\cos\theta_1 & 0 & -\sin\theta_1 & D_2\cos\theta_1\\ \sin\theta_1 & 0 & \cos\theta_1 & D_2\sin\theta_1\\ 0 & -1 & 0 & D_1\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad T_{72}(1,2)=\begin{bmatrix}\sin\theta_2 & \cos\theta_2 & 0 & D_3\sin\theta_2\\ -\cos\theta_2 & \sin\theta_2 & 0 & -D_3\cos\theta_2\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_7(0,2) = T_{71}(0,1)\times T_{72}(1,2)$$

$$= \; = \begin{bmatrix}\sin\theta_2\cos\theta_1 & \cos\theta_1\cos\theta_2 & -\sin\theta_1 & (D_3\sin\theta_2+D_2)\cos\theta_1\\ \sin\theta_1\sin\theta_2 & \sin\theta_1\cos\theta_2 & \cos\theta_1 & (D_3\sin\theta_2+D_2)\sin\theta_1\\ \cos\theta_2 & -\sin\theta_2 & 0 & D_3\cos\theta_2+D_1\\ 0 & 0 & 0 & 1\end{bmatrix}$$

The inverse kinematic solutions for the above configuration can be analytically derived as follows based on the forward solutions:

$$P_{7x} = (D_3\sin\theta_2+D_2)\cos\theta_1$$

from $P_{7y} = (D_3\sin\theta_2+D_2)\sin\theta_1$

$$P_{7z} = D_3\cos\theta_2+D_1$$

$$\theta_1 = \operatorname{atan2}\left(\frac{P_{7y}}{P_{7x}}\right)$$

hence

$$\theta_2 = \pm\operatorname{acos}\left(P_{7z} - D_1\right) / D_3$$

Clearly there are two solutions available for a given position in manipulator cartesian space.

Appendix C Control algorithms of 3 DOF configurations

This appendix describes the derivation of forward and inverse kinematic solutions to articulated configurations with two degrees of freedom. In addition to the two situations discussed in chapter 4 the remaining ten configurations depicted in Figure 4.6 are considered here.

C.1 One revolute and two prismatic axes articulated in the Z, Z and X directions

For the case of (3) in Figure 4.6, the coordinate frame assignment of one revolute and two prismatic axes articulated in the Z, Z, and X directions is shown in Figure C.3. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	0	D_2	D_1	1	0
2	d_2	90	90	0	d_2+D_3	0	1
3	d_3	0	0	0	d_3+D_4	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{3}are respectively

$$T_{31}(0,1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & D_2\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{32}(1,2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_2+D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{33}(2,3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3+D_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3(0,3) = T_{31}(0,1) \times T_{32}(1,2) \times T_{33}(2,3)$$

$$= \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & (d_3+D_4)\cos\theta_1 + D_2\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & (d_3+D_4)\sin\theta_1 + D_2\sin\theta_1 \\ 0 & 1 & 0 & d_2+D_3+D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

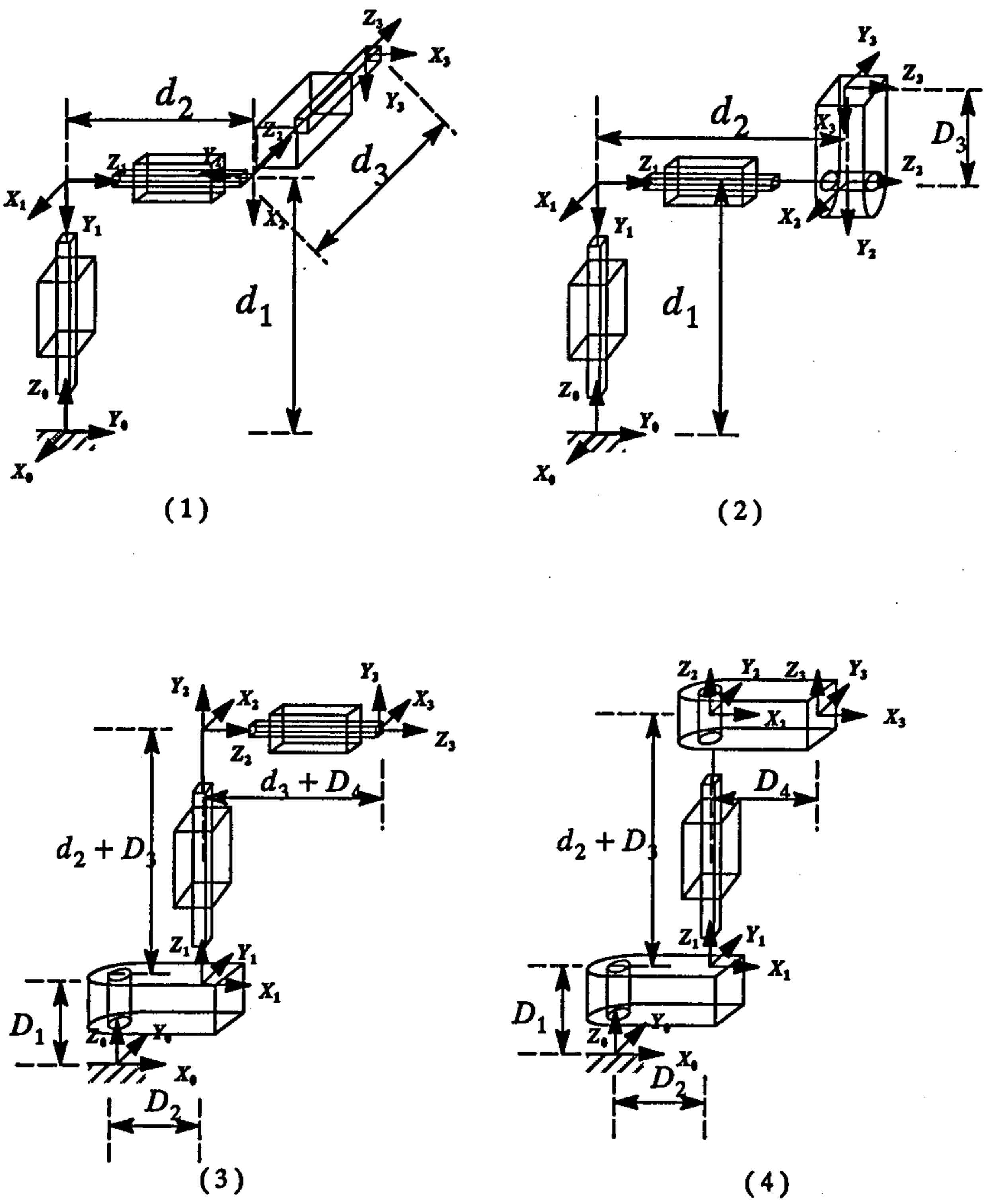


Figure C The assignment of coordinate systems to twelve possible three joint configurations (to be continued)

C.2 Two revolute axes along Z directions connected by one prismatic axis in between

For the case of (4) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.4. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	0	D_2	D_1	1	0
2	d_2	0	0	0	d_2+D_3	1	0
3	θ_3	θ_3	0	D_4	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{5}are respectively

$$T_{41}(0,1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & D_2\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

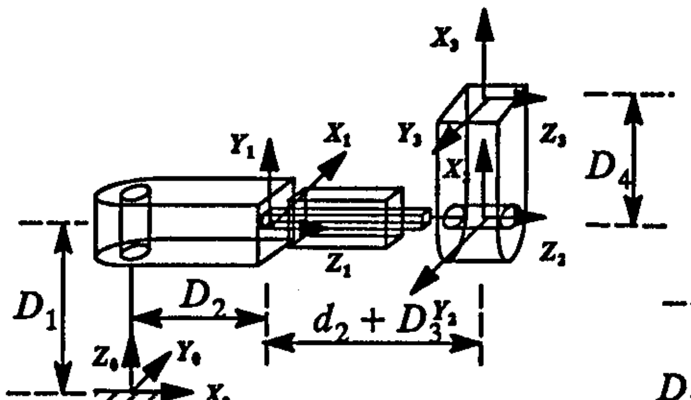
$$T_{42}(1,2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 + \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{43}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_4\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4(0,3) = T_{41}(0,1) \times T_{42}(1,2) \times T_{43}(2,3)$$

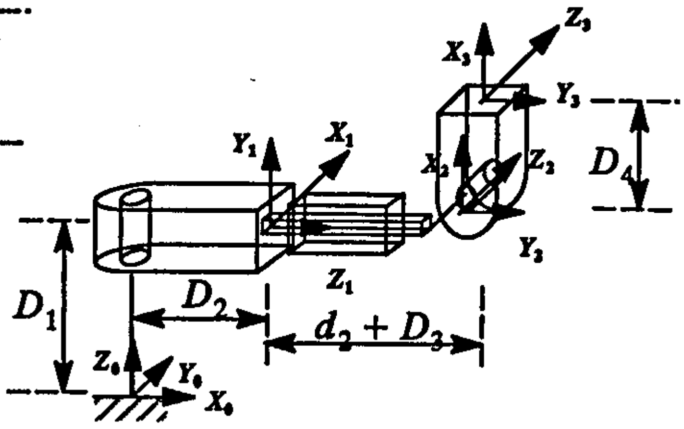
$$= \begin{bmatrix} \cos(\theta_1+\theta_3) & -\sin(\theta_1+\theta_3) & 0 & D_4\cos(\theta_1+\theta_3) + D_2\cos\theta_1 \\ \sin(\theta_1+\theta_3) & \cos(\theta_1+\theta_3) & 0 & D_4\sin(\theta_1+\theta_3) + D_2\sin\theta_1 \\ 0 & 1 & 1 & d_2 + D_3 + D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

C.3 Two revolute axes along Z and X directions connected by one prismatic axis in between

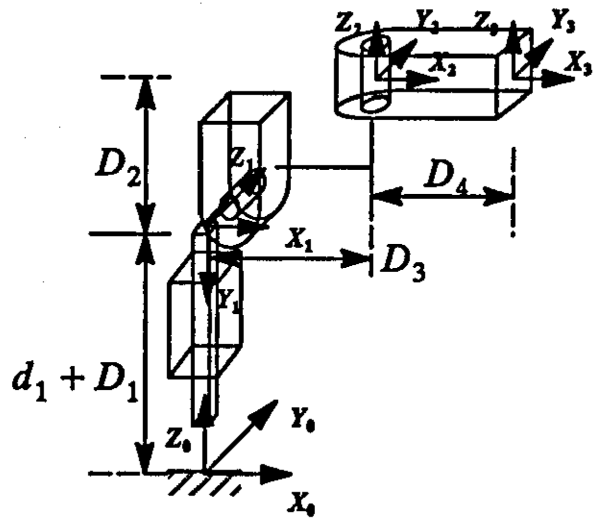
For the case of (5) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.5. The link parameter table required to use the D-H matrix is then



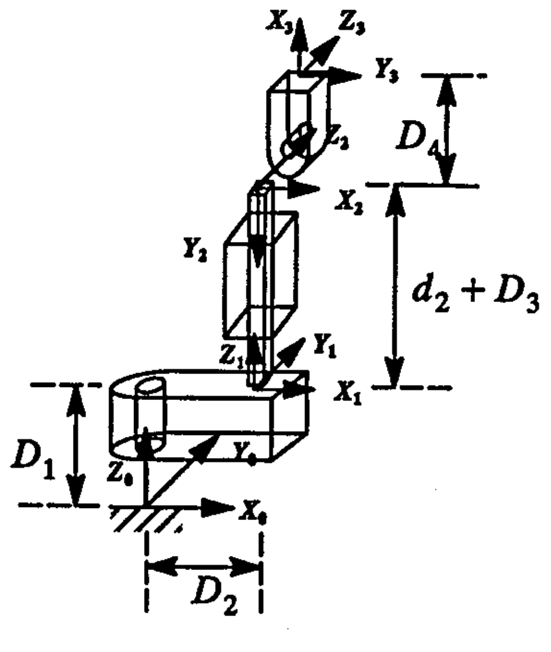
(5)



(6)



(7)



(8)

Figure C (continued) The assignment of coordinate systems to twelve possible three joint configurations (to be continued)

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	$90+\theta_1$	90	D_2	D_1	0	1
2	d_2	90	0	0	d_2+D_3	1	0
3	θ_3	θ_3	0	D_4	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{5} are respectively

$$T_{51}(0,1) = \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & D_2\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & D_2\sin\theta_1 \\ 0 & 1 & 0 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{52}(1,2) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_2+i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{53}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_4\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5(0,3) = T_{51}(0,1) \times T_{52}(1,2) \times T_{53}(2,3)$$

$$= \begin{bmatrix} \sin\theta_1\sin\theta_3 & \sin\theta_1\cos\theta_3 & \cos\theta_1 & D_4\sin\theta_1\sin\theta_3 + (d_2+D_3+D_2)\cos\theta_1 \\ -\sin\theta_3\cos\theta_1 & -\cos\theta_1\cos\theta_3 & \sin\theta_1 & -D_4\sin\theta_3\cos\theta_1 + (d_2+D_3+D_2)\sin\theta_1 \\ \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 + D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

C.4 Two revolute axes along Z and Y directions connected by one prismatic axis in between along X direction

For the case of (6) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.6. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	$90+\theta_1$	90	D_2	D_1	0	1
2	d_2	90	90	0	d_2+D_3	0	1
3	θ_3	θ_3	0	D_4	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{3} are respectively

$$T_{61}(0,1) = \begin{bmatrix} -\sin\theta_1 & 0 & \cos\theta_1 & D_2\cos\theta_1 \\ \cos\theta_1 & 0 & \sin\theta_1 & D_2\sin\theta_1 \\ 0 & 1 & 0 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{62}(1,2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_2+l \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{63}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_4\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6(0,3) = T_{61}(0,1) \times T_{62}(1,2) \times T_{63}(2,3)$$

$$= \begin{bmatrix} \sin\theta_3\cos\theta_1 & \cos\theta_1\cos\theta_3 & -\sin\theta_1 & D_4\sin\theta_3\cos\theta_1 + (D_2+d_2+D_3)\cos\theta_1 \\ \sin\theta_1\sin\theta_3 & \sin\theta_1\cos\theta_3 & \cos\theta_1 & D_4\sin\theta_1\sin\theta_3 + (D_2+d_2+D_3)\sin\theta_1 \\ \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 + D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

C.5 One prismatic axis in Z direction followed by two revolute axes along Y and Z directions

For the case of (7) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.7. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	d_1	0	–90	0	d_1	0	–1
2	θ_2	θ_2	90	D_3	0	0	1
3	θ_3	θ_3	0	D_4	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{2} to frame{3}based on D-H representation are respectively

$$T_{71}(0,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_1+D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{72}(1,2) = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & D_3\cos\theta_2 \\ \sin\theta_2 & 0 & -\cos\theta_2 & D_3\sin\theta_2 \\ 0 & 1 & 0 & D_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{73}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_4\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since there are six parameters required to describe the transformation of six degrees of freedom from joint 2 to 3, i.e. link 2 and 3 do not satisfy the D-H conventions, the D-H representation in the case that there is no intersection between Z_{i-1} and X_i can not be directly used. The author derived the transformation representation from frame{1} to frame{2} based on six degrees of freedom transformation as follows

$$T_{72}(1,2) = Rot(z,\theta_2)\,Rot(X,90)\,Trans(X,D_3)\,Tans(Z,D_2)$$

$$= \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & D_2\sin\theta_2+D_3\cos\theta_2 \\ \sin\theta_2 & 0 & -\cos\theta_2 & -D_2\cos\theta_2+D_3\sin\theta_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore the transformation from frame{0} to frame{3} is

$$T_7(0,3) = T_{71}(0,1) \times T_{72}(1,2) \times T_{73}(2,3)$$

$$= \begin{bmatrix} \cos\theta_2\cos\theta_3 & -\sin\theta_3\cos\theta_2 & \sin\theta_2 & P_{7x} \\ \sin\theta_3 & \cos\theta_3 & 0 & P_{7y} \\ -\sin\theta_2\cos\theta_3 & \sin\theta_2\sin\theta_3 & \cos\theta_2 & P_{7z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$P_{7x} = D_4\cos\theta_2\cos\theta_3 + D_2\sin\theta_2 + D_3\cos\theta_2$$

$$P_{7y} = D_4\sin\theta_3$$

$$P_{7z} = -\,(D_4\cos\theta_3 + D_3)\sin\theta_2 - D_2\cos\theta_2 + d_1 + D_1$$

C.6 Two revolute axes along Z and Y directions connected by one prismatic axis in between along Z direction

For the case of (8) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.8. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	0	D_2	D_1	1	0
2	d_2	0	–90	0	d_2+D_3	0	–1
3	θ_3	$270+\theta_3$	0	D_4	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{3} are respectively

$$T_{81}(0,1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & D_2\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{82}(1,2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2+D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{83}(2,3) = \begin{bmatrix} \sin\theta_3 & \cos\theta_3 & 0 & D_4\sin\theta_3 \\ -\cos\theta_3 & \sin\theta_3 & 0 & -D_4\cos\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_8(0,3) = T_{81}(0,1) \times T_{82}(1,2) \times T_{83}(2,3)$$

$$= \begin{bmatrix} \sin\theta_3\cos\theta_1 & \cos\theta_1\cos\theta_3 & -\sin\theta_1 & D_4\sin\theta_3\cos\theta_1 + D_2\cos\theta_1 \\ \sin\theta_1\sin\theta_3 & \sin\theta_1\cos\theta_3 & \cos\theta_1 & D_4\sin\theta_1\sin\theta_3 + D_2\sin\theta_1 \\ \cos\theta_3 & -\sin\theta_3 & 0 & D_4\cos\theta_3 + d_2 + D_3 + D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

C.7 One prismatic axis in Z direction followed by two revolute axes along Y and Z directions

For the case of (9) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.9. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	–90	D_2	D_1+D_3	0	–1
2	θ_2	θ_2–90	–90	D_5	D_4	0	–1
3	d_3	0	0	0	d_3	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{2} to frame{3} based on D-H representation are respectively

$$T_{91}(0,1) = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & D_2\cos\theta_1 \\ \sin\theta_1 & 0 & \cos\theta_1 & D_2\sin\theta_1 \\ 0 & -1 & 0 & D_1+D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

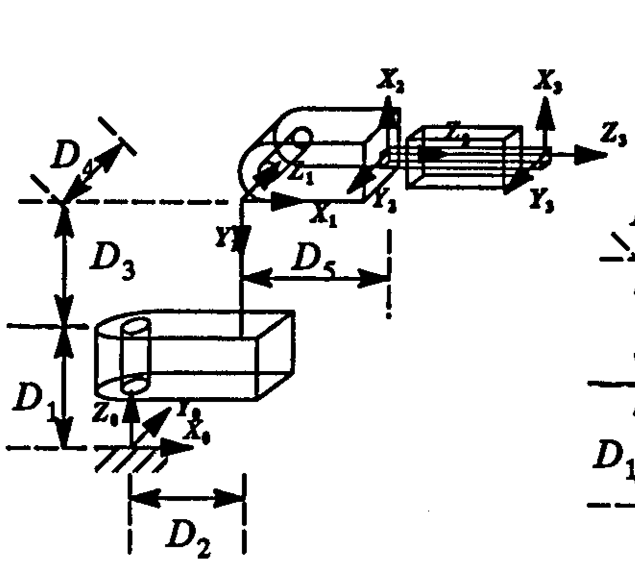
$$T_{92}(1,2) = \begin{bmatrix} \sin\theta_2 & 0 & \cos\theta_2 & D_5\sin\theta_2 \\ -\cos\theta_2 & 0 & \sin\theta_2 & -D_5\cos\theta_2 \\ 0 & -1 & 0 & D_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{93}(2,3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since there are six parameters required to describe the transformation of six degrees of freedom from joint 2 to 3, i.e. link 2 and 3 do not satisfy the D-H conventions, the D-H representation in the case that there is no intersection between Z_{i-1} and X_i can not be directly used. The author derived the transformation representation from frame{1} to frame{2} based on six degrees of freedom transformation as follows

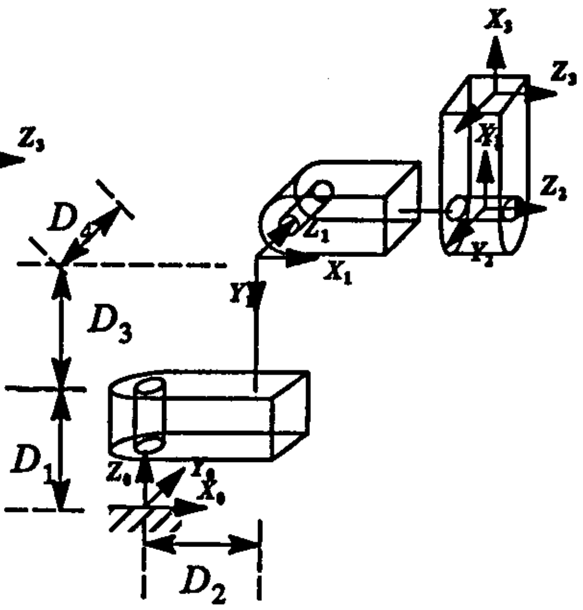
$$T_{92}(1,2) = Trans(Z, D_4)\,Trans(X, D_5)\,Rot(Z, \theta_2)\,Rot(Y, 90)\,Rot(Z, 90)$$

$$= \begin{bmatrix} \sin\theta_2 & 0 & \cos\theta_2 & D_5\cos\theta_2 \\ -\cos\theta_2 & 0 & \sin\theta_2 & D_5\sin\theta_2 \\ 0 & -1 & 0 & D_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

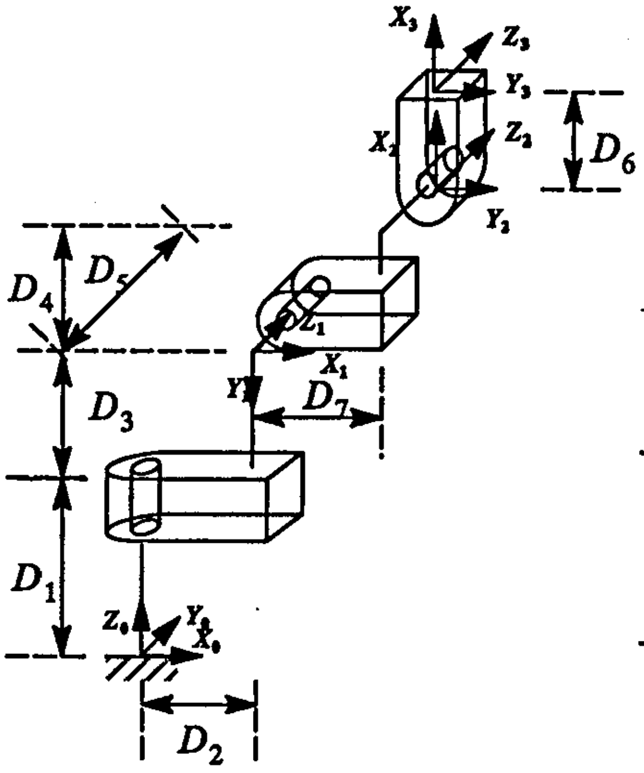
Therefore the transformation from frame{0} to frame{3} is



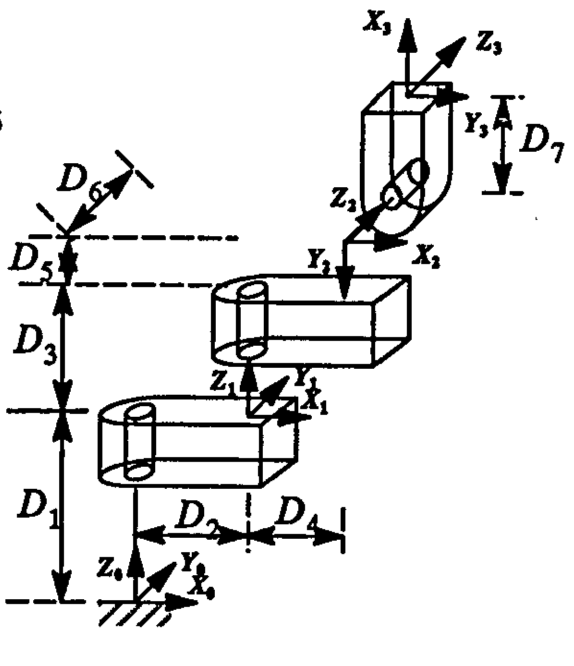
(9)



(10)



(11)



(12)

Figure C (continued) The assignment of coordinate systems to twelve possible three joint configurations

$$T_9(0,3) = T_{91}(0,1) \times T_{92}(1,2) \times T_{93}(2,3)$$

$$= \begin{bmatrix} \sin\theta_2\cos\theta_1 & \sin\theta_1 & \cos\theta_1\cos\theta_2 & P_{9x} \\ \sin\theta_1\sin\theta_2 & -\cos\theta_1 & \sin\theta_1\cos\theta_2 & P_{9y} \\ \cos\theta_2 & 0 & -\sin\theta_2 & P_{9z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$P_{9x} = (D_5\cos\theta_2 + D_2 + d_3\cos\theta_2)\cos\theta_1 - D_4\sin\theta_1$$

$$P_{9y} = (D_5\cos\theta_2 + D_2 + d_3\cos\theta_2)\sin\theta_1 + D_4\cos\theta_1$$

$$P_{9z} = D_1 + D_3 - (D_5 + d_3)\sin\theta_2$$

C.8 Three revolute axes along Z, Y and X directions in articulation

For the case of (10) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.10. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	–90	D_2	D_1+D_3	0	–1
2	θ_2	θ_2–90	–90	D_5	D_4	0	–1
3	θ_3	θ_3	0	D_6	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{3} are respectively

$$T_{101}(0,1) = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & D_2\cos\theta_1 \\ \sin\theta_1 & 0 & \cos\theta_1 & D_2\sin\theta_1 \\ 0 & -1 & 0 & D_1 + D_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{102}(1,2) = \begin{bmatrix} \sin\theta_2 & 0 & \cos\theta_2 & D_5\cos\theta_2 \\ -\cos\theta_2 & 0 & \sin\theta_2 & D_5\sin\theta_2 \\ 0 & -1 & 0 & D_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{103}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_6\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_6\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{10}(0,3) = T_{101}(0,1) \times T_{102}(1,2) \times T_{103}(2,3)$$

$$= \begin{bmatrix} C\theta_1 S\theta_2 C\theta_3 + S\theta_1 S\theta_3 & S\theta_1 C\theta_3 - C\theta_1 S\theta_2 S\theta_3 & C\theta_1 C\theta_2 & P_{10x} \\ S\theta_1 S\theta_2 C\theta_3 - C\theta_1 S\theta_3 & -S\theta_1 S\theta_2 S\theta_3 - C\theta_1 C\theta_3 & S\theta_1 C\theta_2 & P_{10y} \\ C\theta_2 C\theta_3 & -C\theta_2 S\theta_3 & -S\theta_2 & P_{10z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where "*S*" and "*C*" denotes *sin* and *cos* functions respectively, and

$$P_{10x} = (D_6\sin\theta_2\cos\theta_3 + D_5\cos\theta_2 + D_2)\cos\theta_1 + (D_6\sin\theta_3 - D_4)\sin\theta_1$$

$$P_{10y} = (D_6\sin\theta_2\cos\theta_3 + D_5\cos\theta_2 + D_2)\sin\theta_1 + (-D_6\sin\theta_3 + D_4)\cos\theta_1$$

$$P_{10z} = D_6\cos\theta_2\cos\theta_3 - D_5\sin\theta_2 + D_1 + D_3$$

C.9 Three revolute axes along Z, Y and Y directions in articulation

For the case of (11) in Figure 4.6, the coordinate frame assignment of three revolute axes articulated in the Z, Y, and Y directions is shown in Figure C.11. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	–90	D_2	D_1+D_3	0	–1
2	θ_2	θ_2–90	0	D_7	D_4	1	0
3	θ_3	θ_3	0	D_6	0	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, and frame{2} to frame{3} based on D-H representation are respectively

$$T_{112}(1,2) = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & D_7\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & -D_7\sin\theta_2 \\ 0 & 0 & 1 & D_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{113}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_6\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_6\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since there are six parameters required to describe the transformation of six degrees of freedom from joint 2 to 3, i.e. link 2 and 3 do not satisfy the D-H conventions, the D-H representation in the case that there is no intersection between Z_{i-1} and X_i can not be directly used. The author derived the transformation representation from frame{1} to frame{2} based on six degrees of freedom transformation as follows

$$T_{112}(1,2) = Rot(Z,\theta_2)\,Tra(Y,-D_8)\,Tra(Z,D_5)\,Tra(X,D_7)\,Rot(Z,-90)$$

$$= \begin{bmatrix} \sin\theta_2 & \cos\theta_2 & 0 & (D_8\sin\theta_2 + D_7\cos\theta_2) \\ -\cos\theta_2 & \sin\theta_2 & 0 & (D_7\sin\theta_2 - D_8\cos\theta_2) \\ 0 & 0 & 1 & D_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore the transformation from frame{0} to frame{3} is

$$T_{11}(0,3) = T_{111}(0,1) \times T_{112}(1,2) \times T_{113}(2,3)$$

$$= \begin{bmatrix} \cos\theta_1\sin(\theta_2+\theta_3) & \cos\theta_1\cos(\theta_2+\theta_3) & -\sin\theta_1 & P_{11x} \\ \sin\theta_1\sin(\theta_2+\theta_3) & \sin\theta_1\cos(\theta_2+\theta_3) & \cos\theta_1 & P_{11y} \\ \cos(\theta_2+\theta_3) & -\sin(\theta_2+\theta_3) & 0 & P_{11z} \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

where

$$P_{11x} = D_6\cos\theta_1\sin(\theta_2+\theta_3) + (D_8\sin\theta_2 + D_7\cos\theta_2)\cos\theta_1 - D_5\sin\theta_1 + D_2\cos\theta_1 \qquad (1)$$

$$P_{11y} = D_6\sin\theta_1\sin(\theta_2+\theta_3) + (D_8\sin\theta_2 + D_7\cos\theta_2)\sin\theta_1 + D_5\cos\theta_1 + D_2\sin\theta_1 \qquad (2)$$

$$P_{11z} = D_6\cos(\theta_2+\theta_3) + D_8\cos\theta_2 - D_7\sin\theta_2 + D_1 + D_3 \qquad (3)$$

From the above equations inverse solutions for the configuration 11 of three articulated axis group can be obtained as follows. Divide P_{1y} by $D_6\sin\theta_1$ and divide P_{1x} by $D_6\cos\theta_1$ then

$$\frac{P_{1x} - ((D_8\sin\theta_2 + D_7\cos\theta_2)\cos\theta_1 - D_5\sin\theta_1 + D_2\cos\theta_1)}{D_6\cos\theta_1} = \sin(\theta_2 + \theta_3) \tag{4}$$

$$\frac{P_{1y} - ((D_8\sin\theta_2 + D_7\cos\theta_2)\sin\theta_1 + D_5\cos\theta_1 + D_2\sin\theta_1)}{D_6\sin\theta_1} = \sin(\theta_2 + \theta_3) \tag{5}$$

Divide (4) by (5) then

$$P_{1x}\sin\theta_1 + D_5 - P_{1y}\cos\theta_1 = 0 \tag{6}$$

Substitute $\cos\theta_1$ with $\cos^2\alpha = 1 - \sin^2\alpha$ in (6), move $P_{1y}\cos\theta_1$ to the right side of the equation and square both sides of the equation

$$(P_{1x}^2 + P_{1y}^2)\sin^2\theta_1 + 2P_{1x}D_5\sin\theta_1 + D_5^2 - P_{1y}^2 = 0 \tag{7}$$

Therefore the roots for the above equation is

$$\sin\theta_1 = \frac{-P_{1x}D_5 \pm \sqrt{P_{1x}^2P_{1y}^2 - P_{1y}^2D_5^2 + P_{1y}^4}}{P_{1x}^2 + P_{1y}^2} \tag{8}$$

From (8) θ_1 can be expressed as

$$\theta_1 = \pm\text{asin}\theta_1 = \pm\frac{-P_{1x}D_5 \pm \sqrt{P_{1x}^2P_{1y}^2 - P_{1y}^2D_5^2 + P_{1y}^4}}{P_{1x}^2 + P_{1y}^2} \tag{9}$$

From P_{1y} and P_{1z} equations the following equivalent forms can be derived

$$\frac{P_{1y} - ((D_8\sin\theta_2 + D_7\cos\theta_2)\sin\theta_1 + D_5\cos\theta_1 + D_2\sin\theta_1)}{D_6\sin\theta_1} = \sin(\theta_2 + \theta_3) \tag{10}$$

$$\frac{P_{1z} + (D_7\sin\theta_2 - D_8\cos\theta_2) - (D_1 + D_3)}{D_6} = \cos(\theta_2 + \theta_3) \tag{11}$$

Square (10) and (11) and add both squared equations together by using $\cos^2\alpha = 1 - \sin^2\alpha$ then

$$A\sin\theta_2+B\cos\theta_2+C = 0 \tag{12}$$

where

$$A= 2\,(D_8D_5\cos\theta_1-P_{1y}D_8+D_2D_8\sin\theta_1+P_{1z}D_7\sin\theta_1-(D_1+D_3)D_7\sin\theta_1)\sin\theta_1 \tag{13}$$

$$B= 2\,(D_7D_5\cos\theta_1-P_{1y}D_7+D_2D_7\sin\theta_1-P_{1z}D_8\sin\theta_1+(D_1+D_3)D_8\sin\theta_1)\sin\theta_1 \tag{14}$$

$$\begin{aligned}C = {} & (D_7^2+D_2^2+P_{1z}^2+(D_1+D_3)^2-2P_{1z}(D_1+D_3))\sin^2\theta_1+P_{1y}^2+D_5^2\cos^2\theta_1 \\ & -2P_{1y}D_5\cos\theta_1+D_8^2+2D_5D_2\sin\theta_1\cos\theta_1-2P_{1y}D_2\sin\theta_1-1\end{aligned} \tag{15}$$

Move B$\cos\theta_2$ to right side, substitute $\cos\theta_2$ with $\cos\alpha = \sqrt{1-\sin^2\alpha}$ and square both sides the rearranged equation then

$$(A^2+B^2)\sin^2\theta_2+2AC\sin\theta_2+C^2-B^2 = 0 \tag{16}$$

Therefore the roots can be expressed as

$$\sin\theta_2 = \frac{-2AC\pm\sqrt{A^2C^2-(A^2+B^2)B^2}}{A^2+B^2} \tag{17}$$

The control value of second joint can be obtained from following equation

$$\theta_2 = \pm\frac{-2AC\pm\sqrt{A^2C^2-(A^2+B^2)B^2}}{A^2+B^2} \tag{18}$$

Replace θ_1 and θ_2 in P_{1z} with (9) and (18) respectively

$$\cos(\theta_2+\theta_3) = \frac{P_{1z}-D_8\cos\theta_2+D_7\sin\theta_2-D_1-D_3}{D_6} \tag{19}$$

$$\theta_3 = -\theta_2\pm\operatorname{acos}\frac{P_{1z}-D_8\cos\theta_2+D_7\sin\theta_2-D_1-D_3}{D_6} \tag{20}$$

C.10 Three revolute axes along Z, Z and Y directions in articulation

For the case of (12) in Figure 4.6, the coordinate frame assignment of three prismatic axes articulated in the Z, X, and Y directions is shown in Figure C.12. The link parameter table required to use the D-H matrix is then

Link	Variable	θ_i	α_i	a_i	d_i	$\cos\alpha_i$	$\sin\alpha_i$
1	θ_1	θ_1	0	D_2	D_1	1	0
2	θ_2	θ_2	-90	D_4	D_3+D_5	0	-1
3	θ_3	θ_3	0	D_7	D_6	1	0

The transformation matrices from frame{0} to frame{1}, frame{1} to frame{2}, frame{2} to frame{3} and frame{0} to frame{3} are respectively

$$T_{121}(0,1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & D_2\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & D_2\sin\theta_1 \\ 0 & 0 & 1 & D_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{122}(1,2) = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & D_4\cos\theta_2 \\ \sin\theta_2 & 0 & \cos\theta_2 & D_4\sin\theta_2 \\ 0 & -1 & 0 & D_3+D_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{123}(2,3) = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & D_7\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & D_7\sin\theta_3 \\ 0 & 0 & 1 & D_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{12}(0,3) = T_{121}(0,1) \times T_{122}(1,2) \times T_{123}(2,3)$$

$$= \begin{bmatrix} \cos(\theta_1+\theta_2)\cos\theta_3 & -\cos(\theta_1+\theta_2)\sin\theta_3 & -\sin(\theta_1+\theta_2) & P_{12x} \\ \sin(\theta_1+\theta_2)\cos\theta_3 & -\sin(\theta_1+\theta_2)\sin\theta_3 & \cos(\theta_1+\theta_2) & P_{12y} \\ -\sin\theta_3 & -\cos\theta_3 & 0 & P_{12z} \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

where

$$P_{12x} = D_7\cos(\theta_1+\theta_2)\cos\theta_3 - D_6\sin(\theta_1+\theta_2) + D_4\cos(\theta_1+\theta_2) + D_2\cos\theta_1$$
$$P_{12y} = D_7\sin(\theta_1+\theta_2)\cos\theta_3 + D_6\cos(\theta_1+\theta_2) + D_4\sin(\theta_1+\theta_2) + D_2\sin\theta_1$$
$$P_{12z} = -D_7\sin\theta_3 + D_3 + D_5 + D_1 .$$

Appendix D The derivation of times for simulation

The derivation t_0, t_1 and t_2 in trapezoidal velocity profile:

If an axis group has a constant acceleration **a** and constant velocity **v** moving on a defined path, then

$$v = \int_{t_0}^{t_1} a \times dt = a \times (t_1 - t_0)$$

$$d = \int_{t_0}^{t_1} a \times (t - t_0)\, dt = \frac{1}{2}at^2 - a \times t_0 \times t = \left(\frac{1}{2}a \times t^2 - v_0 \times t\right)$$

The distance of a two dimensional curve y = f(x) in the region of x = a and x = b can be obtained from

$$s = \int_a^b \sqrt{1 + \left(\frac{dy}{dx}\right)^2} \times dx = g(x)\,|_a^b = g(b) - g(a)$$

Let s =d, where *d* is the distance moved by articulated axes.

The total length of a defined multi-segment path is then

$$S = \sum_{i=1}^{n} s_i = \sum_{i=1}^{n} \int_{a_i}^{b_i} \sqrt{1 + \left(\frac{dy}{dx}\right)^2} \times dx \qquad \text{where, n = 1,2,3, ...}$$

Since $\sqrt{1 + \left(\frac{dy}{dx}\right)^2}$ is not a simple function, a numerical integration method was employed to obtain the total length of the path. The trapezoidal rules are adopted to calculate the numerical value of S. With the defined velocity v and acceleration a, the path length can be used to calculate the total duration t_2 of axes motion spent on the path. Based on known velocity and acceleration values, the distance moved is

$$d = \int_0^{t_0} K_1 \times t dt + \int_{t_0}^{t_1} v \times dt + \int_{t_1}^{t_2} (K_2 \times t + c) \times dt$$

$$= \left(\frac{1}{2}K_1 \times t_0^2 + v \times (t_1 - t_0) + \left(\frac{1}{2}K_2 \times t^2 + c \times t\right)\Bigg|_{t_1}^{t_2}\right)$$

where $c = -K_2 \times t_2$

To establish path control, a symmetric trapezoid (see Figure7.4.1) is assumed, i. e

$K_1 = \tan(\theta)$, $K_2 = \tan(180-\theta) = -\tan(\theta) = -K_1$

On using $K_2 = -K_1$ and $t_2 = t_0 + t_1$ and substituting into the above equation, then

$d = v \times (t_1 - t_0) + K_1 \times t_0^2$

Let d equal S

then $t_1 = \dfrac{(S - K_1 \times t_0^2)}{v} + t_0$

Since a is constant during the acceleration section of trapezoid then

$v(t) = a \times t$; when $t = t_0$, $v(t) = v = \text{constant}$,

hence $t_0 = \dfrac{v}{a}$

Therefore

$$t_2 = t_0 + t_1 = \frac{v}{a} + \frac{\left(S - K_1 \times \left(\frac{v}{a}\right)^2\right)}{v} + \frac{v}{a}$$

$$= \left(\frac{(2v)}{a} + \frac{(S \times a^2 - K_1 \times v^2)}{(v \times a^2)}\right) = \frac{v^2 + S \times a}{v \times a} \quad . \qquad \text{where } K_1 = a.$$

With the desired velocity v and acceleration a (see Figure 7.4.2), t_2 is the estimated time spent on the specified path. The next step is to find the corresponding point (x_i, y_i) at a specific time t_i. In evaluating the position $P_i(x_i, y_i)$ it is extremely difficult to establish an explicit equation for the relationship between the X coordinate and the curve length S, therefore it is often impossible to calculate the exact point $P_i(x_i, y_i)$ on the path with a given time t_i (because from $S = \int_0^x \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx$, it is impossible for most curves to obtain the inverse function x = g(s)). If an explicit relationship y = y(t), x = x(t) is defined, then the point $P_i(x_i, y_i)$ can be obtained through this set of equations. An approximate relationship describing x = x(t) can be established through the kinematic analysis with reference to the desired path. While the end-point of an articulated motion primitive is moving on a specified path, the defined acceleration and velocity can be divided into two elements along x and y directions. The slope of the curve at the point $P_i(x_i, y_i)$ can be obtained from the relationship $y' = \frac{d}{dx}f(x)$, and θ can thus be obtained. The distance moved along the X direction can be approximately expressed as (see Figure 7.4.2).

$$\Delta x = Vx \times \Delta t + \frac{1}{2}ax \times \Delta t^2 = Vx \times \Delta t + \frac{1}{2} \times a \times \cos(\theta) \times \Delta t^2$$

and

$$x(t_{i+1}) = x(t_i) + \Delta x(t_i)$$

$$= \left(x(t_i) + V_x(t_i) \times \Delta t + \frac{1}{2} \times a_x(t_1) \times \cos(\theta(t_i)) \times \Delta t^2\right)$$

where x (t_i) is the distance moved along the X direction at time t_i ;

V_x (t_i) is the elementary velocity along the X direction at time t_i ;

$a_x(t_i)$ is the acceleration along the X direction at time t_i ;

t is the time interval ;

x (t_{i+1}) is the estimated distance that the motion primitive will have moved after a time interval Δt .

Times t_0, t_1 and t_2 can be used to evaluate the velocity v and acceleration a in Figure 7.4.1 in the above calculations. Using such an approach x (t_{i+1}), y (t_{i+1}) can be easily obtained knowing the path specification equation y = f(x). After calculating values for x (t_{i+1}) (or x_i for short) and y (t_{i+1}) (or y_i for short) on the path, these cartesian coordinates need to be converted into axis coordinates so that the chosen machine axes of motion can establish the target position on the path. The inverse kinematic transformation equations described in chapter 6 can be employed to calculate the axis coordinates at time t_{i+1}.

Although the velocity and acceleration at time t_i are used to calculate the distance for time interval Δt, improved accuracy can be achieved by using the velocity and acceleration at the mid-point between x_i and x_{i+1}. This is achieved at the expense of recalculating x_{i+1} with mid-point v and a after the first time calculation of x_{i+1}.

Appendix E An example of a CAMLINKS motion file

This appendix gives an example of the data format used in a CAMLINKS motion profile. The motion output of the following file is only partially listed. The main objective is to show the data format for various motion profiles used in CAMLINKS.

```
Results of motion design for CAMLINKS
Motion output is linear
At start of segment one, input = 0.00
At start of segment one, output = 0.00
Input speed (cmp) = 60.00
Segment number:     1       Segment type:     dwell
input - start:      0.00    end:        60.00        change:       60.00
output - start:     0.00    end:        0.00         change:       0.00
                 segment valid - true

Values across segment of


     input            output          output vel       output acc

0.0000000E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
1.8181818E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
3.6363637E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
5.4545456E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
7.2727274E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
9.0909091E+00    0.0000000E+00    0.0000000E+00    0.0000000E+00
1.0909091E+01    0.0000000E+00    0.0000000E+00    0.0000000E+00

      ⋮                ⋮                ⋮                ⋮

5.8181822E+01    0.0000000E+00    0.0000000E+00    0.0000000E+00
6.0000000E+01    0.0000000E+00    0.0000000E+00    0.0000000E+00

Segment number:     2       Segment type:     polynomial
input - start:      60.00   end:        180.00       change:       120.00
output - start:     0.00    end:        100.00       change:       100.00
                 segment valid - true
pinpvia = 5.0000000E-01
reqal=
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 1.0000000E+02
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
```

```
ord(reqmask)=
o =          0
o =          0
o =          1
o =          3
o =          3
o =          3
o =          3
o =          3
o =          2
o =          1
o =          1
o =          3
pol=
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 9.9999934E+02
p = -1.4999987E+03
p = 5.9999948E+02
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00

Values across segment of

   input            output           output vel       output acc

6.0000000E+01    0.0000000E+00    0.0000000E+00    0.0000000E+00
6.1791044E+01    3.2508817E-03    1.9454984E+00    7.7024048E+02
6.3582086E+01    2.5422221E-02    7.5479620E+00    1.4704589E+03
6.5373128E+01    8.3850175E-02    1.6464382E+01    2.1028102E+03

      :                :                :                :

1.7283581E+02    9.9805921E+01    2.8364136E+01    -2.6694488E+03
1.7462685E+02    9.9916237E+01    1.6466583E+01    -2.1028093E+03
1.7641790E+02    9.9974668E+01    7.5487059E+00    -1.4704628E+03
1.7820894E+02    9.9996793E+01    1.9467774E+00    -7.7023922E+02
1.8000000E+02    1.0000006E+02    1.4648437E-03    8.7890613E-03

Segment number:    3      Segment type:    polynomial
input - start:     180.00    end:       220.00       change:     40.00
output - start:    100.00    end:       100.00       change:     0.00
                   segment valid - true
pinpvia = 5.0000000E-01
reqal=
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
```

```
r = 0.0000000E+00
r = 0.0000000E+00
r = 1.0000000E+02
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
ord(reqmask)=
o =          0
o =          0
o =          1
o =          3
o =          3
o =          3
o =          3
o =          3
o =          2
o =          1
o =          1
o =          3
pol=
p = 1.0000006E+02
p = 1.6276042E-04
p = 0.0000000E+00
p = -9.7656184E-04
p = 1.3020822E-03
p = -4.8828083E-04
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00

Values across segment of

   input           output          output vel       output acc

1.8000000E+02    1.0000006E+02    1.4648437E-03    0.0000000E+00
1.8181817E+02    1.0000007E+02    1.4146745E-03    -1.9032499E-02
1.8363636E+02    1.0000007E+02    1.2806503E-03    -3.3280870E-02

      |                |                |                |

2.1999999E+02    1.0000006E+02    -1.3096722E-03   -7.5437128E-08

Segment number:    4        Segment type:    polynomial
input - start:    220.00    end:         360.00        change:       140.00
output - start:   100.00    end:           0.00        change:      -100.00
                  segment valid - true
pinpvia = 5.0000000E-01
reqal=
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
```

```
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
r = 6.1035156E-05
r = 0.0000000E+00
r = 0.0000000E+00
r = 0.0000000E+00
ord(reqmask)=
o =           0
o =           0
o =           1
o =           3
o =           3
o =           3
o =           3
o =           3
o =           2
o =           1
o =           1
o =           3
pol=
p = 1.0000006E+02
p = 5.0931704E-04
p = 0.0000000E+00
p =-9.9999934E+02
p = 1.4999987E+03
p =-5.9999948E+02
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00
p = 0.0000000E+00

Values across segment of

    input              output            output vel         output acc

2.1999999E+02      1.0000006E+02      1.4648437E-03      0.0000000E+00
2.2179487E+02      1.0000007E+02      1.4146745E-03     -1.9032499E-02
2.2358973E+02      1.0000007E+02      1.2806503E-03     -3.3280870E-02

      ⋮                  ⋮                  ⋮                  ⋮

3.5641023E+02      1.6291618E-02     -4.8168769E+00      9.4034641E+02
3.5820513E+02      2.0527839E-03     -1.2354911E+00      4.8923507E+02
3.6000000E+02      0.0000006E+02     -1.2555803E-03     -6.4572703E-03
```