# ADVANCEMENT IN ROBOT PROGRAMMING WITH SPECIFIC REFERENCE TO GRAPHICAL METHODS

BY

Sun Fat Chan

A Doctoral Thesis

submitted in partial fulfilments of the requirements

for the award of

Doctor of Philosopy

of Loughborough University of Technology

April 1989

Department of Manufacturing Engineering

Loughborough University of Technology

**TO MY FAMILY FOR THEIR LOVE, ENCOURAGEMENT AND SUPPORT**

## ACKNOWLEDGEMENTS

I am grateful to the Science and Engineering Research Council for its scholarship.

I would like to express my sincere thanks to my supervisors Professor R.H. Weston and Dr K. Case for their invaluable supervision, encouragement and help throughout this research study. Thanks must also extend to my director of research Dr E. Roberts, and those staff of the Department of Manufacturing Engineering who have been kind and helpful. These people include: Mr G.P.Charles, Mr J.D.Gascoigne, Mr S.I.Murgatroyd and Mr D. Walters.

I would like to take this opportunity to express my thanks to BYG Systems limited for their collaborative work and mutually beneficial discussions in the first year of this study.

# SYNOPSIS

This research study is concerned with the derivation of advanced robot programming methods. The methods include the use of proprietary simulation modelling and design software tools for the off-line programming of industrial robots. The study has involved the generation of integration software to facilitate the co-operative operation of these software tools.

The three major research themes of "ease of usage", calibration and the integration of product design data have been followed to advance robot programming. The "ease of usage" is concerned with enhancements in the man-machine interface for robot simulation systems in terms of computer assisted solid modelling and computer assisted task generation.

Robot simulation models represent an idealised situation, and any off-line robot programs generated from them may contain discrepancies which could seriously effect the programs' performance. Calibration techniques have therefore been investigated as a method of overcoming discrepancies between the simulation model and the real world.

At the present time, most computer aided design systems operate as isolated islands of computer technology, whereas their product databases should be used to support decision making processes and ultimately facilitate the generation of machine programs. Thus the integration of product design data has been studied as an important step towards truly computer integrated manufacturing.

The functionality of the three areas of study have been generalised and form the basis for recommended enhancements to future robot programming systems.

# TABLE OF CONTENTS

# CHAPTER ONE

# INTRODUCTION

The increasing pressure of international competition has forced manufacturers to increase their product range and models, and to improve their product quality. An increase in product models and product range results in changes in production processes, and a requirement to minimise set up and programming time. This must be accomplished with the minimum of disruption to production. For these reasons, manufacturing industry has sought to employ advanced automation to achieve the flexibility and productivity levels required. One solution to this problem has been the introduction of robots - machines that can be reprogrammed. As a result, robots have become important tools and can play a significant role in today's flexible manufacturing systems (FMS) and flexible assembly systems (FAS).

Almost all robots are currently programmed by a combination of teach and on-line textual programming methods including: teach by showing, teach by pendant, and teach by showing a replica. Rapid growth and technology developments in the electronics industry have made robot controllers and computers more powerful, and this has led to the development of the current generation of robots, and subsequently the development of numerous robot languages. The use of on-line robot programming languages has improved their flexibility when dealing with complexities in the programming environment, but there are still limitations. The limitations of these on-line programming methods have hindered the wider application of robots.

Advances with respect to graphical displays have allowed simulation on graphical terminals to become a reality. Many of the current generation of robot simulators have been designed and implemented for the specific task of

designing the layout of robotic systems. Graphical off-line robot programming based on the use of these robot simulators, is a subsequent and natural development.

Graphics-based off-line robot programming systems involve high capital investment and their introduction must be justified both on economic and technical grounds. At present, off-line robot programming should still be considered to represent an evolving research topic, which to date has had only limited industrial use. However, it represents a future method that will be used very commonly by manufacturing industry for the programming of robotic devices. The restricted usage of graphical off-line programming methods can often be traced to the original intention of the robot simulators as layout design tools rather than off-line programming tools. However, there are also practical problems associated with this method of programming, where modelling accuracy can cause a serious problem. Off-line robot programming can be widely applied so long as accuracy and other practical problems can be design out or suitable methods found for reducing their effect.

At the start of this research (October 1985), there were only a limited number of robot simulation tools available and very few of these were capable of generating off-line robot programs. The graphical simulation system (GRASP) chosen for this research was considered appropriate as it was a developing system capable of further research and development. At the time the GRASP system had virtually no language post-processors capable of translating graphics output to robot programming languages. It therefore provided a good starting point for the research in studying the

methodologies involved in language post-processors and the subsequent construction of a language post-processor suitable for the robots available to the author. Although readily accepted today, the concept of using post-processors for robots was in its infancy even in 1985. BYG Systems (the vendor of GRASP) completed its commercial version of the VAL II post-processors during the winter of 1986 and this software was supplied to Loughborough University of Technology for pre-release testing. Both the BYG and the author's post-processor versions were evolved concurrently and used in this research.

The main topic of this thesis is the study of off-line robot programming in the context of computer integrated manufacturing, with particular reference to handling and assembly tasks. This has involved an appraisal of robot simulation and off-line programming methods. The limitations of recent commercially available off-line robot programming systems have been identified and an extensive study has been made into the provision of practical solutions. The research work progressed along three themes with software being produced to facilitate calibration, an interface to product design, and improved man-machine interface capabilities. The idea of re-using product design data when programming robot tasks is a direct parallel of using design information in the NC arena during the early sixties, but the target robots and their application areas demonstrate far greater variability and hence complexity. Although the project software has been designed specifically to operate with the GRASP robot simulator, the functionality was subsequently generalised.

This thesis is essentially comprised of three parts. The first four chapters provide an introduction and describe the background evolution and development of programming methods. The fundamental concepts of using simulation for designing the layout of robot systems are also considered. The second part (chapters five to eight) constitutes the main body of the study. Chapter five discusses the methodologies used in language post-processing. Techniques, algorithms and practical solutions to providing improvements to the man-machine interface are reviewed in chapter six. Chapters seven and eight describe the calibration methods used to update the information used in a CAD model and to describe the methods employed to enable the use of product design data in a robot simulator. The third and final part of this thesis shows how off-line robot programming methods can be utilised within existing manufacturing systems and discusses future possibilities for using off-line robot programming.

# CHAPTER TWO

# LITERATURE SURVEY

## 2.0 Introduction

Computer Integrated Manufacture (CIM) is an approach to manufacturing which uses computer technology to improve productivity through the availability and processing of information from all phases of manufacturing [Kusiak and Heragu, 1988]. A CIM system is commonly thought of as a truly integrated CAD/CAM system [Allen, 1987; Crookall, 1987] consisting of all the activities from the planning and design to manufacture of a product. As with traditional manufacturing approaches, the purpose of a CIM system is to transform product designs and materials into saleable products at a minimum cost in the shortest possible period of time.

During the last decade, industrial robots have become an efficient tool in many manufacturing operations [El-Zorkany, 1985] and have been applied to an ever increasing number of manufacturing tasks with different levels of complexity. This advancement was due to the significant improvement in performance capabilities and sophistication of computers, associated computer languages, and robots themselves. Today's production systems must be able to cope with varying production volumes, and be flexible to handle a wide range of product models [Dooner, 1987]. Industrial robots are becoming more advanced and play an important role in many manufacturing areas of a CIM system including assembly where operational flexibility is required for handling the variability implied by small batch sizes [Sata et al, 1987]. To apply robots effectively to these more demanding tasks requires an increase in the flexibility of the robot system and places new demands on the skills of the human operators. In order to achieve such flexibility, robot systems should incorporate more sensing devices [Crosnier and Fournier, 1987] and make more use of effective ways of programming robots

such as off-line programming [Paul, 1983]. Where possible robot systems should be programmed off-line so as to achieve more rapid development of programs and thus better machine utilization. Off-line robot programming methods normally involve the use of design layout information captured in a model structure database. This provides the possibility of linking up different areas within manufacturing industry, for example, integration of product design data for programming robots and vision systems, planning, scheduling and control [Appleton et al, 1988]. Therefore, robot programming methods are an important stage of development for the integration of robots into complex manufacturing systems [Duelen and Bernhardt, 1986], but, most available systems still have the disadvantage of being tailored only to one type of robot [Weck et al, 1984].

There are some parallels in the development of programming methods for robots with the development of part programming for numerically controlled (NC) machines, where initial requirements were for methods of programming single, relatively simple machines through the facilities of their own controllers (on-line programming). Advances in computer-aided design and manufacturing (CADCAM) and the advent of computer and direct numerical control (CNC/DNC) made off-line programming a realistic proposition for groups of more complex machines indirectly communicating with each other through shopfloor computer systems [Milner and Brindley, 1978; Groover, 1987]. Unfortunately, the design and programming of a manufacturing system which includes industrial robots can involve greater complexity than that for a system solely of NC machines. This is a direct consequence of the wider variety of possible application areas and potential solutions given the enhanced flexibility offered by robotics.

## 2.1 Development of Robots in Relation to Robot Programming

The development of robots is a continuous process and thus far three distinct generations [Ambler, 1984] can be classified as follows.

### (a) First Generation

The first generation robots were mainly designed for use in applications where large batch sizes and repetitive tasks were involved. For this reason, robot programming was emphasised in methods of teaching where the programming time is relatively small when compared to its total program running time. Early robots were capable of repeating a strictly specified set of operations under conditions completely determined in advance [Ayres et al, 1985], and programming involved a combination of simple command statements and teach pendant. This type of robot language is known as a "motion level" language [Snyder, 1985]. First generation robots continue to be used in industry but with advances in computer technology, enabling the creation of second and third generation machines, they are no longer produced.

### (b) Second Generation

Since second generation robots are designed to cope with small batch operations, they are reprogrammable and are capable of easily switching from one program to another. This can include adaptive robots capable of operating under variable or partially unknown conditions and able to respond to environmental changes [Vukobratovic and Stokic, 1982]. These robots incorporate sensors which provide information about changing

external conditions. They perform a set of operations determined in advance and are capable of accomplishing the same operations under changing operating conditions. This type of robot language is described as a "structured language".

(c) Third Generation

This involves intelligent robots, possessing certain features of artificial intelligence [Kochan, 1987a]. They are capable of responding to their operating environment in defining instantaneous tasks, of self-learning to provide solutions to particular problems including automatic error correction, and of changing their own action in accordance with the variations in operating conditions. Thus the planning of operations can be generated by robots themselves. This generation of robots is still not yet fully available for industrial implementation [Wolovich, 1987].

If off-line robot programming is to be applied to first generation robots, extra computation is required to be done within a post-processor to compute inverse kinematics (calculate the equivalent robot joint angles for achieving the Cartesian coordinates of the robot's end-effector). Although the first generation of robots are commonly programmed by teach pendant [Ambler, 1984], their positioning accuracy (the ability of the robot to attain a commanded position) is commonly not good enough for assembly tasks, which often require reasonably tight tolerances. These robots will suffer greater problems with accuracy [Chen and Chao, 1987] if target positions are generated by off-line robot programming methods (robot accuracy normally deteriorates significantly

if positions are defined by off-line robot programming methods, due to the approximations made in modelling the kinematics and dynamics and when implementing control algorithms in the robot controller). The languages used for these robots are normally very simple and the program flow control is limited. The robot controller can only process one program at a time (a fundamental problem of processing capability of first generation robot controller) and is therefore not suitable for complex operations where sensory feedback is considered necessary for monitoring the robot movement [Gini et al, 1987]. Off-line robot programming is feasible only for simple robot tasks where the accuracy required is not critical.

The second generation of robots have commonly been designed to achieve better accuracy and repeatability. Communication facilities between the off-line computer systems and the robot controller are also commonplace [Groover et al, 1986]. The processing capability of the second generation robot controllers is far more powerful than that of their predecessors. Off-line robot programming of second generation robots can be applied to assembly tasks where precision is required. The increased complexity of task and computational power makes off-line programming both desirable and achievable.

As more intelligent third generation robots become available with enhanced capabilities for coping with changing operating conditions, these robots are expected to demonstrate even better accuracy than previous generations. This makes off-line robot programming even more suitable and more desirable because these robots could easily meet the accuracy requirement.

## 2.2 Robot Programming Classifications

There is no standard way of classifying robot programming methods, and various approaches have been adopted by different authors and researchers. Some authors classify robot programming methods according to on-line and off-line categories [Storr and Schumacher, 1987] and language syntax [Rock, 1989]. The most commonly used method of classification is according to the way of task specification [Yong et al, 1985; Gini, 1987; Van Aken and Van Brussel, 1988]. This refers to the levels of abstraction that can be made in formulating a task. Lozano-Perez (1983), and El-Zorkany (1985), classify robot programming systems into guiding and language based categories with the inclusion of level of task specification as shown in figure 2-1.

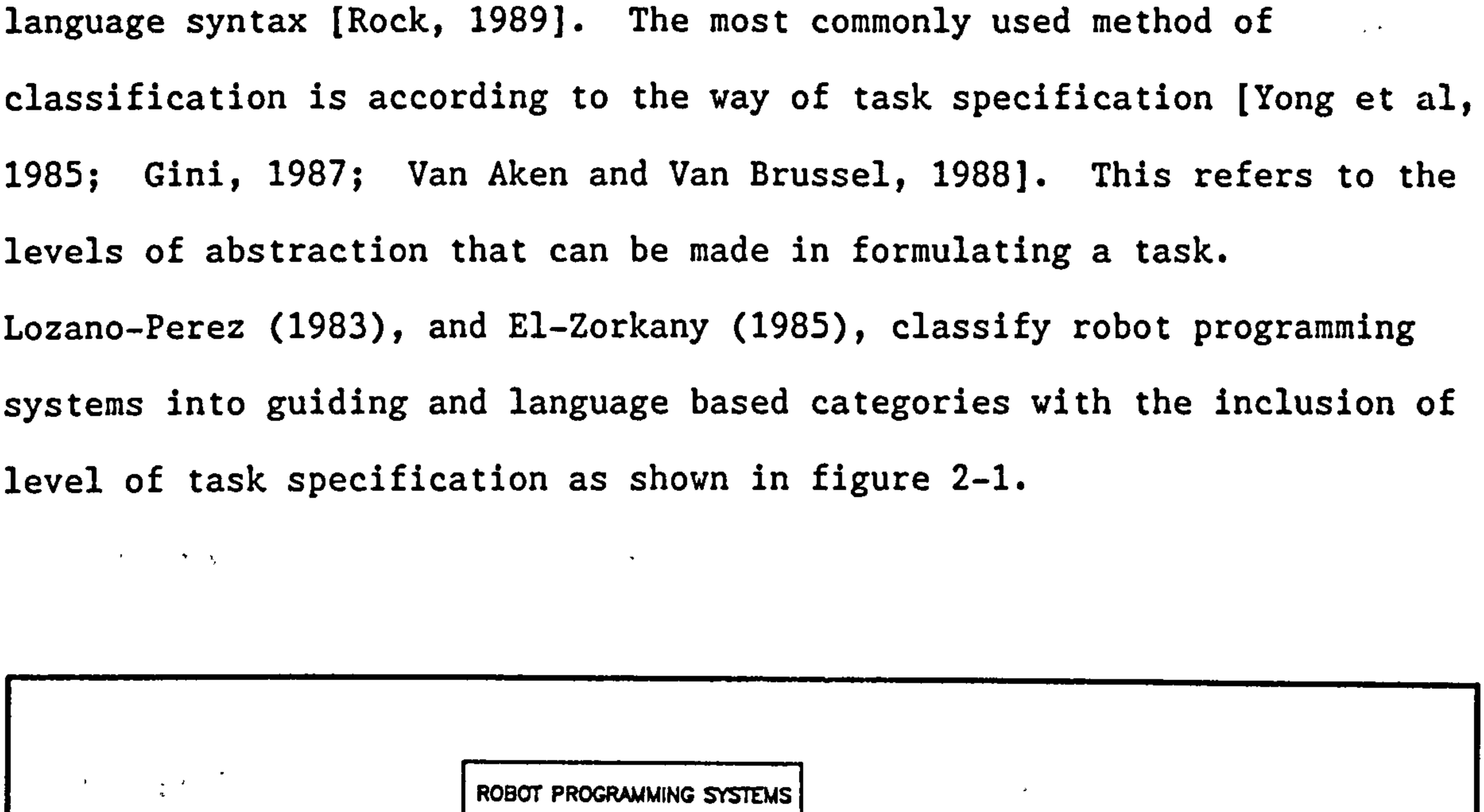



Figure 2–1 Traditional classification of robot programming methods

## (a) Guiding Methods

Guiding was the earliest and still the most widespread and commonly used method of programming industrial robots [Weck et al, 1987]. Basically, the robot is manually moved to each desired position and the sequence of motions is obtained by sampling the robot joint angles. The replay of the motion is achieved by moving the robot through the recorded joint angles.

Guiding can be done in a number of ways:-

(i) The robot is physically moved by the operator.

(ii) The robot is moved by using its own servo system e.g. via a teach pendant.

(iii) A replica of the robot arm is used. It is lighter but less accurate and obviously more costly than (i) and (ii) above.

## (b) Language Based Programming Methods

Language based programming methods [Volz, 1988], where the robots are programmed through the use of high level robot languages, permit the locations and orientations of target positions (to which the robot movements are referred) to be assigned without any need to use the robot itself [Gini, 1987]. Language based programming systems could be subdivided into four levels according to motion specification:- joint level, robot level, task/object level, and objective level. Figure 2-2

shows the inter-relationships of these programming levels. Gini (1987), Van Aken and Van Brussel (1988) classified language based programming methods within these subdivisons.

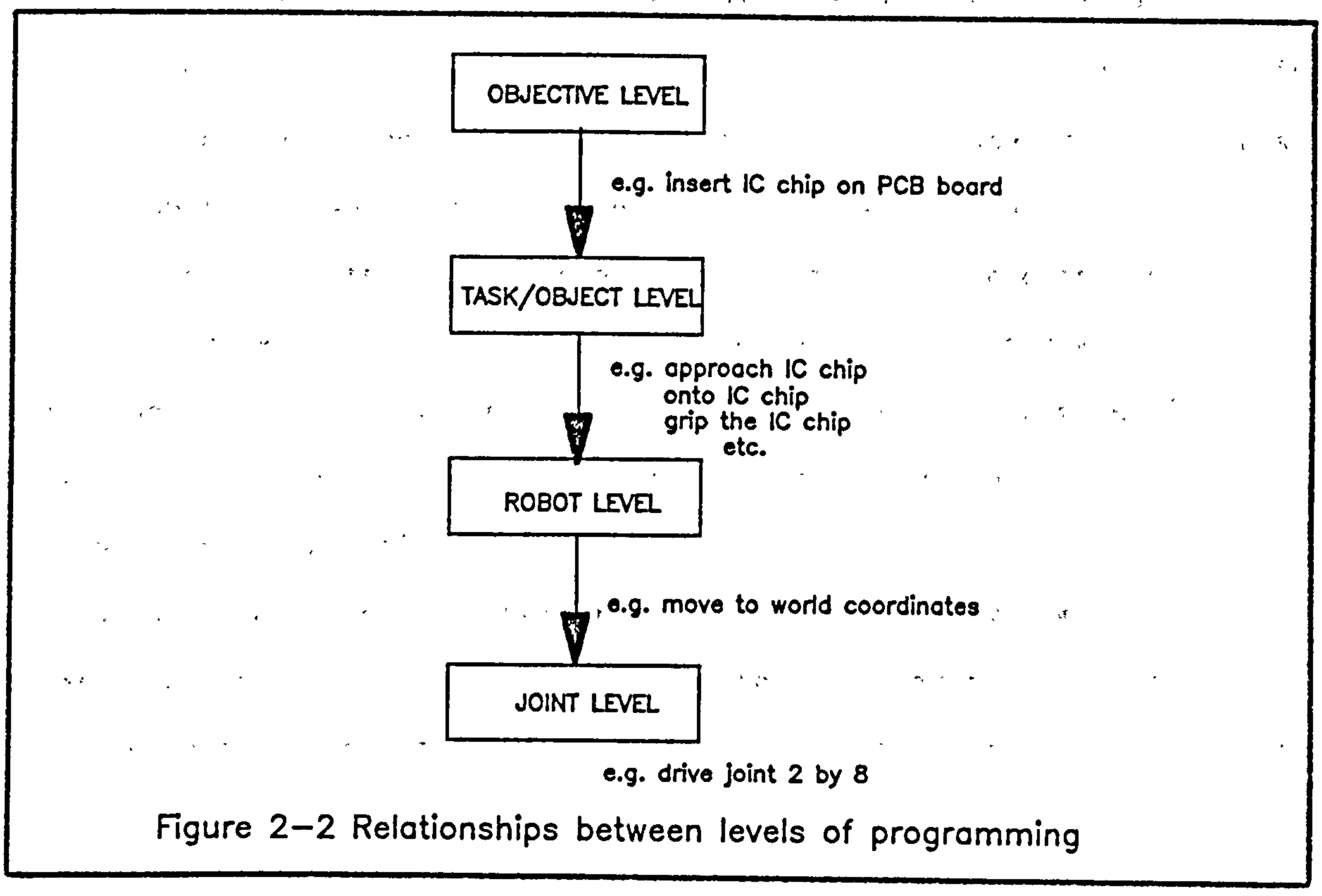



Figure 2–2 Relationships between levels of programming

(i) Joint level programming

The robot movement is programmed in terms of the individual joint movement to achieve the required positions [Lozano-Perez, 1983]. This method is typically used for the restricted programming tasks found with first generation robots. This would be the natural level of programming only in the case of cartesian robots, where the joint description (in this case, prismatic joints) of the end-effector position (location and

orientation) is naturally in Cartesian coordinates.

(ii) Robot level programming

Robot level programming systems incorporate computer programming languages with commands to obtain information from sensors and to specify robot motions [Milovanovic, 1987]. Robot actions are specified in terms of the robot's end-effector locations and orientations, usually in cartesian co-ordinate space. Robot level languages enable the data from external sources, such as vision and force sensors, to be used in modifying robot motions. Robots can cope with a greater degree of uncertainty in the position of objects through the use of sensors, thereby increasing their range of application, especially where high precision is required. However, robot level language systems require the robot programmer to be an expert in computer programming and in the design of sensor-based motion strategies where complex robot tasks are to be defined.

(iii) Task/Object level programming

The aim of task/object level programming is to provide the power of robot level languages without requiring robot programming expertise [Yong et al, 1985]. The focus of attention is on the object being manipulated and the programmer specifies the robot motion in terms of target objects. The objects' coordinate frames are used for computing the robot movement. This requires a complete geometric model of the robot system from which information can be extracted to determine the necessary manipulator locations.

(iv) Objective level programming

Objective level (application) programming is that in which tasks are specified in the most general form. This requires a comprehensive database containing complete geometric models of the robot and of the environment (similar to object level). In addition to this, a knowledge base containing application techniques must be available to support an intelligent planning facility for generating collision free robot paths. Objective level programming is a future goal which at present is a research goal for many university, government agencies and industrial research institutes [Adler, 1986; Rodighiero and Canciani, 1987; Halme et al, 1987; Sata et al, 1987; Howe and Fothergill, 1988 and Hoermann, 1988]. At the present time, objective level programming systems are not commercially available [Van Aken and Van Brussel, 1988].

As an alternative to the above, Bonner and Shin (1982), have suggested that robot languages can be subdivided into five loosely formulated levels according to language emphasis. The five levels are machine code programming of microcomputer, point to point, primitive motion, structured programming, and task-oriented levels. Hocken and Morris (1986), use very similar classifications but the classification emphasis is according to motion and language (mixed) specifications.

Definitive classification of robot programming methods is a difficult task since some robot language characteristics overlap at different levels. Different ways of classification (according to motion specification or language emphasis) have been presented here, but these only represent other authors' points of view.

This author suggests that robot programming languages can also be classified according to their chronological development, from which future development trends can be predicted. Language based methods of robot programming are a development of teach methods so as to provide more flexibility. The language based methods can be subdivided into explicit and implicit robot programming.

(i) explicit robot programming languages

When using explicit robot programming tools, the programmer must specify the robot motion in a complete and detailed way [Storr and Schumacher, 1987]. The programming statements must be sufficient to fully define the required operation sequences together with the required positional and orientational data. Based on this definition, most on-line and some off-line robot programming languages can in appropriate circumstances be considered 'explicit' [Ranky, 1984]. Typical examples of explicit robot languages are AL [Gini and Gini, 1985], VAL [Shimano et al, 1984], AML [Grossman, 1985], MCL [Wood and Fugelso, 1983], RAIL [Gruver et al, 1983] and KAREL [GMF Robotics, 1986].

(ii) implicit robot programming languages

Implicit languages acquire a significant proportion of information from a source other than the human robot programmer. Model and graphical based languages are available. Model based languages involve the use of model databases which capture geometric and spatial information relating to workplace entities [Sata et al, 1987]. Graphics-based languages use similar model based concepts but are extended to include simulation and

animation [Chawla and Gruver, 1984; Leu and Mahajan, 1984]. These graphics-based languages are also known as robot simulators, and can further be subdivided into kinematic and dynamic robot simulators [Kretch, 1982; Thomson, 1984]. The author's view of the spectrum of current robot programming methods is depicted in figure 2-3. Detailed descriptions of each classification, with a representative selection of robot simulators are presented in section 2.6.

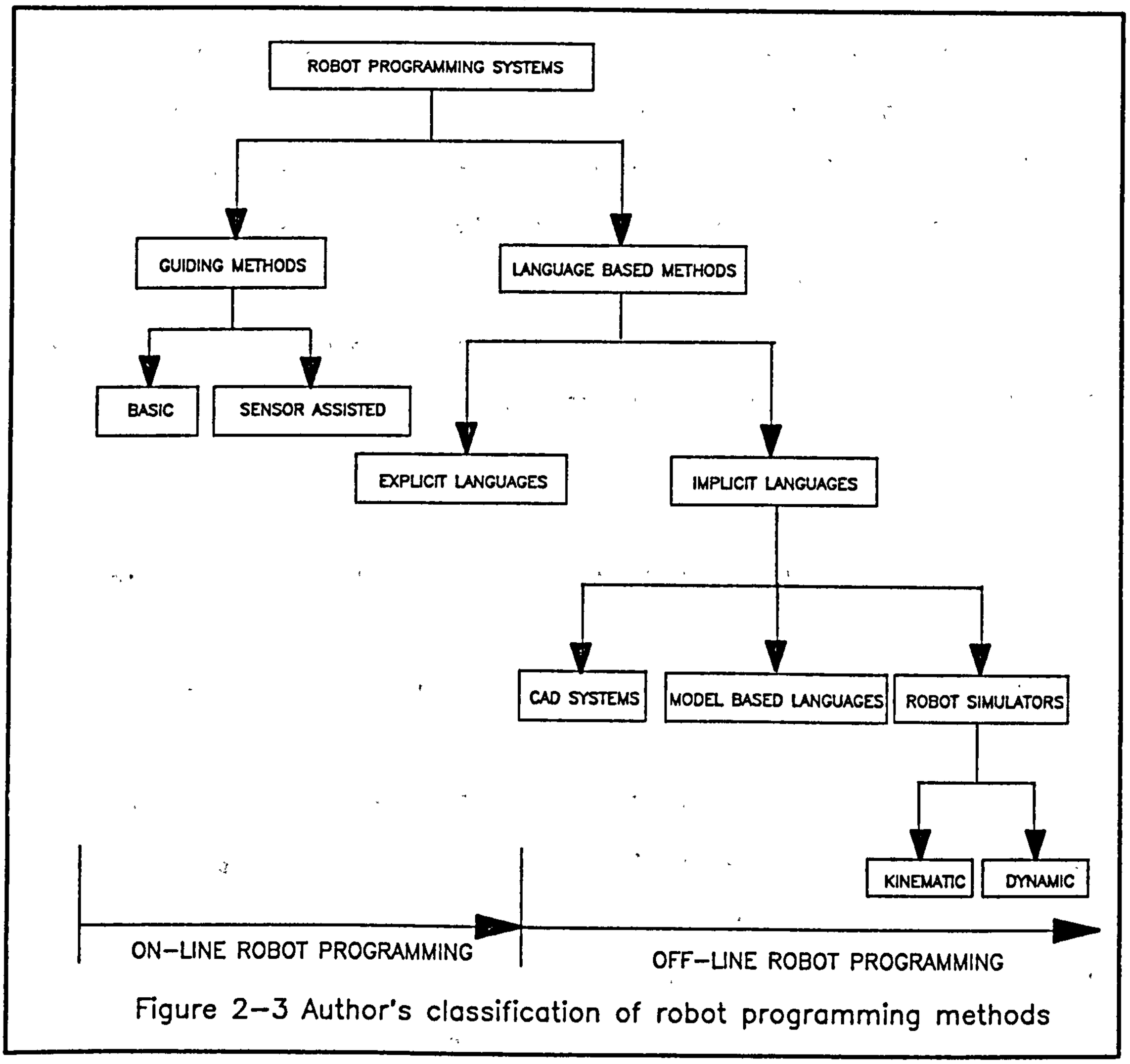



Figure 2-3 Author's classification of robot programming methods

## 2.3 Definition of On-line and Off-line Robot Programming

"On-line" programming of a robot requires the use of the actual robot (and its controller) which is physically put through a sequence of motions in the real environment [Schreiber, 1984a]. This is sometimes alternatively referred to as 'guiding'.

"Off-line" programming means off-line with respect to the robot rather than the robot controller [Gini, 1987]. Off-line programming can be considered as the process of generating robot programs and control logic, where the need to involve the robot is minimised, and only a final tuning (calibration) might be done with the real robot. Possible methods include structured robot languages, model based languages, and graphics-based languages (involving the use of model data describing 3D objects and spatial relationships between objects). For more detail see section 2.6.3.

## 2.4 Comparison of On-line and Off-line Programming

The widespread use of multi-axis robots can still result in excessive programming time and so on-line programming is not satisfactory in many application areas [Weck et al, 1984]. In addition, frequent reprogramming of robots in small and medium batch production increases the production down time of the whole robotic system. The batch size may be so small that it takes more time to programme the robot than to run it. This problem is even more pronounced where the robotic system is part of a wider, highly automated production system such as a flexible manufacturing system or flexible assembly system [Lambourne, 1986]. For

example, in an automotive plant where hundreds of robots are used in an assembly line, it is very inefficient to train robots on-line during the change over period as once a robot is interrupted for training, the whole assembly line is stopped [Schreiber, 1984a]. Although these robots are infrequently reprogrammed, it could take up to several months to reprogramme the robots. If the robot programming method is modified so that new task can be programmed before parts are manufactured and without disrupting the existing system, then car manufacturers can reap the advantages of productivity, flexibility and safety. The ability to shorten lead time for the introduction of new models is most highly valued by industry [George and Mital, 1987]. A typical example is found at Renault, where off-line robot programming is applied to car panel assembly [Renault, 1987].

An alternative solution is to purchase and set up a complete second or training work environment in order to minimize production downtime [Howie and Williams, 1984]. However, this is rarely economically justifiable due to the increased capital investment.

Off-line programming will facilitate the change over from manufacturing one product to another to be carried out in the shortest possible time [Yong et al, 1985]. Off-line programming will allow robots to remain in the workplace performing manufacturing operations while being programmed for another task.

Computer Aided Design (CAD) systems are widely used worldwide, and off-line programming is the best way of using the design data captured in CAD systems [Stobart, 1987]. The use of robot simulators will make it

easier to specify and develop the optimum sequence of robot motions [Rembold et al, 1988]. Integration with CAD systems could take various forms, for example in printed circuit board (pcb) assembly, although the same size pcbs and same electronic components are used, the assembly position of each component may vary. If on-line teach methods are used, the production downtime becomes significant and the location data may not be as accurate as those obtained from the CAD database. Since on-line programs are taught for a specific robot they can only be used on that robot. Teaching hundreds of separate points in a complex welding or riveting tasks by the guiding method leads to increased chance of error [Schreiber, 1984a]. Furthermore, the integration of a CAD system with an off-line robot programming system allows better communication and enables faster modelling where the duplication of model description is no longer required.

Off-line programming systems can generate task location data in a robot-independent format, and hence the replacement of one robot by another should not prove too difficult [Okino and Shono, 1987]. In this way, a variety of robots can be programmed with the use of an appropriate post-procesor for each specific robot controller. In such an off-line programming system, the programmer need not be an expert in many robot programming languages, but ought to be familiar with the programming language of the robot simulator [Chan et al, 1988]. This obviously improves flexibility.

Some tasks, such as programming spherical or circular motion [Gettelman, 1985 ] cannot be taught satisfactorily by an operator using on-line guiding methods. Another typical example might be the cutting of a 45

degree bevel, measured normal to a curved surface. However, these difficult tasks can be performed successfully through off-line programming systems which utilise information from the product design database. This method improves the programming efficiency and the quality of work.

Since robot simulators are commonly used for workshop layout design and task evaluation, robot tasks can be programmed before equipment arrives or parts are manufactured [Pickett, 1984]. A good example is the welding of space shuttle engines [Fernandez, 1988], where each engine has a variety of weld geometry and parameters. With the availability of both computing power and workplace model database, task cycle time can be estimated and used for process planning and scheduling [Chan et al, 1988].

Robots can be dangerous to operators and surrounding auxillary equipment. However, using graphical programming methods to debug programs and to train operators, the human operator is removed from dangerous environments [Ambler, 1984]. As more tasks are programmed away from the robot, the time during which the programmer is at risk from aberrant robot behaviour is reduced [Yong et al, 1985].

As off-line programming of robots will provide many potential advantages over the traditional on-line programming methods, off-line robot programming methods are expected to have potentially wider industrial application. Currently the major hurdles to more widespread usage is user confidence, cost and learning.

Since robot performance is affected by repeatability (the ability of the robot to return to a taught position), the on-line robot programming method is most frequently used. It is therefore natural of robot manufacturers to focus on the problem of repeatability rather than accuracy. Off-line robot programming methods require good robot accuracy as target positions are generated from sources other than human operators. Improvements in robot accuracy will lead to the use of off-line robot programming in an ever increasing number of applications and will accelerate the trend towards fully utilized robot-based flexible manufacturing or assembly systems.

## 2.5 Current Applications of Off-line Robot Programming

Off-line robot programming has been increasingly applied in various industrial sectors. Many manufacturing companies that are using CAD/CAM facilities, and particularly those involved in using robot simulation for factory design, have great potential for applying off-line robot programming. Although off-line robot programming can offer many potential advantages, some applications involving simple tasks would find off-line robot programming methods too complicated; this may mean that more time is required to programme the task programs than to run them (in some cases, it is more time consuming to programme the task off-line than on-line). However, other advantages can be gained through the use of off-line robot programming methods as discussed in section 2.4. Some representative current applications of off-line robot programming are described below.

(a) Aerospace - Welding

The Rockwell Science Centre, California, USA., (under contract to the National Aeronautics & Space Administration NASA), uses the McDonnell Douglas off-line robot programming system to design and program robot welding cells [Kuvin, 1985]. NASA engineers programmed the operation before the equipment arrived [Fernandez, 1988] (based on the welding parameter database specified by welding engineers and on CAD databases that describe the parts to be welded). The simulation relates to a production cell that welds Space Shuttle main engines [Ruokangas et al, 1987]. The welding cell uses at least nine Cybotech H8 electric robots which are controlled by RC-6 robot controllers. Each engine has 14,000 inches of weld and a variety of weld geometries. Obviously, on-line teaching would require extensive debugging and lengthy downtime. The off-line robot programming system holds a weld parameter database, which is processed to facilitate programming of robotic tungsten arc welding.

(b) Automobile Industry - Spray Painting

Robot programming for spray painting applications is currently almost exclusively done by manual guiding methods. This has serious drawbacks since the worker cannot produce the best quality of coating (being hindered by the weight and stiffness of the robot arm and skill of the programmer) especially in the case of large surfaces [Klein, 1986] such as bus bodies [Grunewald, 1984; Frederikson 1984].

Advances in CADCAM methods and a growing interest in off-line robot programming led to this technique being applied for painting robots at

the Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary [Klein, 1987]. Their work led to a better quality of coating and a decrease of paint loss. The approach involves modelling the spatial distribution of the paint particles. The paint particles sprayed should form a cone (which is the usual case [Hauke, 1982]) and by considering its intersection with the surface the amount of paint delivered to any point can be evaluated and visualized in different colours on a CAD system. Theoretically, when the workpiece has large surfaces which are planar or close to planar and the surface normal changes regularly, automatic path generation is possible. Based on this a workpiece of arbitrary shape can also be painted by segmenting the workpiece into planar patches; that is curved surfaces are approximated by polyhedra. From their experimentation, the efficiency of the painting process increases with the size of the surface to be painted.

Efficiencies of 70-80 % have been achieved through off-line programming which are better than with current practice where a 55-60 % efficiency is typical (Efficiency is defined as being the ratio of paint usefully applied to the total consumed).

(c) Automobile Industry - Spot Welding

The General Motors Corporation plant in Doraville, USA, applies off-line robot programming for automotive spot welding [Yoffa, 1988]. This involves the use of the IGRIP simulation system to model the body framing station (a station that holds the car body in place ready for welding to proceed). The framing station includes six GMF S-480R robots, with two arranged on either side of the car body, and one at the front and rear. During the framing and spot welding process, the car body is driven into

the station by an automatic guided vehicle (AGV), and is locked in place by fixtures. The robot then performs the required welding operations, and after the fixture release the AGV moves out of the station. This is an efficient means for programming robots, as approximately 60 hours were required to generate the robot programs through simulation and off-line robot programming whereas 300 hours were required when using on-line methods.

(d) Glass Cutting (Nottingham)

Nottingham University has investigated the automation of the cutting of patterns on glassware [Knight et al, 1986], by generating off-line robot programs from a CAD system. The initial implementation was interfaced to a Cincinnati Milacron T3-726 robot. Once the patterns are designed, the surface profile of each glass is measured using sensors so as to determine a bi-cubic surface patch model of the glass surface. The pattern definition from the CAD system is then mapped onto this surface patch in order to calibrate the robot program. This robot program is tailored to an individual glass and hence the whole process must be repeated for each blank glass.

The research work is continuing [Edwards and Howarth, 1988] with a replacement ASEA IRB1000 robot, and the authors conclude that a traditional highly skilled glass cutting operation can be automated and the control processes can be applied to a wide range of other industries.

## 2.6 Review of Robot Languages and Robot Simulators

The limitations of teach-mode programming have been recognized by users from industry and research institutions [Hocken and Morris, 1986]. This, together with the rapid growth and functionality developments in the electronics industry which has made computers more powerful, has resulted in the development of numerous robot languages [Van Aken and Van Brussel, 1988]. Developments in on-line robot programming languages have improved the programming environment, but suffer from a set of limitations inherited from teach-mode programming [Ambler, 1982] which are:

(i) Programming in explicit robot languages does not allow the programmer to resolve critical design and operational issues.

(ii) Updating a highly complex application with a huge database may be difficult.

In order to overcome the disadvantages of on-line programming, numerous off-line programming languages and systems of various degrees of sophistication have been developed [Hocken and Morris, 1986]. The majority of these languages and systems were developed by robot manufacturers, major industrial users, CAD suppliers and research institutions [Dooner, 1984; Boren, 1985].

Current commercial CAD systems and off-line robot simulators are generally very expensive so that companies using such tools are likely to be major users of industrial robot systems and/or robot system manufacturers [Dooner, 1984]. Seemingly and notably, the majority of

available off-line robot programming simulators are limited to a particular manufacturer's product range. Since the majority of off-line robot programming simulators are produced by the robot system manufacturers and robot users, there are obvious reasons for the limited range of applications. Firstly, they do not have sufficient knowledge of robot systems manufactured by their competitors. Secondly, they prefer to promote their own products. Thirdly, they are likely to have excellent access to and can accomplish calibration of their own robot system in developing off-line programming facilities.

A number of robot simulators have become available in the world marketplace including GRASP [Bonney, 1987], McDonnell Douglas Robotics Software [Carter, 1987], ComputerVision's Robographix [Mattis and Gill, 1988], Intergraph system [Intergraph, 1985], AutoSimulations [Stauffer, 1984] and ROBCAD [Robotics World, 1986a]. These CAD/CAM packages provide a set of modelling and simulation tools which can be used to represent a robot manipulator, and it's attendant equipment, in graphical form and hence simulate a manufacturing task. The use of such packages can allow the manufacturing engineer to try several solutions for robotic cells before purchasing any equipment. Hence these workplace design tools can be used to improve the choice and layout of robot systems, reduce set-up costs, reduce installation times and improve system performance. Certain manufacturers claim that "engineers can design and lay out robot cells up to 70 per cent faster through the use of a robot simulator" [Industrial Robot, 1982; Robotics World, 1986a].

An additional feature of many robot simulators is the availability of post-processing software for the off-line programming of robots. Such a

post-processor reformats the geometric and sequential information generated by the modeller and simulator, to produce a robot task program in the native language of the robot. Obviously this post-processing function is robot dependant (currently there is no internationally accepted neutral language for robots) and hence this facility will only be readily available for commonly used industrial robots.

The McDonnell Douglas robotic system [Carter, 1987] is generally recognised as the most sophisticated kinematic robot simulator commercially available today. However, these kinematic robot simulators do not take dynamic effects (damping and following errors etc) into consideration. This has provided the motive for researchers to generate robot simulators that can simulate dynamic effects. As a result, a limited number of dynamic robot simulators are being applied to the off-line robot programming problem. Among these are ROBOT-SIM [Novak, 1984], ROSI [Industrial Robot, 1987], and STAR [Hornick and Ravani, 1986].

There are many different robot simulators and off-line robot programming systems which have been developed for in house use or academic research. It would be impossible to gather all the information for every software package available today. It should be noted that the comparison of these simulators is a rather difficult task as the information provided by the suppliers of simulators is not presented in a uniform manner and is often only superficial in nature. Furthermore, without practical hands on experience on individual software package, a thorough evaluation is not possible. The following paragraphs describe a representative selection of robot simulators. The following classification is according to the

chronological order as mentioned in section 2.2.

### 2.6.1 Explicit - Structured High Level Robot Programming Languages

ROPS (Robot Off-line Programming System) is a simulation package offered by Cincinnati Milacron. It contains several modules specially developed for T3 700, T3 800 series, and version 3 and 4 robot controllers. ROPS is available on IBM PC-AT and DEC VAX computers, and is capable of communicating with CAD/CAM systems [Cincinnati Milacron, 1985]. This menu driven programming system provides the programmer with assistance in developing an off-line robot program. The ASEA off-line programming system [ASEA, 1986] has similar functions to ROPS. The survey papers by [Bonner and Shin, 1982], [Gruver et al, 1983], [Soroka, 1983] and [Schreiber, 1984b] reviewed some of these languages including AL, AML, HELP, JARS, MCL, RAIL and VAL.

### 2.6.2 Implicit - Model Based Off-line Robot Programming Systems

AUTOPASS is the automatic parts assembly system developed by IBM [Wesley et al, 1980]. It is a high level, object oriented, compiled language for assembly, based on a world model of the workplace. This language describes assembly operations rather than robot movements [Ranky and Ho, 1985] thereby allowing a programmer to specify assembly procedures implicitly. The compiler then transforms the assembly procedure into motion commands through the use of data from a geometric database or world model where the geometric information

(spatial relationships between objects) is stored. The programmer is allowed to use commands such as "PLACE", "OPERATE", and "RIVET" for assembly, tool functions and fastening respectively. There are other similar systems such as RAPT (Robot APT) developed within the Department of Artificial Intelligence, at the University of Edinburgh [Ambler, 1982; Ambler et al, 1982; Durham, 1985; Howe and Fothergill, 1988]. ROBEX (ROBoter EXapt) developed in Germany [Eversheim et al, 1981; Weck et al, 1981, 1984 and 1987; Weck and Niehaus, 1984]. Both RAPT and ROBEX can link up with graphics display via a pre-processor to the graphics system.

### 2.6.3 Implicit - Graphics Based Off-line Programming Systems

Graphics based off-line programming systems include CAD systems and robot simulators. As previously stated, robot simulators can be classified into kinematic and dynamic robot simulators (this being considered in greater detail in chapter 4 section 4.3.1). The Denavit-Hartenberg algorithm [Denavit and Hartenberg, 1955] is the most commonly used [Gupta, 1986] in kinematic robot simulators. These kinematic robot simulators do not take dynamic effects into consideration. Dynamic robot simulators can only produce a dynamic model which closely approximates to certain dynamic characteristics of a given robot. A kinematic model also forms part of the essential framework of a dynamic robot simulator. The frequently referenced algorithms in the context of dynamic simulation include the Lagrangian method [Murray and Neuman, 1984; Wang and Kohli, 1985] and the Newton-Euler method [Khosla and Neuman, 1985; Featherstone, 1987].

## (a) CAD Systems

This involves the use of common CAD systems. These systems do not have the capability of modelling robot kinematics nor dynamics, and can only produce 2D or 3D designs of products. Typical examples are to be found in the Department of Production Engineering and Production Management, at Nottingham University, where a CAD system is used to design glass patterns. The output of these patterns is used to programme a robot in carrying out glass cutting [Knight et al, 1986]. Similarly, another CAD system has been applied for off-line programming of painting robots at the Computer and Automation Institute, Hungarian Academy of Sciences [Klein, 1987].

## (b) Kinematic Robot Simulators

### (1) AutoSimulation system

The AutoSimulation suite includes AutoBots, AutoMod, AutoGram and InterFaSE modules [Robotics World, 1986b]. The AutoBots module allows robot simulation and off-line programming. The InterFaSE module is a simulation-base factory scheduler which allows the user to test and change operating and decision making rules to optimise the performance of facilities.

AutoMod is a numerical simulation package based on the GPSS simulation language, whereas AutoGram works in conjunction with AutoMod to produce graphic representations [Stauffer, 1984]. The AutoMod simulation package uses English language inputs thereby

allowing programmers to describe robotic devices and associated manufacturing system elements. From this a simulation model is constructed. Subsequently the model data is utilised by AutoGram, in providing a graphic display.

## (2) CATIA

The Computer Aided Three dimensional Interactive Applications (CATIA) package [Crosnier and Fournier, 1987; Forestier, 1985] was developed in France by Dassault Systems. It consists of five modules:

(i) a wireframe module providing 3D geometric definitions.

(ii) surface descriptions used to define complex 3D surfaces and volumes, together with NC machining capabilities.

(iii) polyhedral solids used to define simple volumes or solids.

(iv) kinematics to define 2D joint mechanisms.

(v) a robotics off-line programming system.

CATIA runs on IBM workstations and can be used to communicate with the MCL (Manufacturing Control Language) language developed by McDonnell Douglas [Gettelman, 1985 ].

## (3) GMF system

The GMF off-line programming system has several analogies with NC part programming languages in that a set of S and G codes may be used to construct robot programs. Alternatively English-like mnemonics may be used to indirectly specify the S and G codes. It is difficult for the programmer to input point data since input is required in joint axis form [Jacobs, 1984].

## (4) GRASP

GRASP is the acronyn chosen for the General Robot Arm Simulation Program developed at the Rensselaer Polytechnic Institute in the United States (and should not be confused with the identically named software used in connection with this research study). It can model robot arms with up to six joints, describe robot tasks and evaluate proposed robotic systems by providing animation in wireframe representation. The capabilities of the simulation program include cycle time estimation, joint violation detection, along with torque and member bending simulation. The data generated by GRASP [Derby 1984a, 1984b] has been translated into VAL programs for the PUMA and into a language specially designed for the Cincinnati T3.

## (5) HERON

HERON is described as a stand-alone CAD/CAM workstation [Miller, 1985], which allows the programmer to perform workcell design and

optimization, off-line programming of robots, animation and collision detection. The system provides 3D wireframe models and 3D solid models. It consists of six modules:

(i) ROBOLIB provides libraries of available robots and accessories.

(ii) ROBOGEO allows geometric modelling and mechanism design.

(iii) ROBOSIM facilitates workcell design, task description and animation.

(iv) ROBOLOAD translates a task description into a robot program and downloads it to the specific robot controller.

(v) ROBODOC produces drafts and documents.

(vi) ROBOPERT provides project management analysis.

## (6) IGRIP

IGRIP (Interactive Graphic Robot Instruction Program) was developed by DENEB Robotics. IGRIP claims to be computer system independent, being designed in modular form to provide flexibility and enabling the software package to be ported from one machine to another through changes to two of the modules [Schreiber, 1984b; Harrison and Mahajan, 1986]. There are four graphics modes and thus graphics models of different devices can be displayed [Yoffa, 1988] in any of the following forms.

(i) wire frame (which is the fastest mode).

(ii) wire frame with hidden lines removed (which can avoid ambiguity).

(iii) a simple shading mode (which provides a more realistic representation).

(iv) sophisticated shading ( this being the best but slowest).

IGRIP also provides display modes at three levels: world, device and link. Simulation control is specified via GSL (Graphical Simulation Language) which is device independent [Robotics World, 1986c]. Interestingly, IGRIP allows the user to enter or load programs written in an actual robot language or code such as VAL II and KAREL. This is then converted to GSL for use in the graphical simulation. Hence it possible for example to verify whether an existing program is correct in different workplace arrangements, or for calibration purposes. IGRIP displays the maximum and minimum reachable workspace of a robot as a transparent shell around the robot. It also offers automatic collision detection which runs concurrently with the animation.

## (7) INTERGRAPH

INTERGRAPH robotic software was originally developed in conjunction with GMF Robotics. It supports robot programming via two types of workstations which include on-board processors with significant

memory storage area, thereby providing display generation capabilities. This capability can be used to significantly enhance modelling responses and improve real time animation. The off-line programming of a robot is divided into 5 phases [Intergraph, 1985; Kacala, 1985].

(i) operations planning and definition: this is a group of libraries of definitions including robots and accessories;

(ii) workplace composition which enables programmers to design workplace layouts;

(iii) process simulation, editing and verification: the robot task program can be simulated and if any error is detected during simulation (including joint violation, collision and program sequence) it can be modified.

(iv) output program: verified robot task program output can be translated into a robot program or an engineering drawing; and

(v) process feedback and workplace calibration: the operating conditions measured during the actual real world performance are fed into the system, and a simulation is performed which calibrates the workplace to produce a more precise factory environment model.

## (8) McDonnell Douglas Robotics Suite

McDonnell Douglas's graphic simulation package was designed for off-line robot programming and comprises four modules called PLACE, BUILD, COMMAND, and ADJUST [Howie, 1984; Haffenden, 1984; McDonnell Douglas, 1986].

(i) BUILD as its name implies, allows users to build their own robot models which may be configured in some specialised way. This module allows the analysis of user defined robot devices of up to six degrees of freedom.

(ii) PLACE (Positioner Layout And Cell Evaluation system) is a simulation tool for designing and evaluating robotic cells in 3D with smooth motion animation.

(iii) COMMAND is a module for the creation of robot off-line programs based on PLACE sequences. Off-line robot programs can be generated in a format suitable for Unimation, Cincinnati T3 and GMF robots.

(iv) ADJUST is robot independent calibration software which allows errors between the CAD model and actual robot system to be adjusted.

PLACE accepts data directly from UNIGRAPHICS or via an IGES interface from other CAD modellers.

## (9) ROBCAD

ROBCAD implements 2D and 3D wireframe models or colour shaded solids, and the IGES standard is implemented to interface with general purpose engineering CAD/CAM systems. ROBCAD includes powerful "canned" cycles and subroutines for specific applications like welding and palletising, which can be selected from a menu of operations [Adler, 1986]. In 1986, the developers claimed that ROBCAD was the only system in the world [Robotics World, 1986a] which could produce concurrent parallel shaded simulation.

## (10) ROBOCAM

The ROBOCAM simulation package [Craig, 1985] was developed by SILMA, Inc. and offers two programming methods, one of which uses a high-level robot language called RISE, whereas the other uses a graphical approach via menu control. Models can be saved in files and transferred between ROBOCAM and other CAD systems through the use of IGES interfacing. After a RISE program has been generated and debugged by simulation, the program can then be translated into a specific robot language by the use of an intermediate language called RCODE.

A special feature of ROBOCAM called "AUTOPLACE" can assist the programmer in assessing the performance of different robots in achieving the same task. By inputting all the important points which are necessary to be reached, AUTOPLACE will suggest a location in the workplace for the chosen robot.

## (11) ROBOGRAPHIX

The ROBOGRAPHIX package is a product of Computervision, a major supplier of CAD/CAM systems, and has four major functions [Gondert, 1984; Mattis and Gill, 1988]:

(i) design and build the workplace models.

(ii) create robot programs.

(iii) verification of robot programs.

(iv) post-processing and output of robot programs to actual robots.

The package includes a library of three dimensional models of standard robots and accessories. Users can expand the library by building their own robots and equipment. Since the CAD/CAM system database and the ROBOGRAPHIX library are integrated, users can easily manipulate information between the two. A robot language processor within ROBOGRAPHIX translates robot programs into specific robot programming languages such as VAL for Unimation robots, RAIL for Automatix and code for Cincinnati robots. A distinguishing feature of ROBOGRAPHIX is the use of an electronic pen and digitising tablet to specify point to point motion for the robot on the graphical terminal.

## (12) RoPL

RoPL (Robot Programming Language) was developed by E.S-I Incorporated in New York. It provides wireframe display on 16 or 32 bit microcomputer systems with sophisticated capabilities such as hidden line removal, zooming and angular viewing, collision avoidance, and joint violation detection [Price, 1984]. Furthermore, this system provides looping constructs, macro capability and user defined input/output devices. It is written as a menu driven graphic system, comprising of several sub-menus which guide the programmer. Animation, where needed, can be shown with two different views concurrently.

## (13) WRAPS

WRAPS (Welding Robot Adeptive off-line Programming) was developed in the Department of Manufacturing Engineering at Loughborough University of Technology. The system is implemented on microcomputers (IBM compatibles) as a relatively low cost simulation system [Goh and Middle, 1985] and is specifically aimed at the off-line programming of robotic arc welding tasks. This system is also integrated with an expert system [Middle and Goh, 1987] which is responsible for welding procedure selection and optimisation.

## (14) WSO System

The WSO system [Marklund, 1986] is a 3D CAD/CAM system designed for robotic layout design and off-line robot programming. There are two types of WSO system available:

(i) general purpose CAD/CAM systems with WSO modules which have a common database available for both product development and production engineering.

(ii) dedicated WSO systems which are more likely to be able to support the real time graphics that are needed for simulation.

The system consists of libraries of robots, accessories and peripheral equipment. It can communicate with other systems for receiving CAD geometric data, and drafting and dimensioning for documentation. It has a language post-processor which translates output to the ASEA ARLA language. A calibration function can be used for adjustment of the robot program for any discrepancies between the idealised model and the real installation. If absolute accuracy is required, a TEACH function can be used to upload reference positions to be included in the off-line programs.

## (15) GRASP

GRASP [Bonney et al, 1984; Bonney, 1987] stands for Graphical Robot Applications Simulation Package and was derived from SAMMIE (System for Aiding Man-Machine Interface Evaluation) [Case et al,

1986 and Porter et al, 1986]. SAMMIE and GRASP were developed in the Department of Production Engineering and Production Management at the University of Nottingham. The Nottingham group [Dooner, 1984] developed GRASP with funding from the Science and Engineering Research Council and subsequently formed a company called BYG Systems.

GRASP allows kinematic modelling of serial link manipulators and complex joint mechanisms forming specific manipulator structures. Since there is no general solution in forming kinematic models of complex joint mechanisms, the modelling of non-standard configurations requires the assistance of BYG Systems in writing specific Fortran subroutines.

Once the world model has been built using typical solid modelling techniques, there are two ways of achieving robot/workplace simulation viz :-

(i) high level robot textual programming to describe geometric and sequence information.

(ii) by graphical interactive programming.

After an event sequence (or track) has been created, the programmer can generate a time dependent continuous motion simulation known as a Process.

Animated motion can be displayed in a variety of views. Furthermore, the display can be shown in projection mode with either first or third angle projection being selected.

GRASP was originally designed as a powerful CAD simulation tool for workplace design rather than as an off-line robot programming tool. GRASP is continuing to be developed and is now being extended to cope with off-line robot programming. This has been achieved by writing post-processors for the GRASP output, converting this robot independent output into a specific robot language or codes (including VAL, VAL II and ASEA AR-Language).

## (c) Dynamic Robot Simulators

### (1) ROBOT-SIM

Calma has introduced the ROBOT-SIM simulation software package for robot workplace design and evaluation. The software permits robot and workplace design, robot motion programming, cycle time estimation and dynamic properties simulation of the robot. Dynamic data, including velocities, accelerations, link inertias and motor torque characteristics are input to the simulation process thereby offering facilities to determine robot performance. However, this is restricted to robots with a maximum of six degrees of freedom. With a specific robot pre-programmed desired path, the simulation package may be used to predict the actual path of the robot arm (i.e. path tracing error, overshoot error etc are predicted).

Principal capabilities [Imam et al, 1984; Novak, 1984] of the simulation include:

(i) kinematics and dynamics of a multiple link robot arm (comprising revolute and prismatic joints),

(ii) models of the robot drive systems, including inertia and torsion and bending effects in motor transmission elements,

(iii) digital controls for individual joint servo loops, including finite sampling rate and amplitude quantization effects,

(iv) algorithms used for coordinate transformation, path interpolation, and to provide special programming features such as rough positioning, dwells, and smooth decelerations.

## (2) ROSI

RObot dynamic SImulator (ROSI) was developed in the Department of Artificial Intelligence at the University of Edinburgh and is marketed through Cambridge Controls Limited. Cambridge Controls claim that ROSI uses highly efficient Walker-Orin recursive Newton-Euler algorithms [Hemami et al, 1975]. This system has no limitation on the number of degrees of freedom that can be modelled, and comprises two major parts:

(i) a comprehensive dynamics engine

This is a library of functions for constructing robot models and

performing dynamic computations [Industrial Robot, 1987]. It is designed specifically for use as a component in a larger software package. This is system independent module and can be used for custom built software.

(ii) a user interface for communication between the dynamic engine and the programmer in graphical form.

Libraries are available storing information relating to joint types, electric and hydraulic parameterised actuator models, and control systems parameters.

## (3) STAR

STAR (Simulation Tool for Automation and Robotics), was developed for off-line robot motion planning and programming. As STAR is not a CAD program [Hornick and Ravani, 1986], the graphic shape of the robot can not be developed. However, it is equipped with a CAD interface, which allows animation of solid geometric models of objects generated on the GMOS solid geometric modeller system. This simulation software package consists of four basic modules:

(i) an input module for model building;

communication between the programmer and the software takes place at a conversational level through the use of a high level input language.

(ii) a mathematical module for kinematic and dynamic calculations;

this module automatically formulates kinematic and dynamic equations and uses an iterative procedure to solve the inverse kinematic equations.

(iii) a trajectory planner;

this module is claimed to calculate optimal trajectories so that cycle times can be minimised.

(iv) an interactive motion planning, programming and editing environment.

The programming of robot motion is assisted by the use of 3D interactive computer graphics and a high level programming language. At the operational level, robot motions are implicitly defined by specifying motion operations that are to occur between the objects being manipulated. Thus for example, it is possible to issue commands such as "MOVE PEG TO HOLE", in a way which is very similar to AUTOPASS and RAPT.

STAR can display multiple views of the robot and its environment, and can visually detect any potential interferences between the robot and objects in its environment.

## 2.7 Limitations of the Present Generation of Robot Simulators

Here we consider three key functional areas in which the present generation of robot simulators demonstrate limitations, these limitations reducing their effectiveness and widespread usage. These functional areas are "calibration", "integration" and "ease of use".

### (a) Calibration

With off-line robot programming, the use of the robot as a digitiser is lost, and the robot end-effector may not be positioned at the location as commanded (i.e. in operation, performance is dependant on accuracy rather than repeatability). This results in a need for standardised calibration procedures. Generic robot calibration procedures have yet to evolve in an internationally accepted sense, resulting from the complexity involved in accurately measuring the position and orientation of workplace elements. Workplace calibration is undertaken to ensure that errors between the robot simulator and the real robot system are accounted for. The need for error correction procedures and tools is vital before language programming can be use in isolation, i.e. without the need for a teach pendant. A major theme in this research study has been to investigate various error correction methods, the detail of which is discussed in chapter 7.

### (b) Integration

Robot simulators are still largely isolated systems. They should be integrated into CAD/CAM facilities so that existing CAD databases can be

used to speed up the programming process. The robot simulator should also be capable of providing integration with expert systems for the planning of robot tasks. The activities of a robot simulator has been integrated with a PCB CAD design package so that previously created PCB layout information can be used in creating product models in the subsequent simulation of robot insertion of various electronic components. The integration methods studies have also addressed the inclusion of process planning activities. The integrated pcb design, planning, simulation and programming system for pcb assembly demonstrates novel features. The principles involved and actual implementation is described in chapter 8. The principles so described can be widely extended in defining various product realisation activities. The principles can also be applied in various industrial domains with the overall goal of reducing the design to manufacture time involved in creating new products.

(c) ease of use

Robot simulators may be difficult to use and are becoming so complex that sometimes very simple tasks are difficult to program. A friendly user interface is required together with facilities for computer assisted solid modelling and robot task generation. This is similar to parametric programming which is used to ease the task of generating models of families of parts in mechanical design. Chapter 6 illustrates the principles involved and describes the implementation attempted.

# CHAPTER THREE

# GENERALISED FEATURES OF FUTURE ROBOT PROGRAMMING SYSTEMS

## 3.0 Introduction

Many of the simulators described in the foregoing chapter were constructed in modular form and offer the common operational features that are essential for robot simulation. However, other features included are different from system to system. Generalised features of a future robot programming system are discussed to provide a backcloth to subsequent observations and recommendations. A knowledge of generalised features is important so that standards can be developed and applied to robot simulators such that post-processing effort can be reduced and the versatility of new robot simulators can be improved. Furthermore, the simulation model can be transferred from one simulator to another, and this is particularly useful for a CAD/CAM service house to provide a conversion facility for different robot simulators used at customers' design offices.

## 3.1 Conceptual Robot Simulation Systems

When considering features of the next generation of robot simulators we should not be bound by conventional approaches nor necessarily consider the constraints imposed by available processing power. For example, future physical implementations of robot simulators may be distributed and thus cross the conventional off-line/on-line boundaries. Thus we will consider logical robot simulators to describe the logical functionality required from future systems. The logical system will be distinct from the physical system which will reference the physical implementation techniques and enabling technologies involved. Figure 3-1 illustrates the possible features of a logical or conceptual robot simulator deduced by the author through referencing existing systems and

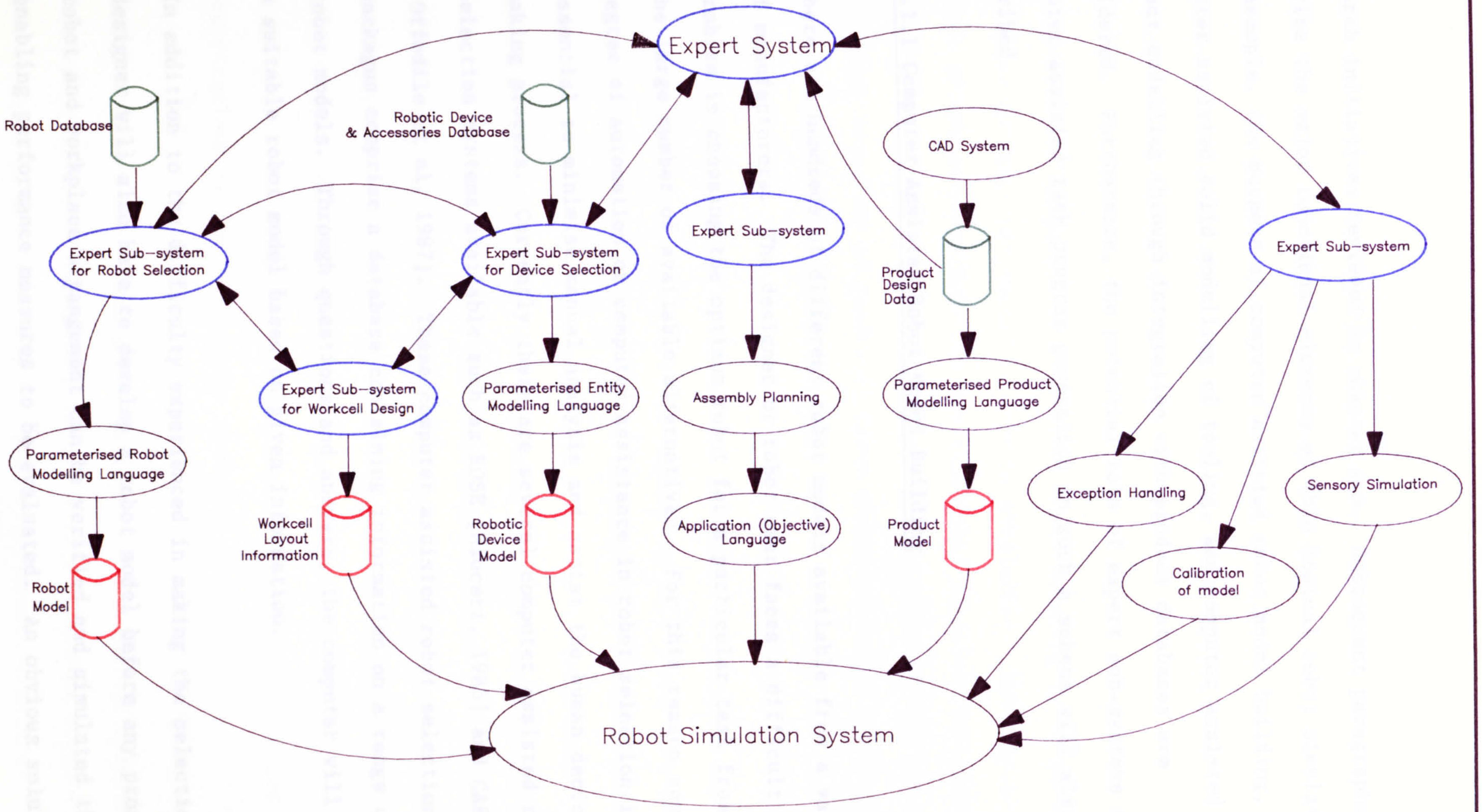



Figure 3–1 An example logic robot simulation system

research initiatives reviewed in chapter two. Subsequent paragraphs describe the major functional elements of this logical robot simulator. For example, the purpose of computer assisted robot model building, computer assisted solid modelling of tooling, and computer assisted product modelling through integration with product databases are considered. Furthermore, the potential role of expert sub-systems and computer assisted task program generation in such a schema will also be described.

### 3.1.1 Computer Assisted Robot Model Building

There are hundreds of different robot models available from a variety of manufacturers. The designer or robot user faces a difficult problem in choosing the optimum robot for a particular task from among the large number of available alternatives. For this reason some degree of automation or computer assistance in robot selection is essential to minimise manual analysis and assist the human decision making process. Currently there are several computer assisted robot selection systems available such as ROSE [Mauceri, 1985] and CARSP [Offodile et al, 1987]. These computer assisted robot selection packages comprise a database containing information on a range of robot models. Through questions and answers, the computer will select a suitable robot model based on given information.

In addition to the difficulty experienced in making the selection, the designer will also have to develop a robot model before any proposed robot and workplace arrangement can be verified and simulated thereby enabling performance measures to be evaluated. An obvious solution is

through making a library of robot models available which, although possibly restricted to the most common and widely used robot models, could be extended as needs dictate. Computer assisted robot modelling software is desirable, to provide a proven and consistent method of adding information to the library.

The purpose of computer assisted robot model building is to partially automate an important aspect of robotic simulation. The concept is based on the existing computer assisted robot selection packages, which allow the user to choose an appropriate robot for the job. A CAD robot model can be generated based on the selection of robot model giving access to the appropriate part of the database.

Computer assisted robot selection can be based on available expert system technology. Expert systems applied in this field have three basic elements: the control system (questions and decision rules), an inference engine and a knowledge base. The way in which such knowledge is accessed may vary depending upon the requirements of the user. At the present time, the inference engine and the content of the knowledge base are different from system to system. The contents of almost all of the existing robot databases are insufficient and inappropriate for use in computer assisted robot modelling [Loucopoulos and Champion, 1988]. Therefore some additions and modifications to the information stored is necessary.

Present systems have neither standard decision rules nor standard database structures [Van Assche et al, 1988]. The wide acceptance of this concept is only possible if standard database structures are

evolved [Hayes-Roth et al, 1983; Offodile et al, 1987; and Cronk et al, 1988]. The additional data required as part of the database should include parameters of robots such as the type of joints (revolute or prismatic), geometric relationships between links (common normal, offset, twist angle and rotation angle) together with the limits to motion. Some of the requirements are the same as the input requirements for CAD robot modelling and are discussed in chapter four. Due to the wide variety of possible geometric shapes for links, it is impossible for any system to automatically generate CAD models [Szeto and Lichten, 1985]. However, it is possible to synthesise the robot body by features (i.e. a method that allows user to choose from an existing library of geometric shape of robots) [Dudko et al, 1984]. The shape of the robot is not as important as its kinematic structure and equations.

### 3.1.2 Computer Assisted Solid Modelling of Tooling

As new robot models are continually introduced, along with models of ancillary equipment (e.g. grippers and tooling), the designer will also experience difficulties of building models as earlier discussed. Additional complexity will also be involved if data structures are to be entered into databases so that computer assisted procedures can be utilised.

### 3.1.3 Computer Assisted Product Modelling Through Integration with Product Databases

In a simulation, the inclusion of product descriptions can also play a very important role. Information concerning the product specification may be obtained through access to product databases, and ideally it should not be necessary to duplicate existing information. The modelling of the product could in fact be achieved, through use of expert sub-systems which could for example synthesise products by features. The efficiency with which computer assisted solid modelling can be achieved will be dependant of the existence and use of standards which can enable the timely and reliable sharing of information between various manufacturing entities.

### 3.1.4 Potential Role of Expert Sub-systems

Future expert sub-systems will perform various functions such as robot selection, the definition of optimum layout solutions, assembly planning, calibration of models and generation of exception handling conditions. In achieving this functionality the 'experts' must reference data structures stored in databases which model the robotic devices, their workplace elements, product descriptions, process and production planning information and state history representations of the robot and its workplace sensors and tools. Again it is important to stress that such facilities can be viewed from both a 'logical' and 'physical' viewpoint. However, whatever the physical implementation this will imply the use of standardised data structures and information transfer mechanisms. The reader should again refer to

figure 3-1 which illustrates the logical interaction between sub-systems when accomplishing robot selection and designing workplace layouts. Similar logical interactions could be constructed for assembly planning and exception handling.

### 3.1.5 Computer Assisted Task Program Generation

The discussion so far has centred upon computer assisted (CAD) modelling and robot simulation. Robot task simulation will have an increasing role in predicting system performance and identifying potential problem areas thereby creating more directly useable programs. Let us consider the potential role of expert systems in automating or semi-automating the simulated execution of task programs.

Using today's commercially available simulation tools the task programs will be stored and executed using the syntax of proprietary simulation systems. Obviously standard data structures for simulation systems are desirable, but, currently there are no standards which can be applied appropriately.

Having created the robot workplace and product models according to an optimised layout design, the simulated robot task should be verified. Discrepancies between theoretic and measured models should be accounted for. Any problems should be identified and modified before the simulated task program is post-processed into a robot language which can be executed. Again, ideally a standardised or neutral language should be adopted here so a given simulation system can be

widely applied without the need to generate bespoke post-processors for each robot model.

## 3.2 Conceptual System Mapped onto Reality and the Role of Existing and Emerging Standards

Here we will consider various standardisation initiatives which should be referenced when mapping logical robot simulators onto physical computer hardware and software.

### 3.2.1 Product Design Data Format

There are several related standards which currently address problems in product data transfer [Owen and Bloor, 1987].

(i) IGES (Initial Graphics Exchange Specification) developed under US Air Force ICAM programme.

(ii) PDDI (Product Definition Data Interface) is another research project initiated by the US Air Force ICAM Project.

(iii) XBF (Experimental Boundary File) developed by CAM-I (Computer Aided Manufacturing-International).

(iv) PDES/STEP (Product Data Exchange Specification) developed by the CAM-I as a follow up standard to IGES.

(v) EDIF (Electronic Design Interchange Format) was developed under

the user initiatives to form standard for the exchange of electronic CAD design data.

(vi) SET (Standard d'Echange et de Transfert) developed by Aerospatiale in France.

(vii) VDA-FS (Verband des Automobilindustrie Flachen-Schnittstelle) developed under the initiatives of the Association of German Car Manufacturers.

The Initial Graphics Exchange Specification (IGES) [Wilkinson and Hallam, 1987] is a neutral data format for describing drawings and more recently solid models. SET is a French national standard which provides more compact data structures than IGES, and SET has provision for transferring information relating to curved surfaces. Future versions will enable finite element modelling data, solid models, polyhedral models, schematics and NC paths to be transferred.

A standardisation initiative known as the Product Data Exchange Specification (PDES) [Smith, 1987] seeks to build on IGES and the Product Definition Data Interchange (PDDI) model [Owen and Bloor, 1987]. Furthermore the recent amalgamation of PDES and STEP should lead to an ISO standard for the part description in the not too distant future. This standard also seeks to identify methods of utilising product design data in various manufacturing activity areas.

### 3.2.2 Robot Program Data Format

In West Germany considerable effort has been directed towards defining interface standards for robots, and has been referred to as the Industrial Robot Data (IRDATA) [Weck et al, 1984]. The IRDATA concepts are based on the existing CLDATA (Cutter Location DATA) standard for numerically controlled machines. IRDATA includes descriptions of robots, tools, sensors, working space, frame lists, motion specifications and execution criteria, arithmetic and boolean operations, program flow control and input/output operations. Various data types are included, such as, integer, real, boolean, character, string, pointer, vector, orientation and joint angle. IRDATA text comprises a sequence of records of unlimited length, each record consisting of at least 2 and at most 125 words of 6 characters.

The user does not programme in IRDATA but in some other high level programming language. Subsequently IRDATA is used as the interface message format between the off-line programming system and the robot controller. There has been considerable support for IRDATA [Rembold et al 1985, 1987] although standardisation in this area has been superceded to some extent by MMS (see sub-section 3.3) and its robot companion standard [Rembold, 1988]. It may be that IRDATA will not stand the test of time with changes in enabling tools and the emergence of other more generally applicable and widely supported standards [Rembold, 1988] such as MMS and its robot companion standard representing an advancement towards more comprehensive and widely adopted data structures.

Since 1981 in Japan, standardisation of robot programming has been encouraged [Arai et al, 1985] by a committee of the Japanese Industrial Robot Association. The committee was originally formed from working groups, initiating surveys into existing robot languages. Each working group has been responsible for a specific activity, such as robot language function, protocol, and language syntax. As the result of these activities, in 1984 the committee proposed a generic set of functions for robot controllers along with associated function codes which introduce notation in a robot language called STROLIC (STandard RObot Language in Intermediate Code) [Arai and Matsumoto, 1983]. STROLIC can be considered to be a generic robot instruction set, represented by binary codes. The main reason for adopting binary code was to improve the machine-machine interface. Since it is not convenient to programme in binary codes, a programming language called STROL (STandard RObot Langauge) was proposed as a programming language associated with STROLIC. The main programming capability is similar to that of AML (A Manufacturing Language) [Gini and Gini, 1985], but stack operations and program flow control are inherited from FORTH [Toppen, 1985]. Each function has an associated update routine which makes debugging, tracing and teaching easier. Motion and sensor commands are defined with pointers to appropriate registers relating to position, velocity or sensor information. Various data structure types are available including arrays, strings, records, etc.

Although both IRDATA and STROL/STROLIC are being used within their respective host countries there is no evidence that they are being adopted internationally.

## 3.3 Integration Architectures

Systems integration methods are evolving on worldwide basis to create more responsible manufacturing systems [Albus, 1984; O'Grady, 1986; Jones and McLean, 1986; and Weston et al, 1988].

A major problem in this area is the need to account for the differences between proprietary machines. Integration of a stand-alone machine with other manufacturing entities can be achieved through making the machine conform with the Virtual Manufacturing Entity (VME) concept [MMS draft 6 document, 1987]. Essentially this concept involves making a proprietary machine (such as a robot) act like a standard machine when viewed by external manufacturing entities. Thus for a robot, integration is implemented through establishing a physical connection and appropriate protocols which enable the transmission of a standard set of robot "words" or "messages" (including programs and information) from one machine to another. The approach can enable remote operation of the machine and the transmission of information, as shown in figure 3-2.

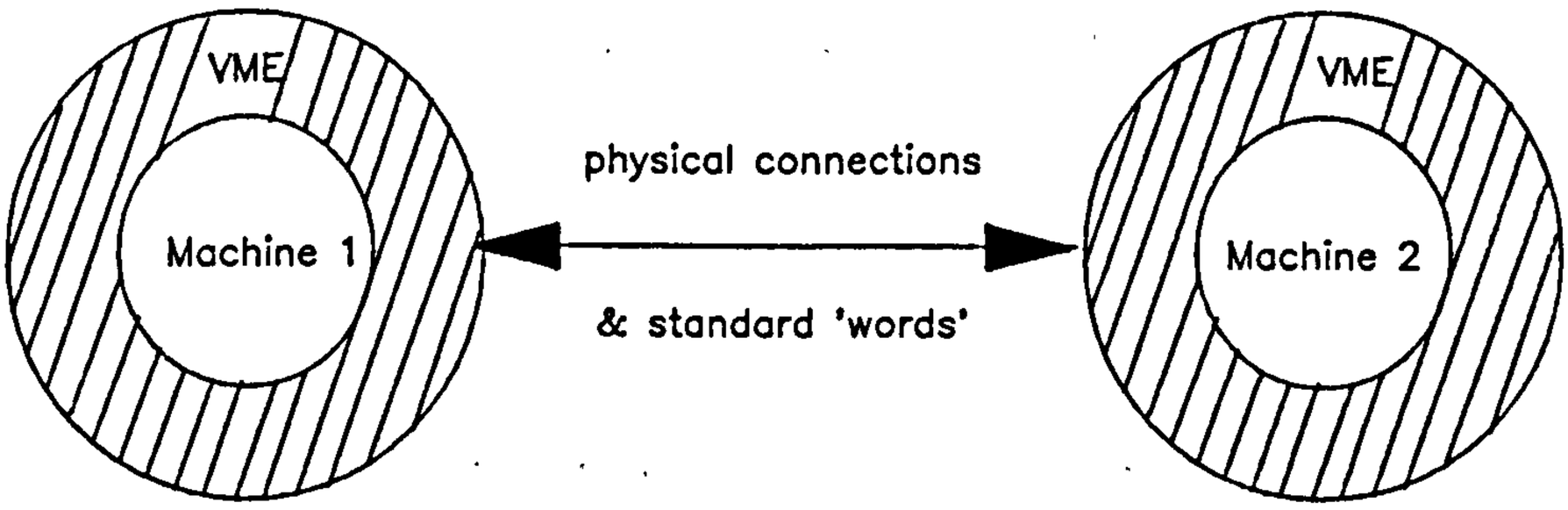



Figure 3–2 Integration through physical connections and protocol

In manufacturing industry, there have been a number of different solutions to integration problems, both proposed and implemented. Some representative physical connections include the use of an RS232 data link [Putman, 1987] running proprietary (non-standard) protocol. Such an approach yields bespoke solutions and high integration costs [Weston et al, 1988]. The use of MAP/TOP [Hollingham, 1986; Weston et al, 1986 and 1987a; and McGuffin et al, 1988] interfaces, which represents an internationally accepted standard mechanism (embodying the VME concept), is becoming commonplace and overcomes certain of these integration problems. However both approaches provide a physical connection which merely provides the capability of transmitting data from one machine to another, and there is a fundamental difference between data and information transfer. Although, data storage and manipulation on machines may be identical, the information represented by these data are not necessarily the same. This means that data may be successfully transferred from one machine to another but the recipient machine may not understand its meaning. Thus standardised techniques can be devised for the automatic storage and dissemination of information [Rui et al, 1988].

Here we describe two different types of integration : hard integration and flexible integration. Almost invariably today's factory integration schemes incorporate 'hard' integration with its software being written in accordance with some previously defined specification even if the physical connection is based on OSI protocols such as MAP/TOP. This approach is specific to the manufacturing entities involved. Conversely, manufacturing entities in flexible integration will interact in a reconfigurable manner whereby changes in product, production processes and resources and manufacturing organisation can be accommodated. To

facilitate flexible integration a generic framework is necessary. This requires three inter-dependant architectures (see figure 3-3):- namely the Application, Information and Network architectures [Weston et al, 1988].

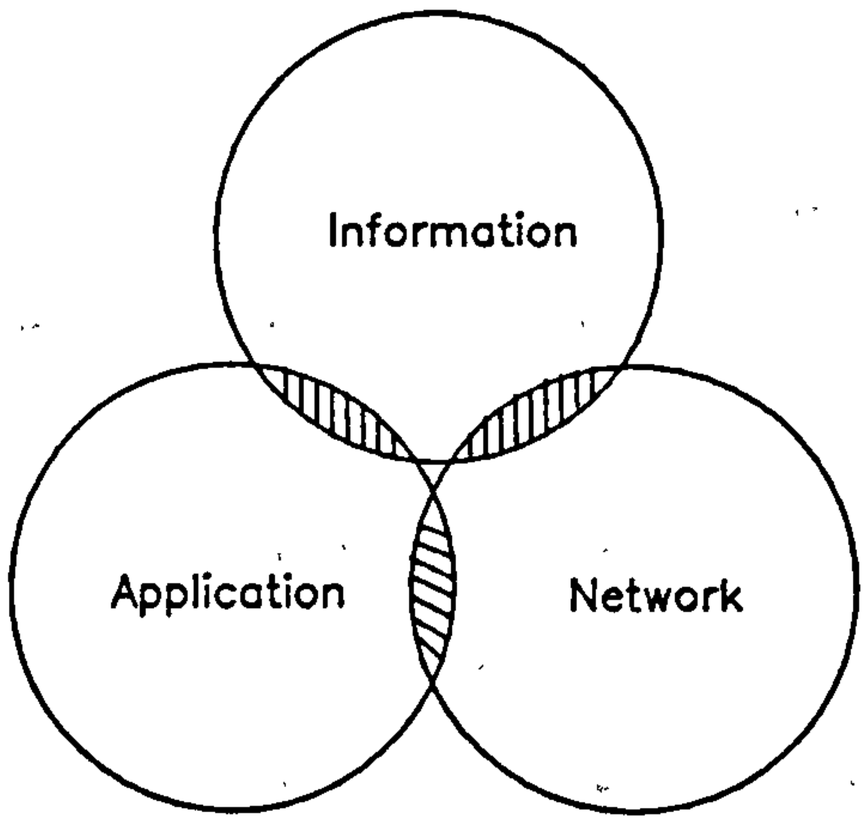



Figure 3–3 A generic framework for flexible integration

The application architecture contains action (manufacturing) causing statements which define the interactions between different machines on a shopfloor and with their organisational planning and control systems. Certain problems of flexibly creating this architecture have been addressed in AUTOMAIL [Weston et al, 1987b], and the Manufacturing Message Services (MMS) provided within the application layer services defined by the MAP/TOP specification.

The information architecture is concerned with the control of access and sharability of information, i.e. a library of manufacturing information. Access to this library must be controlled ideally by a distributed

database management system which can provide controlled access to the data distributed across the whole CIM system. An example of this kind of work is the information administration system evolved as part of the US Air Force ICAM programme. This system is known as the Integrated Information Support System (IISS) [ICAM, 1983]. Another example is the initiative of the US National Bureau of Standards (NBS) - an Automated Manufacturing Research Facility (AMRF). The aim of this initiative was to study the hierarchical modelling of the future factory, the standard interfaces between component systems and the integration of distributed databases [Su et al, 1986]. This work has resulted in the specification of an Integrated Manufacturing Data Administration System (IMDAS). Recently work at Loughborough University by Rui [Rui et al, 1988] has enhanced these concepts in creating a specification of an information adminstration system for distributed heterogeneous databases.

The network architecture is responsible for the transmission of information and is primarily based on physical connections, and protocols (e.g a set of "words" relating to classes of manufacturing entities such as robots, PLCs and NC machines). Various implementations of the ISO/OSI reference model such as the broadband MAP, TOP, and MAP-EPA will provide a limited range of widely accepted network architectures. The top-most layer of the seven layer ISO/OSI reference model is the application layer which provides support functions for the users application software so that virtual connection between distributed computers can be achieved. Part of this philosophy is the evolving MMS specification which has companion standards which will define the "words" for manufacturing entities. With respect to robots, it is not clear as to the eventual extent of the robot companion, but draft 6 of the MMS specification

includes standard high level access protocol for starting, stopping, and downloading robot task programs along with a protocol for reading and writing variables associated with robot control and monitoring functions. There is some overlap of functionality between MMS and IRDATA but generally MMS can be considered to provide an integration protocol whereas IRDATA is mainly concerned with robot task program formats.

In the context of off-line robot programming, the robot programs should be generated in a standard format and transferred to the target robot via a standard robot words and connection mechanisms (the network architecture). The manufacturing "sentences" (formed from an association of "words") will define the way in which manufacturing entities interact to create products and will be created by the application architecture. The robot information will form part of library of manufacturing information which will be administered by the information architecture.

In this thesis, the author has aimed to generate generic solutions to robot simulation problems while referencing standards development which can yield flexible implementations of the logical systems derived. However, it has been necessary to utilise available software and hardware sub-systems (e.g. GRASP, RACAL REDBOARD, ADEPT ONE robots) which were not designed with integration in mind. Thus the solutions implemented represent a mapping of the logical system onto specific hardware which provides knowledge which can enable more generic and flexible solutions to be derived in the future.

## 3.4 Conclusions

Computers are becoming more powerful with faster processing capability which allows robot simulation to be animated at much faster speeds approaching real time animation. Simulation has been successfully applied in the workplace layout design and the evolution of many complex robot and robotic systems.

As robots are becoming more sophisticated with more degrees of freedom (and hence dexterity), more complex sensory and tooling systems and improved kinematic control, more complicated robot tasks are applied. As robot tasks become more complex and hence also the programming, the effectiveness of robot simulation and off-line robot programming are becoming apparent. The evolution of CAD/CAM integrated with robotics and incorporating computer assisted modelling and programming facilities, makes further automation in the design and programming phases possible. Together with this the distributed management databases, simulation neutral data format, robot program data format and communication standards, considerable enhancement of CIM systems can be achieved.

# CHAPTER FOUR

# ROBOT SIMULATION SYSTEM ARCHITECTURES

## 4.0 Introduction

Methods of simulating robots as an integral part of CADCAM systems will represent a major step towards the planning and designing of manufacturing systems. Simulation techniques offer a very powerful tool which can be used for designing manufacturing cells from initial conceptual design, right through to detailed design and actual implementation in the factory.

The word "simulation" has been used widely in manufacturing industry, and it is necessary to make a definitive statement as to the use of the term "simulation" in this thesis.

Computer simulation can be categorised into two levels [BYG, 1987]. The highest level, usually referred to as the strategic planning level, is used to evaluate alternative factory strategies, determine the required number of manipulators or machining stations to carry out a particular task and so on. ECSL [ECSL, 1986], HOCUS [P-E publication, 1986], and SIMAN [Pegden and Ham, 1982] are commonly used examples of such systems of which there are many [Miller, 1985]. The simulation output normally consists of performance measures such as reliability, machine utilisation, throughput times, work in progress, buffer queue performance and product arrival pattern contingencies. It is not possible to use these systems to assist in the design of individual robot/machine workplaces or to evaluate how robotic systems will behave.

This is where a second and more detailed level of simulation is required. This type of simulation makes use of powerful 3D computer graphics to help engineers to design installations and in the case of robotic systems to

evaluate the interaction of robotic devices and other devices within the workcell. Detailed graphical simulation can be used for robot workcell design and thereby enable feasibility studies to evaluate certain robot attributes and to predict how a system will perform. This is useful for example where there is a need to check robot reach capability, detect collisions, and estimate cycle times, and finally generate robot programs which are verified off-line. It will be possible to predict problems which can only normally be experienced after purchasing and installing the complete production system. This type of simulation will be referred to as "simulation" or "robot simulation" throughout this thesis.

Although a clear distinction between strategic planning and detailed graphics simulation has been made, it is useful and often desirable for a hybrid simulation system to function in both areas. The performance evaluation of the total system obtained from a strategic planning simulation can be broken down and utilised in detailed graphics simulation of a particular work station. AutoSimulation and Heron are systems which use this approach as described in the previous chapter.

## 4.1 Objectives Of Simulation

The broad objectives of simulation applied to manufacturing systems, are to:

(i) train personnel in basic principles governing the design and use of manufacturing systems;

(ii) illustrate the elements of manufacturing systems so as to provide

experience in seeking alternatives and thereby improve decision making;

(iii) demonstrate feasibility of a design;

(iv) and in the case of robotic systems to: provide a programming facility whereby the robot task can be programmed without disrupting production;

(v) reduce the on-line programming hazard.

In recent years improvements have been made in robotics both in terms of hardware and software. This has made robots valuable work tools in the industrial environment and many new robotic installations continue to be developed. Unfortunately, the potential benefits which may be gained by using these advanced mechanisms are not automatically achieved simply by the purchase of a robot. A proper evaluation plan [Dooner, 1984] must be followed to correctly assess the many details and options concerned with the installation of robotic equipment. This robotic evaluation plan must seek to balance the many complicated system interfaces and wisely select from alternatives to create an efficient and cost effective system.

Simulation has proven effective for evaluating manufacturing systems to determine the relative merits, efficiency, and cost effectiveness of alternative system designs. A close approximation simulation model of a proposed robotic system can be developed and exercised to predict how the proposed system will perform and to illustrate the sensitivity of the design to various machine performance, and product arrival pattern contingencies [Welch, 1983]. This model may provide valuable insights

into system performance before the purchase and installation of the robotic system. Robot programs can also be developed off-line without disrupting the production system.

## 4.2 Requirements Of Simulation Models

There are a number of problems that must be overcome before robot simulators can be a very cost effective tool in real world situations.

In the robot simulators developed thus far kinematic models are used for the manipulator. However the large majority of these robot simulators do not account for the dynamic effects that are inherent in the actual robot [Sjolund and Donath, 1983] i.e. the robot simulators represent an idealisation;

(i) with no backlash in the joints, as feedback control dynamics is considered to be instantaneous.

(ii) with no gravitational or inertial effects, so the effects of any loads are not considered.

(iii) with no overshoot errors, as the links have infinite acceleration and deceleration capabilities.

A dynamic model would be required to truly simulate a manipulator in terms of kinematics and dynamics. This model would vary for each robot type. Most effort today involving the development of dynamic models is motivated by a wish to improve the control or structural performance of

the actual robot. Although the motivation is apparent, simulation of dynamic effects for animated graphics has not been implemented to any great extent. It is evident from the literature overview that only a limited number of dynamic simulation systems exist. Solutions for the dynamic models are relatively difficult to develop and often involve large computations. However even if the solutions are slow, it will still be useful to simulate a dynamic model of the task in non-real time, store the results for verification before downloading the program to the real system. This opens up new application areas where high precision is required.

Another modelling requirement is the creation of workplace entities. There is a need to consider how accurately the model should reflect the entire workplace environment (i.e. how detailed the model should be). This leads to further consideration of the conflicting aspects of very detailed simulation models which can provide better accuracy and visualization but are slow in animation.

## 4.3 Architecture of Simulators

Simulation involves the description of the robot's working environment and simulation of motion. The robot's working environment will consist of a variety of items of varying complexity.

This can be further considered as "Robot Modelling", "Object Modelling", "Geometric and Spatial Description", "Motion Specification" and "Animation".

### 4.3.1 Robot Modelling

Robot behaviour modelling was initiated as an application of time-and-motion methodologies to predict cycle times. In the model, robot displacements are calculated based on the assumption that all links are rigid, and the graphics are updated as the robot moves about the workplace.

#### (a) Robot Joint Definitions

Robot joints can be prismatic or revolute and will be constrained between minimum and maximum extensions. Figure 4-1 shows a revolute joint with three degrees of rotational freedom. For control simplicity, the orientation represented by ($\phi$, $\theta$, $\psi$) is governed by the following maximum constraints:

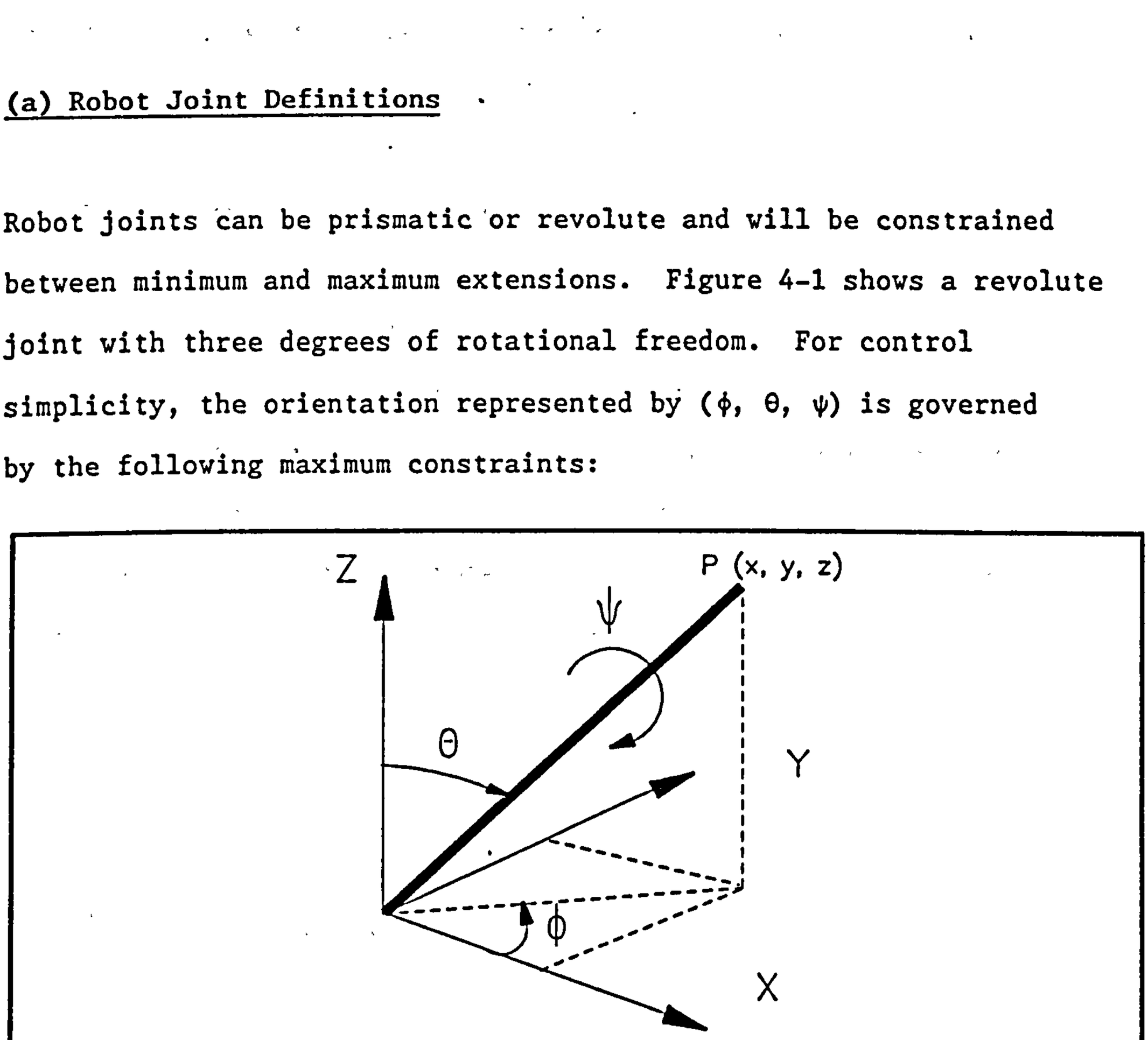



Figure 4–1 A single revolute robot joint with three degrees of freedom

(i) $-\pi \leq \phi \leq \pi$
(ii) $0 \leq \theta \leq \pi$
(iii) $-\pi \leq \psi \leq \pi$

With this set of constraints, there will only be one possible configuration which allows the end point (x, y, z) to be reached. However, relaxation of the joint constraints may result in degeneracies (i.e. more than one set of joint variables that result in the same position and orientation). In some instances this degeneracy is used to provide additional flexibility in positioning (e.g. as in the Adept robot). For this reason, robot simulation and robot control systems require robot configuration rules to avoid ambiguity.

## (b) Homogeneous Transformations for a Single Joint

Before the simulation of robot kinematics and dynamics are discussed, it is necessary to consider some basic mathematics of robot modelling. Homogeneous transformations are used to represent the translations and rotations of prismatic and revolute joints. The homogeneous transformation matrix for rotation about the X axis through $\theta$ degrees (denoted by ROT[X,$\theta$]) is given by

$$\mathrm{ROT}[X,\theta] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, the homogeneous transformation matrices for rotation about the Y and Z axes through $\theta$ degrees (denoted by ROT[Y,$\theta$] and ROT[Z,$\theta$] respectively) are given by

$$ROT[Y,\theta] = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$ROT[Z,\theta] = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The displacement of a coordinate frame along the X, Y, and Z axes is known as a translation transformation, and is given by

$$TRANS[X,Y,Z] = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These transformations are used in the formulation of joint models. Thus for example a roll-pitch-yaw angle set is defined as sequential rotation about the Z, Y, and X axes [Ranky and Ho, 1985]. This method is ambiguous and rarely used in robot simulation or robot control as the interdependence of these rotations results in multiple angle values when performing the inverse transformation.

An Eulerian (independent) angle set is described in terms of a rotation about Z axis followed by a rotation about the transformed Y

axis and a final rotation about the twice rotated Z axis. Eulerian angle sets are normally used for simulation or actual robot control because the inverse transformation may be performed without ambiguity. Denoting the Z-Y-Z rotations as $\phi$, $\theta$, $\psi$ respectively, and writing Cos$\phi$, Sin$\phi$ etc as C$\phi$, S$\phi$ etc, the total transformation matrix for a three degree of freedom revolute joint is given by:-

$$ROT[Z,\phi]\ ROT[Y,\theta]\ ROT[Z,\psi]$$

or,

$$\begin{bmatrix} C\phi & -S\phi & 0 & 0 \\ S\phi & C\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\psi & -S\psi & 0 & 0 \\ S\psi & C\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The general form of the transformation matrix is given as

$$\begin{bmatrix} C\phi C\theta C\psi - S\phi S\psi & -C\phi C\theta S\psi - S\phi C\psi & C\phi S\theta & 0 \\ S\phi C\theta C\psi + C\phi S\psi & -S\phi C\theta S\psi + C\phi C\psi & S\phi S\theta & 0 \\ -S\theta C\psi & S\theta S\psi & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{——————} \quad (1)$$

## (c) Homogeneous Transformations for a Sequence of Joints

The above approach can be extended to represent two or more joints which are sequentially linked. As the first example, consider a robot

arm which has a prismatic joint followed by one that is revolute about Z (see figure 4-2). By multiplying these matrices

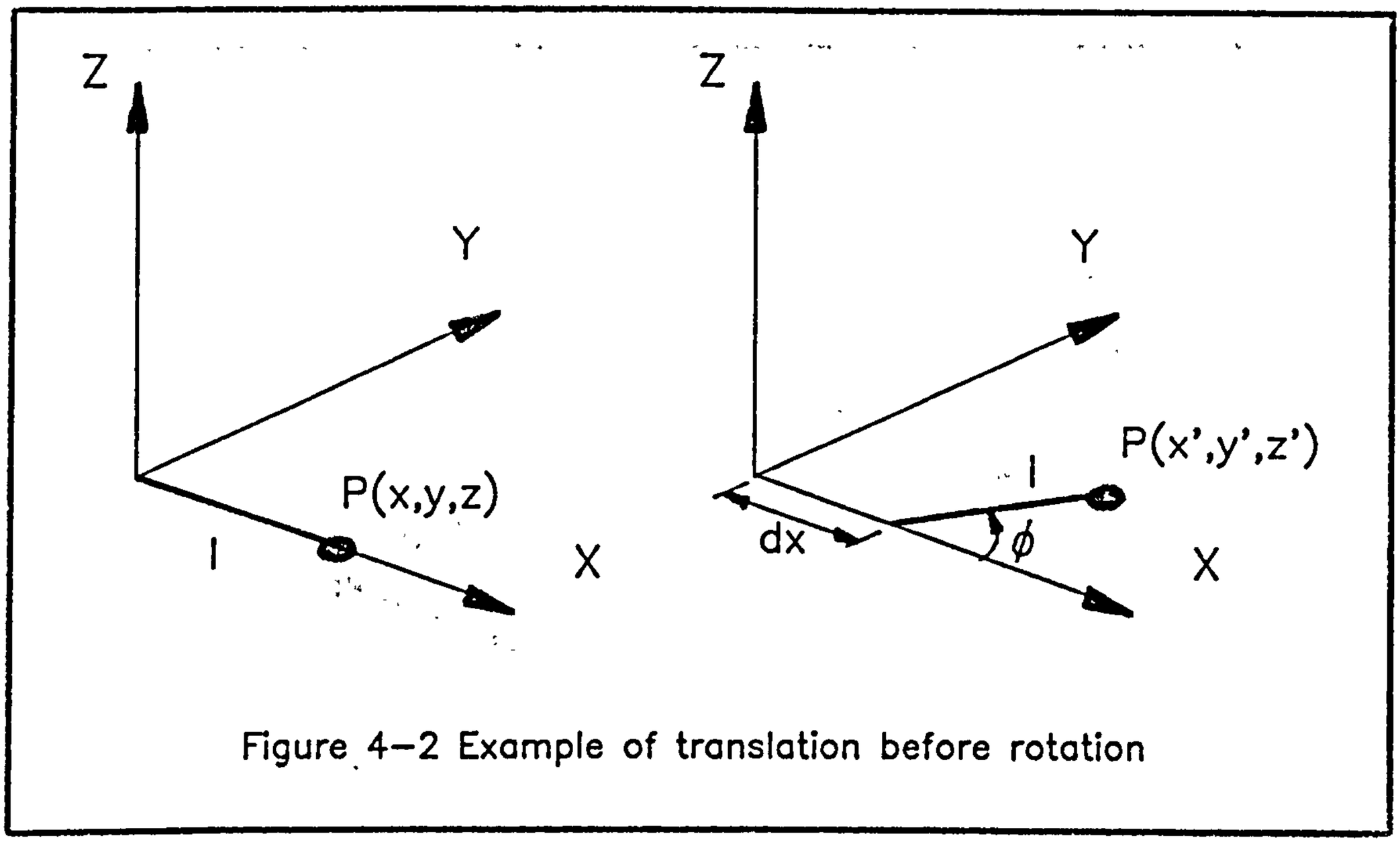



Figure 4–2 Example of translation before rotation

translation matrix rotation matrix link vector

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\phi & -S\phi & 0 & 0 \\ S\phi & C\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The Cartesian transformation in the general form are given by:-

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} XC\phi-YS\phi+dx \\ XS\phi+YC\phi+dy \\ Z+dz \\ 1 \end{bmatrix}$$

Secondly, consider a robot arm which undergoes a rotation $\phi$ degrees about the Z axis before translation along the X, Y, and Z axes (see figure 4-3). Similarly, by multiplying these matrices

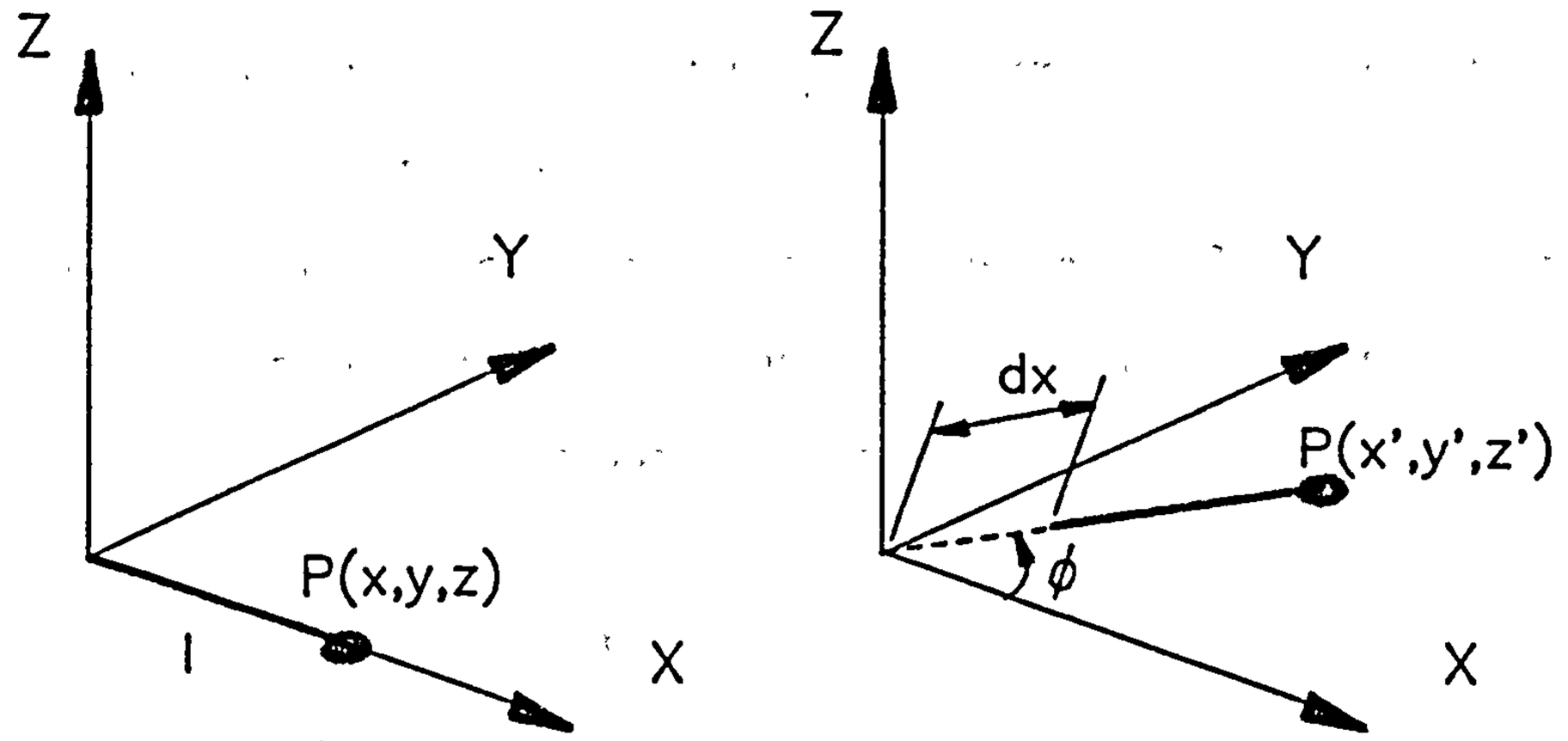



Figure 4–3 Example of rotation before translation

rotation matrix translation matrix link vector

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} C\phi & -S\phi & 0 & 0 \\ S\phi & C\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The Cartesian transformation in the general form are given by:-

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} (X+dx)C\phi-(Y+dy)S\phi \\ (X+dx)S\phi+(Y+dy)C\phi \\ Z \\ 1 \end{bmatrix}$$

## (d) Inverting Homogeneous Transformations

In robot simulation and control, the system must be capable of performing forward transformation (to convert joint angles to Cartesian coordinates) and inverse transformation (to convert Cartesian coordinates into individual joint angles). The inverting of a homogeneous transformation is one of the basic elements for inverse transformation and for calculating the relationships between objects. For a given homogeneous transformation T

$$T = \begin{bmatrix} Nx & Ox & Ax & Px \\ Ny & Oy & Ay & Py \\ Nz & Oz & Az & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

then the inverse of the transformation T is

$$T^{-1} = \begin{bmatrix} Nx & Ny & Nz & -P.N \\ Ox & Oy & Oz & -P.O \\ Ax & Ay & Az & -P.A \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first 3x3 elements (rotational elements) of the matrix can simply be inverted by transposition, while the elements of the last column of the matrix are the dot products of the vectors concerned. The element P.N is the dot product of vectors P and N, thus

$$P.N = Px*Nx + Py*Ny + Pz*Nz$$

Similarly,

$$P.O = Px*Ox + Py*Oy + Pz*Oz \quad \text{and}$$

$$P.A = Px*Ax + Py*Ay + Pz*Az$$

## (e) Forward Kinematics of a Robot with two Revolute Joints

Consider a manipulator which consists of only two degrees of freedom, with motion restricted to the X and Y plane (see figure 4-4). The forward kinematic transformation is performed to determine the end-effector or tool centre point (TCP) in Cartesian space. For simplification, Cos$\phi$1 and Sin$\phi$1 are written as C1 and S1 etc.

$$\begin{bmatrix} X" \\ Y" \\ Z" \\ 1 \end{bmatrix} = \begin{bmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & L1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C2 & -S2 & 0 & 0 \\ S2 & C2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The robot's end-effector or TCP in Cartesian coordinates are

$$X" = L1*C1+L2*C1*C2-L2*S1*S2 \quad \text{——————} \quad (2)$$

$$Y" = L1*S1+L2*S1*C2+L2*C1*S2 \quad \text{——————} \quad (3)$$

$$Z" = Z$$

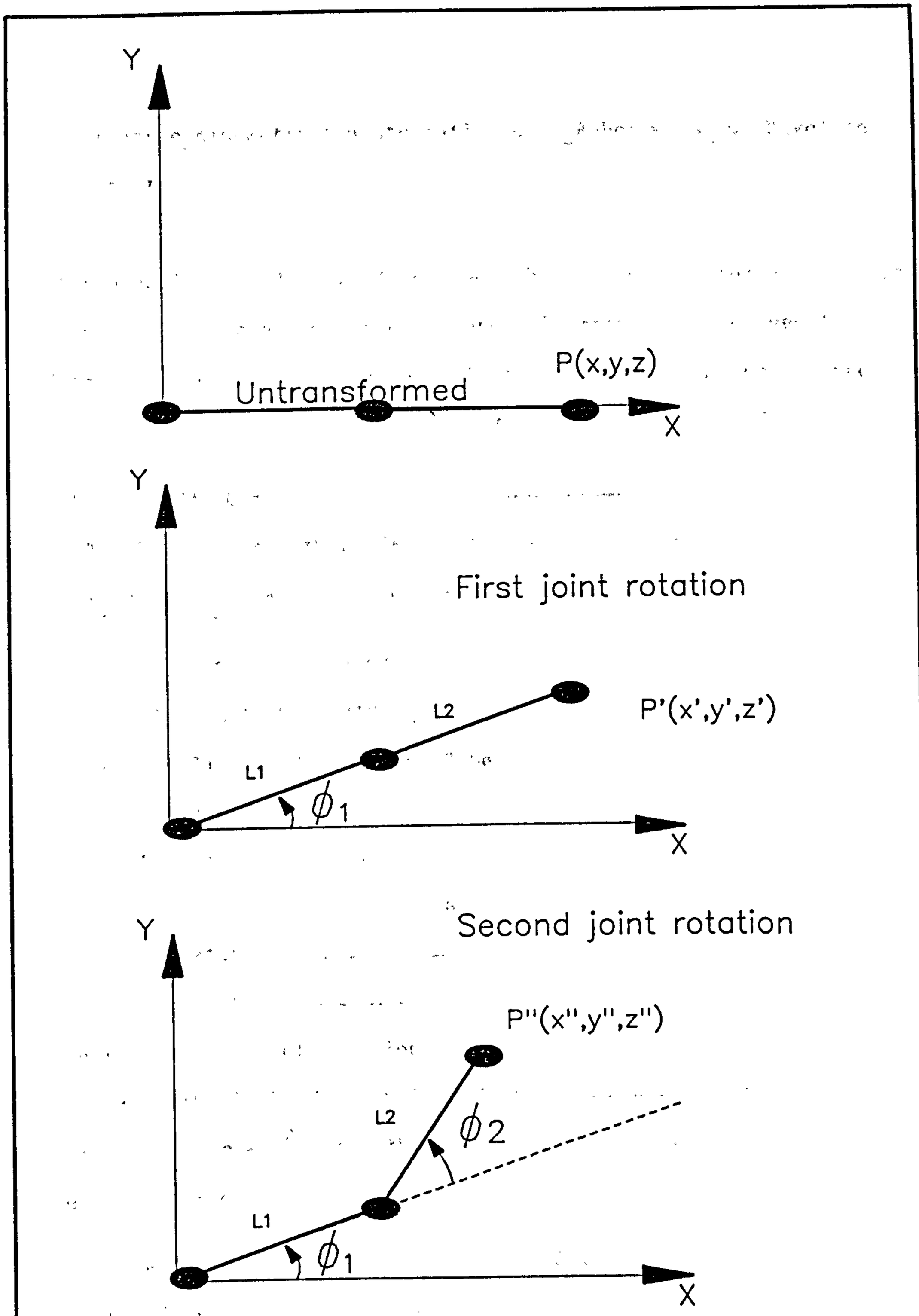



Figure 4–4 A robot with two revolute joints

## (f) Inverse Kinematic Transformation of a Robot with two Revolute Joints

If the position (X", Y", Z") of the robot's end-effector and the arm lengths L1, L2 are known then inverse kinematics can be used to determine the arm joint angles. In the inverse situation, $\phi 1$ and $\phi 2$ can be calculated through solving equations (2) and (3).

$$X" = L1*C\phi 1+L2*C\phi 1*C\phi 2-L2*S\phi 1*S\phi 2 \quad \text{——————} \quad (2)$$

$$Y" = L1*S\phi 1+L2*S\phi 1*C\phi 2+L2*C\phi 1*S\phi 2 \quad \text{——————} \quad (3)$$

Half angle rules are used to simplify the complex angle multiplications.

$$S(\phi 1+\phi 2) = C\phi 1S\phi 2 + S\phi 1C\phi 2$$

$$C(\phi 1+\phi 2) = C\phi 1C\phi 2 - S\phi 1S\phi 2$$

Rearrange (2) and (3), X" and Y" becomes

$$X" = L1*C\phi 1+L2*C(\phi 1+\phi 2)$$

$$Y" = L1*S\phi 1+L2*S(\phi 1+\phi 2)$$

Substitute $\phi' = (\phi 1+\phi 2)$, we obtain

$$X" = L1*C\phi 1+L2*C\phi' \quad \text{——————} \quad (4)$$

$$Y" = L1*S\phi 1+L2*S\phi' \quad \text{——————} \quad (5)$$

Square both sides of equations (4) and (5)

$$X"^2 = L1^2*C^2\phi 1+L2^2*C^2\phi'+2*L1*L2*C\phi 1*C\phi' \quad \text{——————} \quad (6)$$

$$Y"^2 = L1^2*S^2\phi 1+L2^2*S^2\phi'+2*L1*L2*S\phi 1*S\phi' \quad \text{——————} \quad (7)$$

Adding equations (6) and (7)

$$X"^2+Y"^2 = L1^2+L2^2+2*L1*L2*(C\phi'*C\phi 1+S\phi'*S\phi 1)$$

$$X"^2+Y"^2 = L1^2+L2^2+2*L1*L2*C(\phi'-\phi 1)$$

$$X"^2+Y"^2 = L1^2+L2^2+2*L1*L2*C\phi 2$$

which results in

$$\phi 2 = \text{Cos}^{-1}\left[\frac{X''^2+Y''^2-L1^2-L2^2}{2*L1*L2}\right]$$

Since $\phi 2$ is known, therefore $\text{Sin}\phi 2$ and $\text{Cos}\phi 2$ are also known. If we let constants a and b be defined by

$a = L1+L2*\text{Cos}\phi 2$, and $b = L2*\text{Sin}\phi 2$

Substitute a and b into (2) and (3), we have

$X'' = a*\text{Cos}\phi 1 - b*\text{Sin}\phi 1$ ———————— (8)

$Y'' = b*\text{Cos}\phi 1 + a*\text{Sin}\phi 1$ ———————— (9)

multiplying (8) and (9) by a and b respectively, we obtain

$aX'' = a^2*\text{Cos}\phi 1 - a*b*\text{Sin}\phi 1$ ———————— (10)

$bY'' = b^2*\text{Cos}\phi 1 + a*b*\text{Sin}\phi 1$ ———————— (11)

Adding (10) and (11) lead to

$aX''+ bY'' = (a^2+b^2)*\text{Cos}\phi 1$

$$\phi 1 = \text{Cos}^{-1}\left[\frac{aX''+ bY''}{(a^2+b^2)}\right]$$

This inverse transformation solution is specific to this example only, as there is no general inverse solution which is appropriate for robots with different configurations. If a robot has a more complex configuration, then the inverse kinematics is more difficult. This may involve the inverting of homogeneous transformations to perform the require manipulation. For example, the location of the robot's end-effector can be described as

$$^{R}T_{H} = T1*T2*T3*T4*T5*T6$$

where T1, T2 etc are the transformations for single degrees of freedom.

In solving for individual joint angles, matrix inversion is frequently a useful mathematical technique. For example once one joint is solved then:

$$T1^{-1} * {}^{R}T_{H} = T2*T3*T4*T5*T6$$

etc

These specific inverse solutions may be applied to a range of robots with a similar configuration, and several may be found in Snyder (1985), Craig (1985), and Ranky and Ho (1985).

## (g) Common Algorithms Used in Robot Simulators

Robot simulators can be classified as being kinematic or dynamic. In the context of robotics, kinematic analysis is a study of robot postures adopted at different times. The properties of the robot motion depend on the geometry of its configuration and hence rigid links are simulated to maintain fixed configurations between joints. Most of today's industrial robots are kinematically simple, (although some multiple axis robots are configured with complex joint interconnection mechanisms to connect several links, which may be simplified to serial joint mechanisms). The present generation of kinematic robot simulators are capable of modelling robots with serial links but complex joint interconnections would require special treatment to deal with this ad hoc situation. Thus usually a robot is represented by a series of links and each link is jointed with reference to a coordinate frame. The robot link representation is modelled in the same way as object modelling using the same object modeller.

The Denavit-Hartenberg algorithm [Denavit and Hartenberg, 1955; Paul, 1981; Goldenberg and Lawrence, 1985; Cook and Vu-Dinh, 1985; McCarthy, 1986] is the most commonly used [Gupta, 1986] in kinematic robot simulators. The Denavit-Hartenberg analysis uses closed form equations (the most efficient means of expressing their kinematics [Paul and Zhang, 1986]) where the transformation of the robot end-effector to the robot base is directly equivalent to the concatenation of the sequential transformations between each robot link from the robot end-effector to its base. The general expression of the forward kinematics of an n-jointed robot is:

$$ {}^{\text{robot base}}_{\text{end-effector}}T = {}^{\text{base}}_{\text{joint 1}}T * {}^{\text{joint 1}}_{\text{joint 2}}T * \ldots {}^{\text{joint n-1}}_{\text{joint n}}T $$

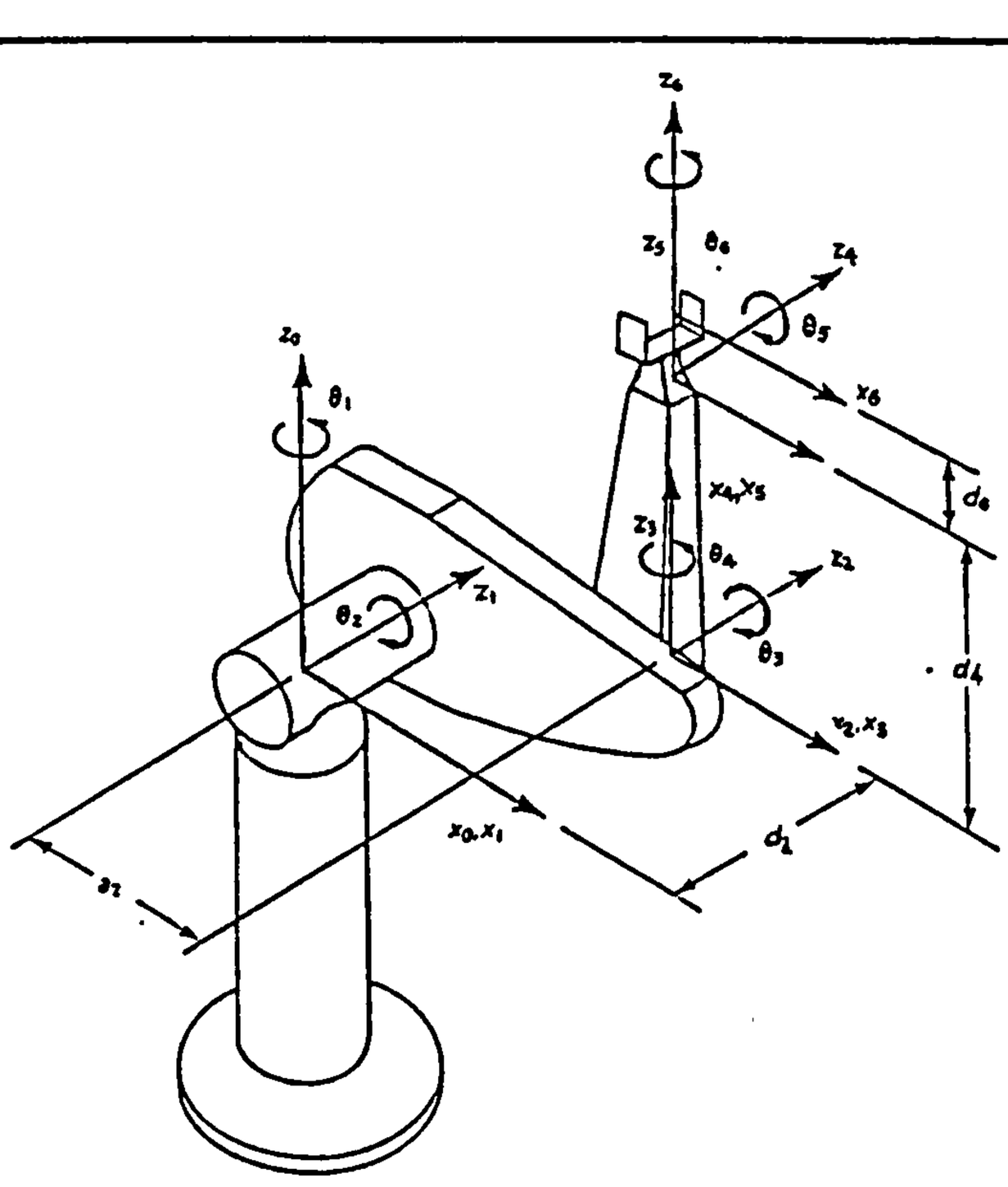



Figure 4–5 Assignment of coordinate frame to each robot joint for the formulation of homogeneous transformation matrices

Figure 4-5 shows a six degrees of freedom Puma robot with a coordinate frame assigned to each joint. The formulation of an homogeneous transformation matrix for each joint is discussed later.

The Denavit-Hartenberg method is based on 4x4 homogeneous transformation matrices with special conventions for applying coordinate frames to the structure of a robot. These conventions include:

(i) The analysis starts with joint 1 (the moveable joint attached to a fixed reference) and finishes at link n.

(ii) A right handed coordinate frame must be applied to a joint with its Z axis always arranged to be in line with the axis of joint rotation/translation.

(iii) The X-axes of all of the joint frames are aligned in the same direction as the robot base frame.

(iv) All revolute joints must rotate about their respective Z axes.

(v) All prismatic joints must translate along their respective Z axes.

The following rotation and translation parameters (see figure 4-6) are required to form the 4x4 homogeneous transformation matrix at joint i.

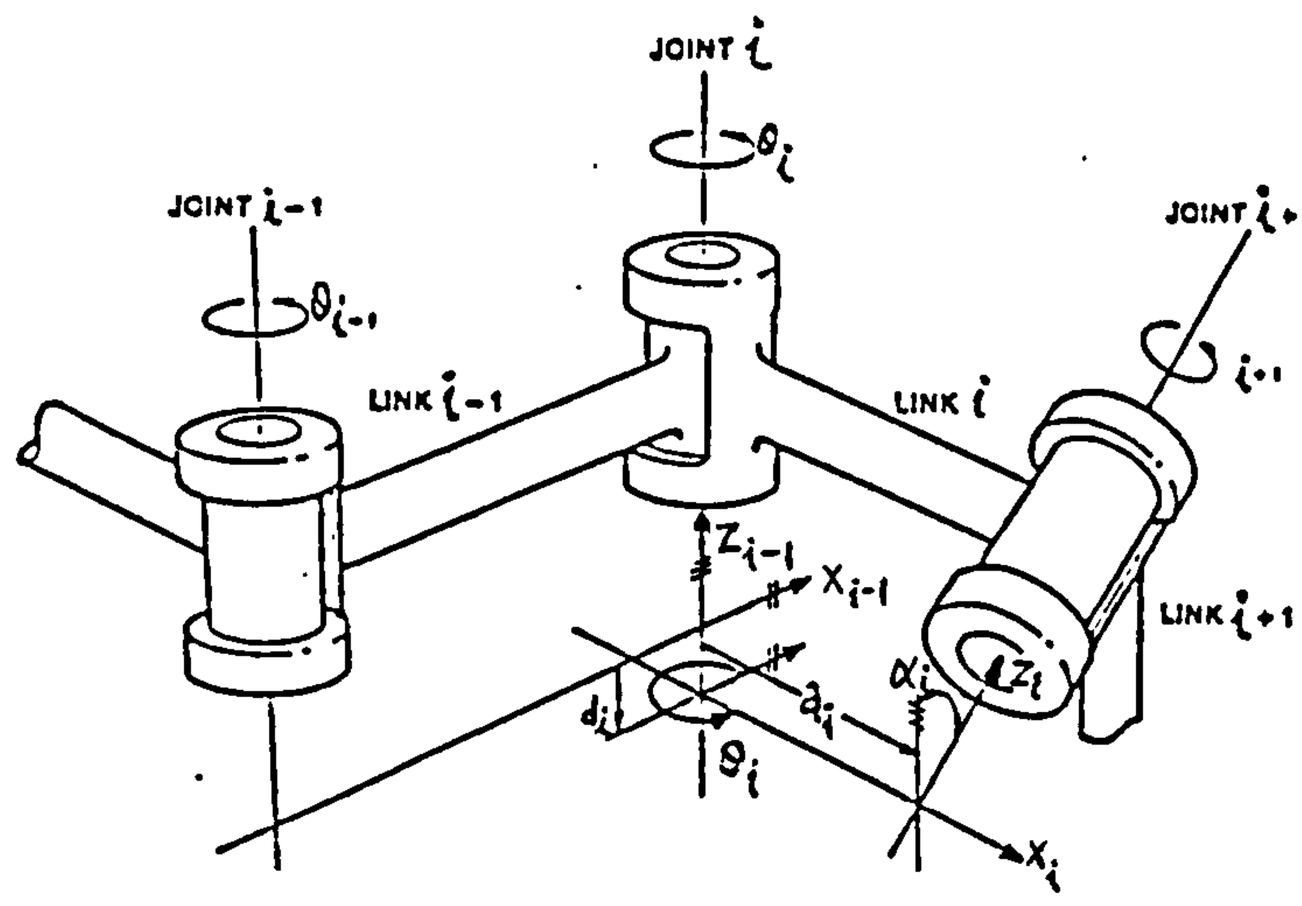



Figure 4–6 Rotation and translation parameters required for the formulation of homogeneous transformation matrix at joint i

$a_i$ distance between Z(i+1)and Z(i) axis translated along the X(i) axis

$\alpha_i$ rotation about the X axis of joint i+1 with respect to joint i

$d_i$ distance between X(i) and X(i-1) axis translated along the Z(i) axis

$\theta_i$ rotation about the Z axis of joint i

The relationship between the joint coordinate frame i-1 and i can be expressed by a homogeneous transformation (Denavit-Hartenberg) matrix Ti, and

$$Ti = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The kinematic model requires kinematic data for the robot involved. A typical example of input data required for the GRASP robot simulation package includes:

(i) Spatial relationships between joints and the type of degree of freedom (either revolute or prismatic);

(ii) Spatial relationship between the tool attachment point (i.e. mounting flange) and the last joint of the robot;

(iii) configuration rules that the robot has to obey (for reaching positions which can be attained with different configurations);

(iv) the initial position of the robot expressed as a set of joint extensions;

(v) minimum and maximum joint constraints;

(vi) robot park position;

(vii) joint velocities;

(viii) joint accelerations.

An example of such inputs for an industrial robot with six degrees of freedom is to be found in the STAR syntax as shown in figure 4-7. The syntax of STAR is chosen because this is a dynamic robot simulation system which can be used to illustrate that the kinematic input data is the skeleton of dynamic robot simulation.

Kinematic robot simulators face problems when there is a need to model high precision tasks in which the dynamic characteristics of the robot affect its performance. This provides the motive for taking dynamic effects into consideration, and as a result a few dynamic robot simulators have been developed.

```
1: !*****************************************************************************
2:       STAR MODEL OF A SIX DEGREE OF FREEDOM INDUSTRIAL ROBOT.
3: !*****************************************************************************
4: !
5: !CONNECTIVITY DEFINITION STATEMENTS.
6: REVOLUTE(BASE,WAIST)= PIVOT
7: REVOLUTE(WAIST,LOWERARM)= SHOULDER
8: REVOLUTE(LOWERARM,UPPERARM)= ELBOW
9: REVOLUTE(UPPERARM,WAIST)= BEND
10: REVOLUTE(WAIST,HAND)= TWIST
11: REVOLUTE(HAND,TOOLHOLDER)= SPIN
12: ! JOINT LOCATION DEFINITION STATEMENTS.
13: COORDINATES(PIVOT)= 0 10 0,0 50 0,50 10 0,50 10 0
14: COORDINATES(SHOULDER)= 0 70 0,0 70 50,50 70 0,50 70 0
15: COORDINATES(ELBOW)= 0 139 0,0 139 50,50 139 0,50 139 0
16: COORDINATES(BEND)=  67 139 0,67 139 50,150 139 0,150 139 0
17: COORDINATES(TWIST)= 75 139 0,100 139 0,75 139 -100,75 139 -100
18: COORDINATES(SPIN)= 85 130 0,85 0 0,200 130 0,200 130 0
19: ! TOOL CENTER POINT (TCP) LOCATION STATEMENTS.
20: COORDINATES(TCP)= 85 130 0,85 0 0,200 130 0
21: ! GRAVITATIONAL CONSTANT AND FIELD VECTOR STATEMENTS.
22: CONSTANT GRAVITY= 1.0
23: DATA GRAVITY= 0 -980 0
24: ! LINK INERTIAS DEFINITION STRAEMENTS.
25: WEIGHT(WAIST,PIVOT)= 25, 35 0 0
26: WEIGHT(LOWERARM,SHOULDER)= 15,0 35 0
27: WEIGHT(UPPERARM,ELBOW)= 15,35 0 0
28: WEIGHT(WRIST,BEND)= 5,5 0 0
29: WEIGHT(HAND,TWIST)= 5,5 0 0
30: WEIGHT(TOOLHOLDER,SPIN)= 2,0 0 1
31: ! JOINT ACTUATOR TORQUE LIMITS.
32: LIMIT FORCE(PIVOT)= -.3E7, .3E7
33: LIMIT FORCE(SHOULDER)= -.35E7, .35E7
34: LIMIT FORCE(ELBOW)= -.25E7, .25E7
35: LIMIT FORCE(BEND)= -.25E6, .25E6
36: LIMIT FORCE(TWIST)= -.4E5, .4E5
37: LIMIT FORCE(SPIN)= -.2E5, .2E5
38: ! STATEMENTS FOR RETRIVAL OF ROBOT
39: ! LINKS' GRAPHICS DATABASE FROM DATA FILES.
40: SHAPE BASE @ PIVOT= G:BASE
41: SHAPE WAIST @ SHOULDER= G:WAIST
42: SHAPE LOWERARM @ ELBOW= G:LOWER
43: SHAPE UPPERARM @ ELBOW= G:UPPER
44: SHAPE WAIST @ BEND= G:WAIST
45: SHAPE HAND @ TWIST= G.HAND
46: !*****************************************************************************
47: !      DEFINITION OF OBJECTS IN THE ROBOT ENVIRONMENT.
48: !*****************************************************************************
49: !
50: ! DEFINE AN OBJECT CALLED TOOL FOR GRIPPING.
51: OBJECT= TOOL
52: !PUT IT AT THE END OF THE ROBOT ARM.
53: COORDINATES(TOOL)= 85 130 0,85 0 0,200 130 0
54: WEIGHT(TOOL)= 1,2 -1 0
55: ! AFFIX AN AUXILIARY COORDINATE NAMED GRIPPER TO THE
56: !TOOL AT THE CENTER OF THE TOOL'S GRIPPER FINGERS.
57: AFFIX GRIPPER = > TOOL
58: COORDINATES(GRIPPER)= 0 0 10,0 10 10,10 0 10
59: ! DEFINE AN OBJECT NAMED TABLE IN FRONT OF THE ROBOT ARM.
60: OBJECT= TABLE
61: COORDINATES(TABLE)= 100 0 0,100 0 10,200 0 0
62: ! DEFINE AN OBJECT NAMED STOCK ON THE TABLE.
63: OBJECT= STOCK
64: COORDINATES(STOCK)= 90 60 40,90 60 100,100 60 40
65: WEIGHT(STOCK)= 4,2 25 0
66: ! AFFIX AUXILIARY COORDINATES TO STOCK.
67: AFFIX STOCKTOP = > STOCK
68: AFFIX ABOVESTOCK = > STOCK
69: COORDINATES(STOCKTOP)= 0 35 0,0 35 35,35 35 0
70: COORDINATES(ABOVESTOCK)= 0 45 0,0 45 45,45 45 0
71: ! DEFINE AN OBJECT NAMED FIXTURE ON THE TABLE.
72: OBJECT= FIXTURE
73: COORDINATES(FIXTURE)= 110 60 -40,110 60 0,200 60 -40
74: WEIGHT(FIXTURE)= 3,1 5 0
75: ! AFFIX AN AUXILIARY COORDINATE TO FIXTURE.
76: AFFIX HOLE = > FIXTURE
77: COORDINATES(HOLE)= 0 5 0,0 5 5,5 5 0
78: ! AUXILIARY COORDINATE SYSYTEM FIXED IN SPACE.
79: AFFIX HOME = > BASE
80: COORDINATES(HOME)= 85 120 0,85 120 100,200 120 0
81: ! STATEMENTS TO RETRIEVE THE GRAPHICS DATABASE FOR THE OBJECTS.
82: SHAPE TOOL= G:TOOL
83: SHAPE STOCK= G:STOCK
84: SHAPE FIXTURE= G.FIXTURE
85: SHAPE TABLE= G:TABLE
86: RETURN
```

Figure 4–7 Example robot simulator inputs for a robot with six degrees of freedom (adapted from Hornick and Ravani, 1986)

These simulators represent a step change which indicates the direction of future robot simulators, and open up the opportunity for more precise operations where the accuracy requirement is critical. A kinematic model also forms an essential part of a dynamic robot simulator. The frequently referenced algorithms in the context of dynamic simulation include the Lagrangian method [Murray and Neuman, 1984; Wang and Kohli, 1985] and Newton-Euler method [Khosla and Neuman, 1985]. The Lagrangian method is based on energy law in which the forces or torques are described in terms of the differences in kinetic and potential energy, whilst the Newton-Euler method is deduced from Newton's laws of motion. The Lagrangian method presents a systematic way of forming dynamic equations in closed form whereas the Newton-Euler method is based on a recursive algorithm. The computation required for dynamic robot simulation is significant when compared with kinematic robot simulation due to the complexity involved in the equations. The computation involved in these two methods has been quoted [Ramaswamy et al, 1985] in terms of the number of links involved. With the Newton-Euler method, the amount of computation increases linearly with the number of links, whereas the Lagrangian method increases with the fourth power of the number of links.

The Lagrangian algorithm for a robot with n joints can be described as

$$E = K - P$$

where K is the kinetic energy
P is the potential energy
E is the difference between the kinetic and potential energy

The Newton-Euler equation of motion for a robot with n joints can be described as

$$T = I(q)\, q'' + C(q, q') + G(q)$$

where

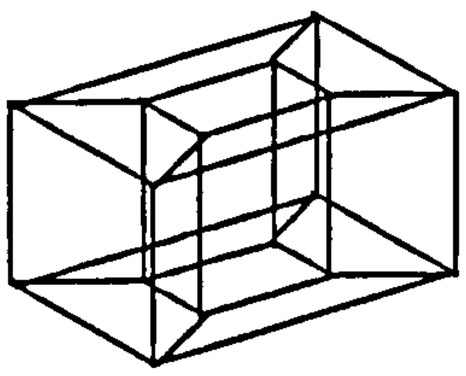
T is the torque/ force applied
I is a n x n inertia matrix
C is a n x 1 vector defining the coriolis and centrifugal terms
G is a n x 1 vector defining the gravitational term
q is coordinates
q′ is velocities
q" is acceleration

In addition to input data required for the kinematic model, the dynamic robot model takes the weight of a robot arm into consideration, and formulates the relationships between forces acting at each joint and its associated displacements, velocities and accelerations, to predict the dynamic properties of a robot. As is to be expected, dynamic robot simulators can only represent the dynamic behaviour of a single robot since no two robots are identical in every aspect even though they were of the same model supplied from the same manufacturer. The robot model is constructed based on dynamic information obtained from experiments. To complicate the problem further, individual robots would suffer different dynamic characteristics over time. Thus a dynamic robot simulator is a representation of a close approximation of real robots' dynamic behaviour provided that parameters of the model can be established. Although different algorithms are used in different robot simulators, their eventual aim is the same (i.e. modelling and off-line robot programming). Both kinematic and dynamic simulations are integral parts of a robot simulator.

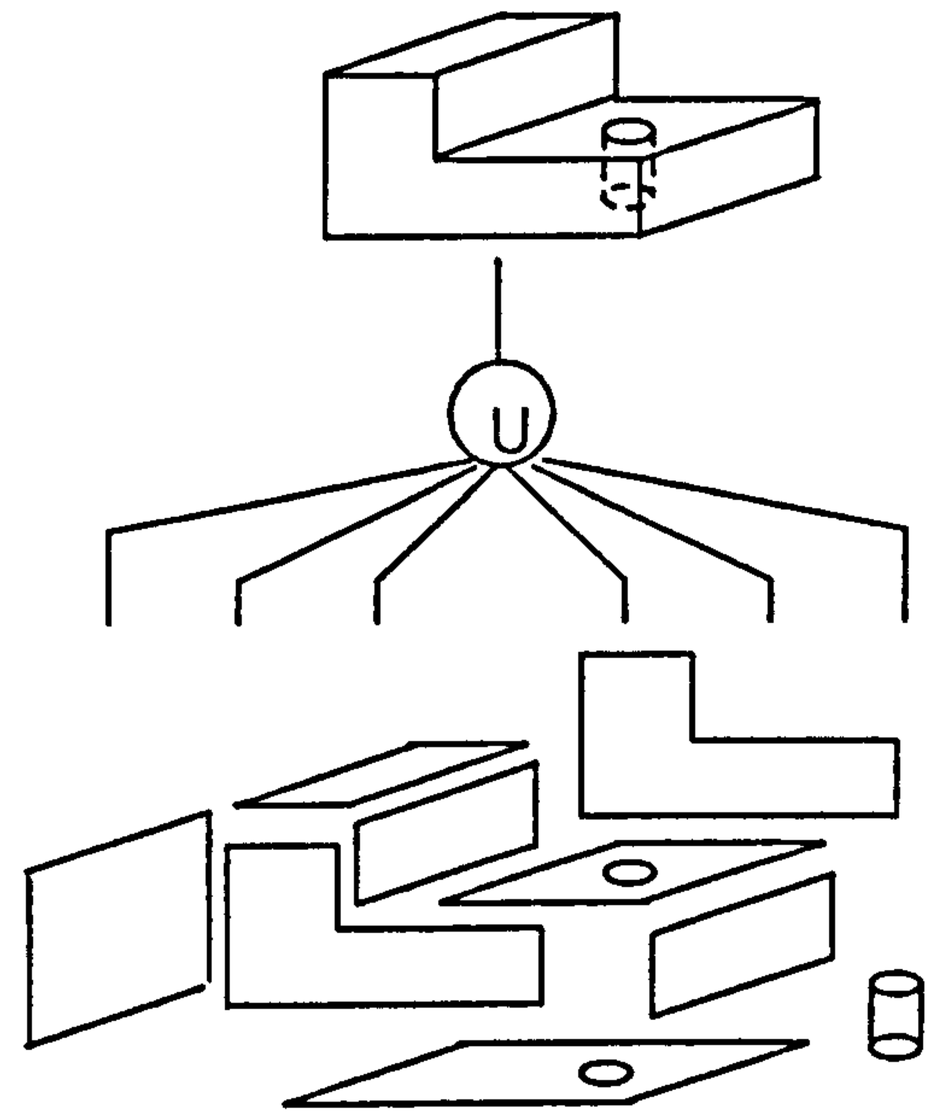
### 4.3.2 Object Modelling

Let us consider some of the necessary features of the geometric part of the modeller of the robot simulator. Geometric representation can be broadly divided into wireframe and solid representation. Although wireframe representation can be ambiguous, the data structure is relatively simple (containing edges and vertices). The visual ambiguity of wireframe representation can be reduced [Woodwark, 1986] by using perspective projection and by depth-cueing techniques in which the intensity of the lines in the wireframe is reduced with distance away from the viewer. A wireframe picture can be obtained quickly by applying a projection to each vertex. Picture segments are normally held in a display list, and special hardware performs transformations on every element in the list so quickly that fast manipulation of objects in graphics space can be achieved (i.e. wireframe can be made to move or spin on the screen). The McAuto simulator is an example of this when it uses an Evans and Sutherland refresh graphics terminal with hardware transformation. The modelling transformations just described can also be handled dynamically in the same way. The wireframe technique is useful for motion animation and constraint checking, but collision detection among objects is not possible.

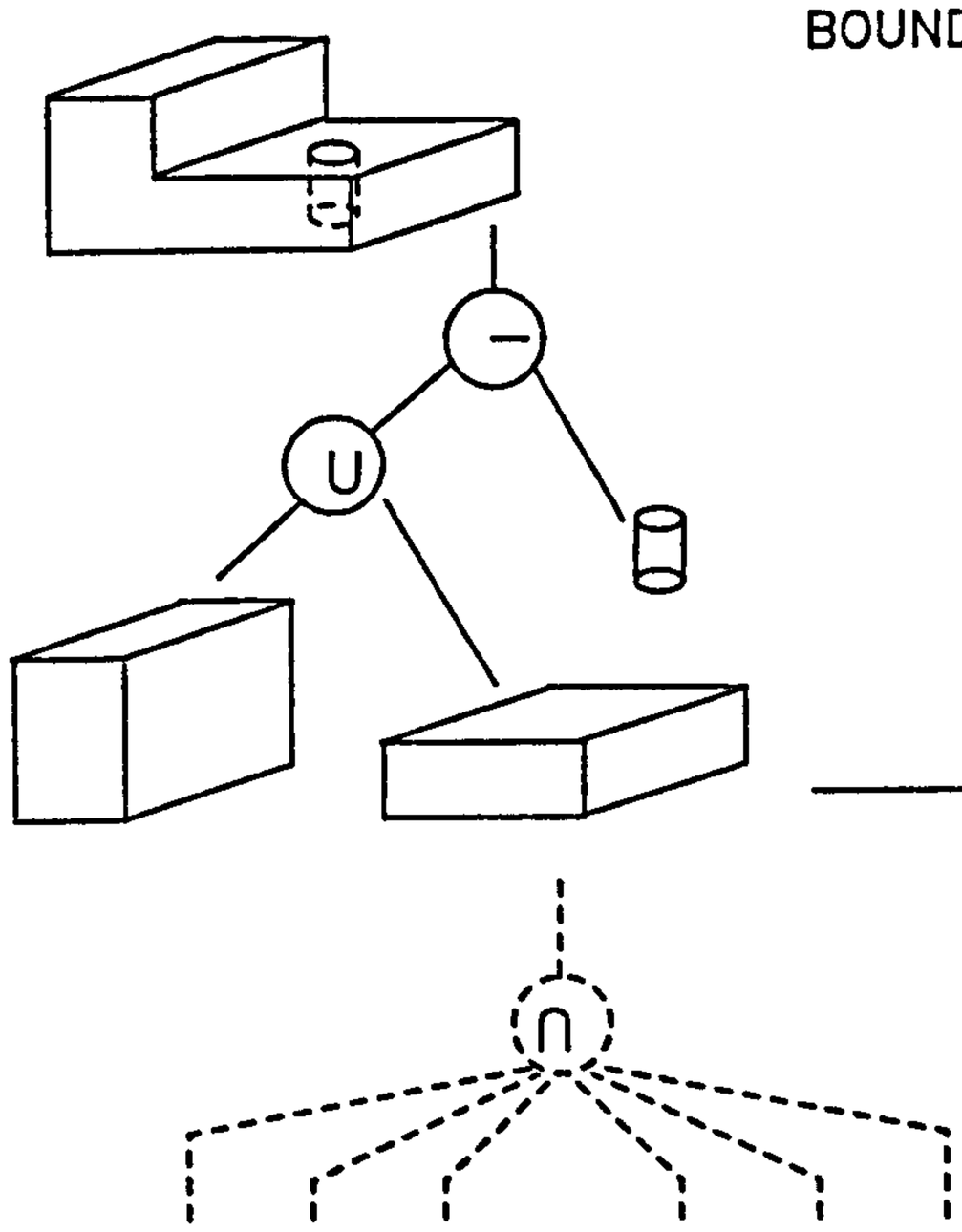
Commonly either "Constructive Solid Geometry" [Requicha, 1980] or "Boundary Representation" [Braid, 1975] solid modelling techniques are used to allow a three dimensional model of the robot and workplace to be constructed from simple primitive shapes such as cuboids, regular prisms and cylinders or generally by closed polyhedra. Figure 4-8



WIREFRAME PRESENTATION




BOUNDARY REPRESENTATION




bounded primitives

half spaces

six planar half spaces

CONSTRUCTIVE SOLID GEOMETRY

Figure 4−8 Differences between Wireframe, Boundary Representation and Constructive Solid Geometry (adapted from Requicha, 1980)

shows the differences between wireframe, Boundary Representation and CSG.

In Boundary Representation (Brep), a solid is represented by a number of segments or faces, bounded by edges and vertices. This face information (possibly including simply curved surfaces) gives a method of determining solidity. The Constructive Solid Geometry (CSG) (also known as Set-theoretic) defines solids based on primitives or sets of half spaces linked together by boolean operators. It contains no explicit data about edges or vertices.

Since the data structure of solid modelling is relatively complex, the retrieval of a model would take a long time. Although many simulation systems use solid modelling representations, wireframe models can be produced using converting algorithms for faster screen display. Since Set-theoretic solid models contain no explicit data about edges and vertices, producing wireframe models from Set-theoretic models is more difficult than from Boundary models.

Typical robot simulators will facilitate model creation, using "ease of use" data input methods, and allow the created models to be stored in, and recalled from, a library. When created the robot and workplace model may be viewed and manipulated in standard ways e.g. displayed graphically in plan, front and side elevations, and in perspective and plane parallel projections from any viewpoint. The spatial relationship between entities in the model (the robot and its equipment for example) can be controlled using the normal CAD input and display practices.

### 4.3.3 Geometric and Spatial Description

All model based programming languages (including robot simulators) use a structured geometric database (figure 4-9). The geometric database has complete knowledge on the geometry of the environment to the needs of the modelling objective. The simulation model is normally constructed based on coordinate frame concepts [Van Aken and Van Brussel, 1987] to represent the physical environment layout (figure 4-10), and each frame is described with reference to the owner frame in the hierarchy above. An important element of such coordinate frames is known as the "Centre Frame". If an object is gripped in a robot gripper, it is no longer evident to reason on the gripper motion in order to obtain the desired robot motion for the object. Any frame which belongs to an object gripped by the robot can be declared as the new centre frame. Any subsequent motion specification then refers to that frame (figure 4-11).

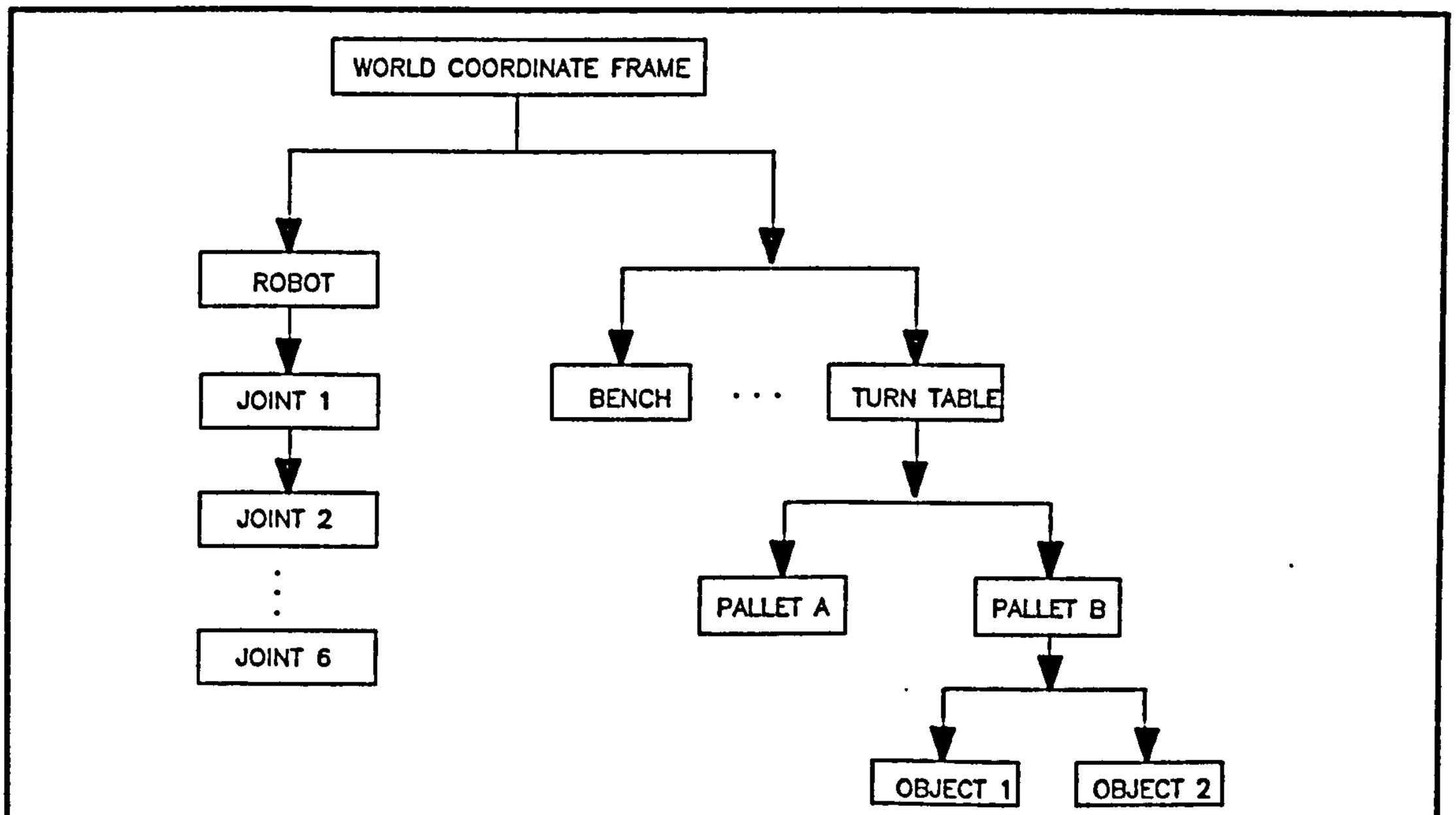



Figure 4-9 Structure of a geometric database

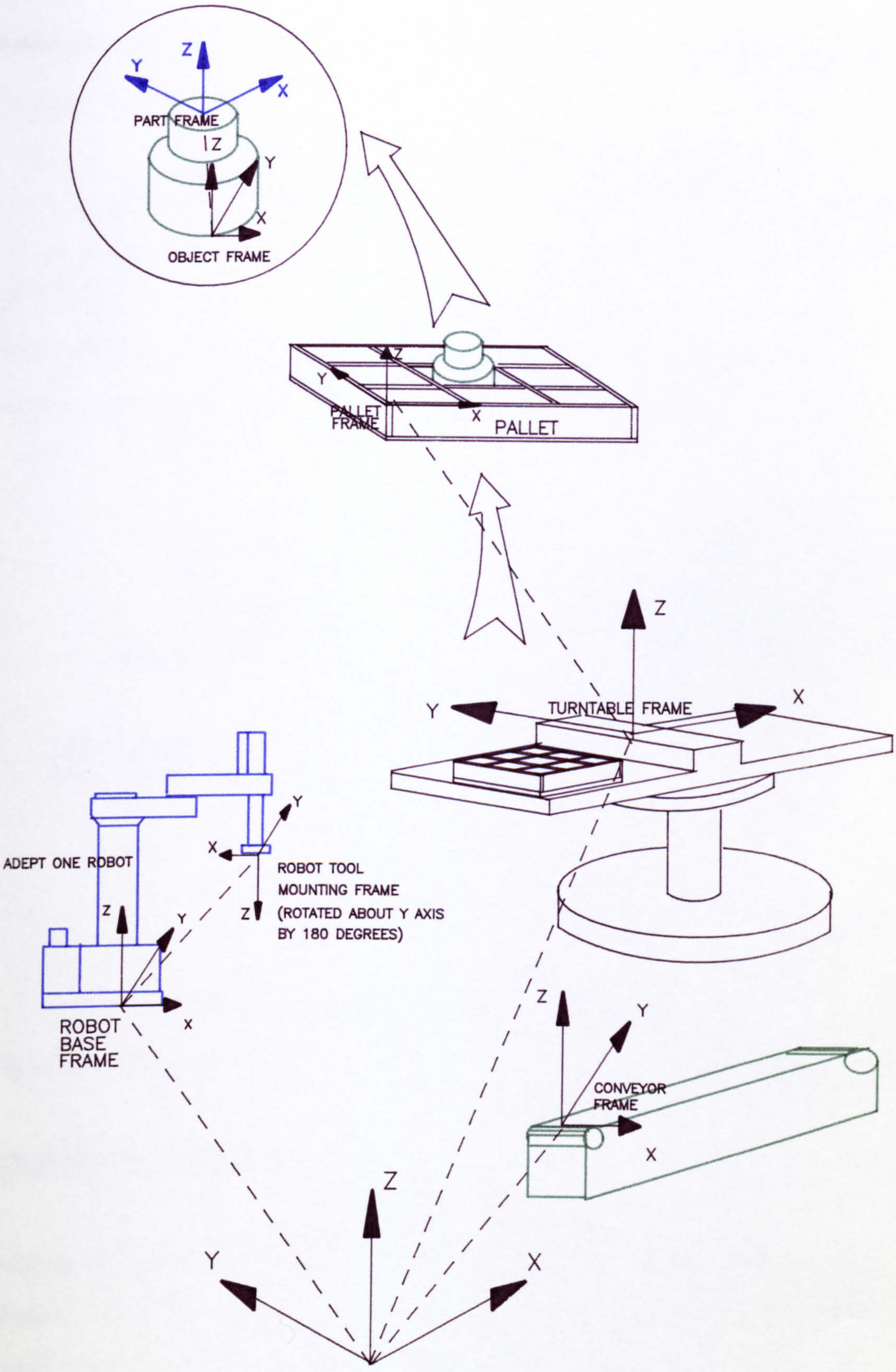



Figure 4–10 Coordinate frame concept

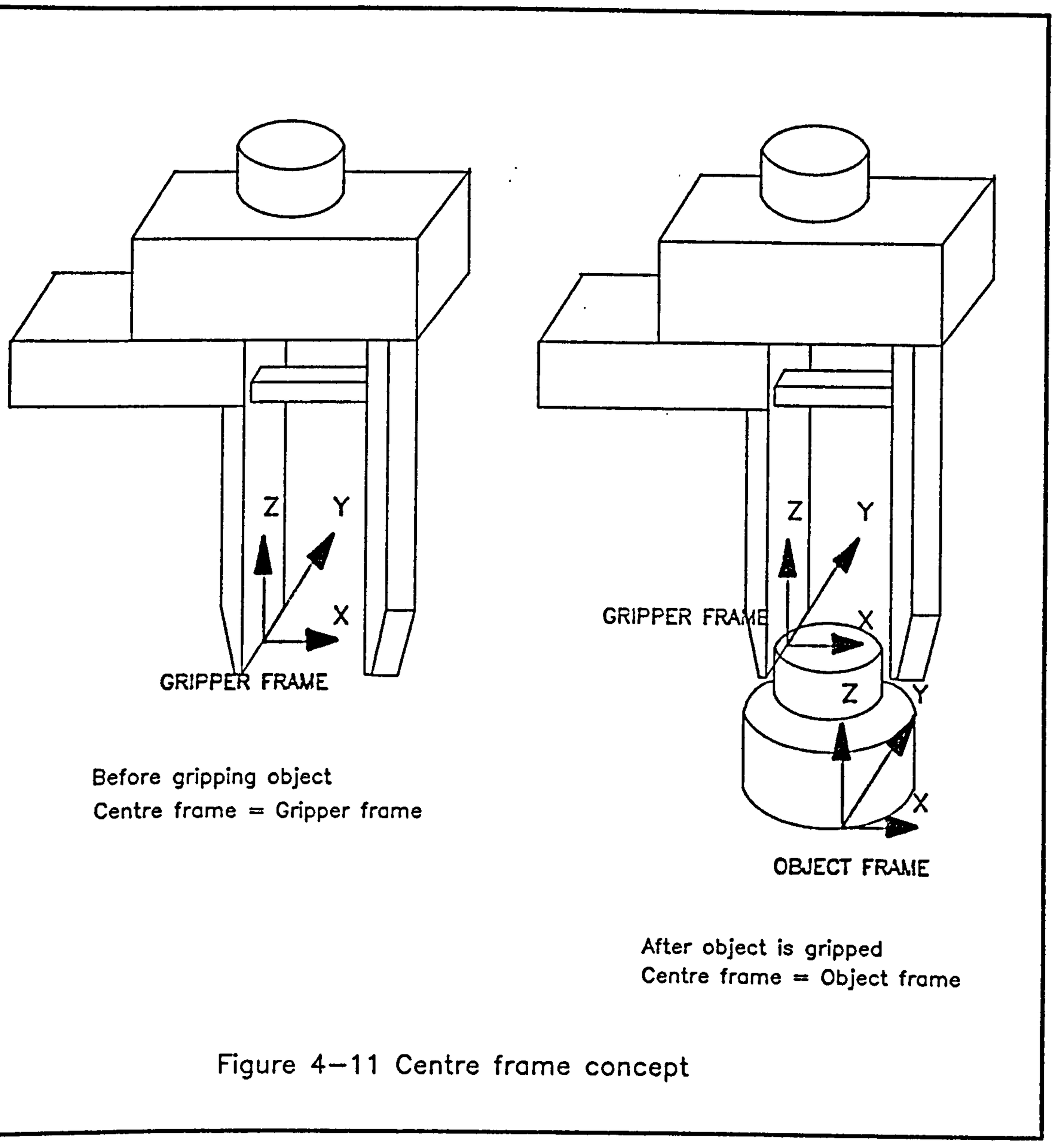



Figure 4–11 Centre frame concept

## 4.3.4 Motion Specification

Typically input data methods are also provided to allow robot movements to be controlled in conventional ways such as in "world", "tool" and "joint" coordinate systems. For example, the user may specify the desired position of the tool centre point in world coordinates with the robot simulator performing the inverse

transformations required to determine all the joint extensions of the manipulator. A number of such positions can be stored as a series of sequential events in a manner analogous to the teach method of robot programming. Subsequently, a time based simulation can be executed to represent the movement of the robot (relative to it's peripheral equipment, such as machines, feeders, conveyors and components), between these various workplace coordinate positions. During this simulation process, interference detection software can be executed so that collisions in the workplace can be flagged. This is inherently possible through the use of Set-theoretic or Boundary Representation modelling techniques, although it can be extremely time-consuming.

As stated earlier, the present generation of robot simulators, can be considered essentially to be kinematic in nature, although commonly an estimate of the cycle times related to various workplace movements can be obtained by the assignment of velocity information to the manipulator model. Thus the dynamic characteristics of the robot and it's workplace elements are not usually accurately modelled. For example, in a high speed contouring application the real manipulator will be subject to backlash, deflection of manipulator links, following error, etc. However, facilities for obtaining cycle time estimations can be particularly useful as this information can be used in investment appraisal and resource planning exercises.

### 4.3.5 Animation

Normally, robot simulation systems can offer capability analysis including joint violation checking, reach verification, cycle time estimation and collision detection. When robot motions are specified, animation is essential for performing the robot capability analysis. The incremental method and the swept-volume method are the two known methods for collision detection.

The swept-volume method exists in current solid modellers, but its application to interference checking is still a research topic [Leu, 1985]. The principle involves the computation of the volume swept by the robot and checks if any object lies within this swept-volume. The advantage of this method is that checking has to be done only once. However, there are major shortcomings with this method. The computation of the appropriate representation of the swept volume creates major difficulties. In addition, we are often concerned with several moving objects with a degree of unpredictability in their movement (at least as to when it might happen). The swept volume technique could not be used in such instances. It should therefore be consider only as an initial check. If any collision is likely to happen then the incremental method should be followed as an accurate check.

Checking of interference with the incremental method is performed at every small increment of robot movement. Interference checking based on this method is readily available if a solid modeller is used. The checking is done incrementally for collision between any number of

specified objects, and although this method is relatively simple, it is very time consuming when there are many objects involved.

If any potential problems or difficulties are identified, the programmer can modify the planned robot motion before the robot program is post-processed and downloaded to the target robot. Some robot systems are extended from robot workplace layout design tools to off-line robot programming tools, and include a robot language post-processor. Methodologies involving post-processing will be discussed in the following chapter.

This chapter has attempted to briefly outline the way in which current robot simulators work and to identify shortcomings and potential for future development. This forms the basis for future chapters where an architecture of the type described is used as the basis for creating an enhanced off-line programming environment featuring ease of use interface facilities, novel calibration tools and novel facilities for product design integration.

# CHAPTER FIVE

# METHODOLOGIES OF POST-PROCESSING FOR OFF-LINE ROBOT PROGRAM GENERATION

## 5.0 Introduction

The post-processor is an important element of an off-line robot programming system, as it allows the commercially available robot simulators (design tools) to be used as off-line robot programming tools. The post-processor translates output statements from a robot simulator to a target robot language. Typically the sequence of motion and data required to drive the robot can be transmitted through the use of a serial or parallel data link between an off-line computer (where the proposed system is usually designed and simulated) and a robot controller. When using contemporary systems the post-processor translates from the model output into a vendor specific robot language, and extracts location information from the simulation model. The design of a general post-processor which is capable of translating the output of multiple robot simulators to multiple robot programming languages is an extremely complex problem and requires further consideration (discussed later in this chapter). Figure 5-1 describes a general off-line robot programming system.

There are two basic ways of describing robot movements in robot control systems; one describes the manipulator movements in terms of the manipulator end effector location (in compound transformation or absolute) whilst the other describes the movements in terms of manipulator joint angles. The former approach is suitable for future and present generation robots with a language processor installed in their controllers, whilst the latter is only specifically linked to earlier generation robot languages. In the U.K., there have been several efforts to establish a post-processor for robot simulators including GRASP. There are a number of acadamic institutions working on this topic involving different robots and

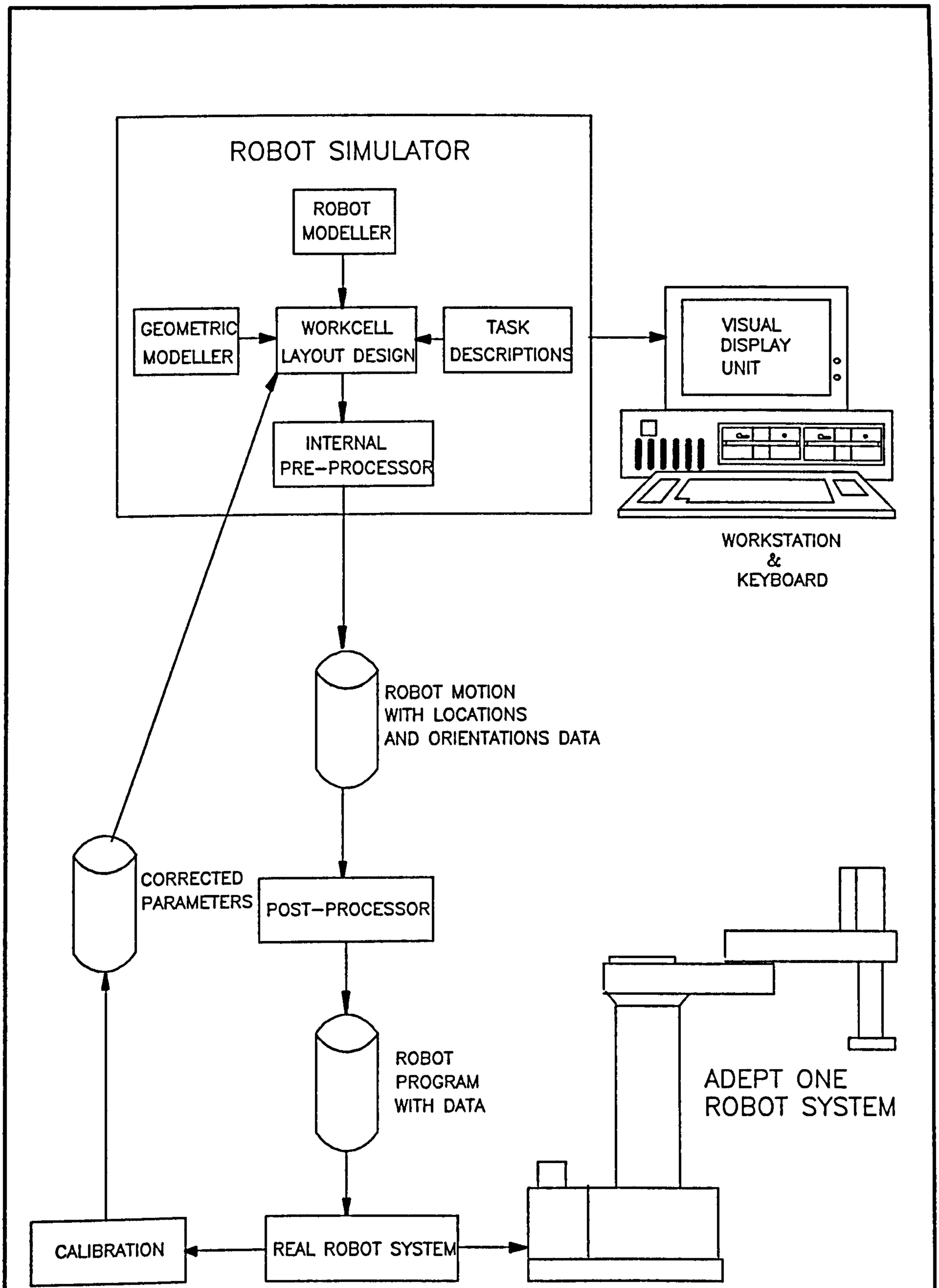



Figure 5-1 General off-line robot programming system

applications. These efforts include the work being carried out in the Department of Engineering at Bristol Polytechnic [Andrews and Cliffe, 1986], where a post-processor is coded to translate the output of GRASP into the high level robot language AR-BASIC for the programming of a Reflex industrial robot. At Queen's University in Belfast [Wright, 1987], work is based on the post-processing of the output of GRASP into joint angles via an APPLE microcomputer to the ASEA IRB6 robot. BYG systems, has also contributed its effort in making its robot simulator more versatile by the development of post-processors for different robot languages. So far there are essentially two different post-processors developed by that company to service, VAL I, VAL II and ASEA AR Language which are suitable for the Unimation family robots, Adept and ASEA series robots.

As a starting point for this research, the author carried out a study regarding available post-processors as these provide an essential link for an off-line programming system. The feasiblilty of generalising the post-processor was also studied. In the early phases, the author devised a specific post-processor which required sufficient knowledge of the chosen robot simulator (GRASP), available target robots (two Adept Ones) and methods for dealing with peripheral devices such as sensors, vision systems and tooling, and the functionality available from the corresponding robot language (VAL II). The findings of this work have been used in formulating the following observations.

## 5.1 Discrepancies Between Robot Simulators and Real Robot Systems

There are a number of facilities available in robot simulators which are not available in real robot languages and vice versa (e.g. the real operation of sensors and vision systems cannot be easily simulated and instead the robot simulators use control conditions to emulate sensors and vision systems).

As previously described, the coordinate frame concept is used in most simulators. In fact the majority of robot simulators use an eulerian angle set (chapter 4) for any kinematic (direct forward and inverse) calculation. However, in the case of the Adept One robot, although the eulerian angle set is used, the coordinate frame uses a right-handed axis set but it's tool reference frame is rotated through 180 degrees about the Y axis and 90 degrees about the rotated Z axis (see figure 5-2).

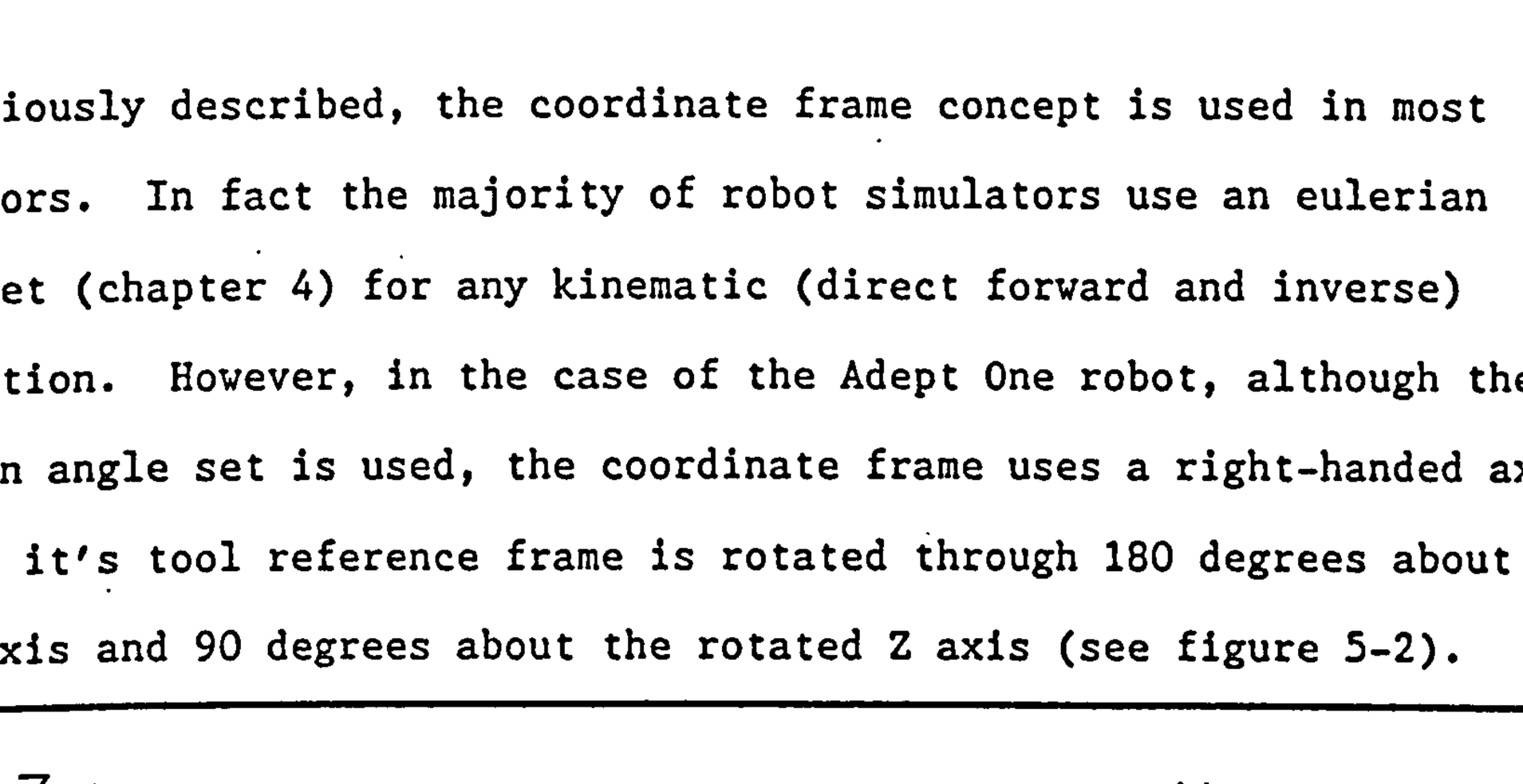



Figure 5–2 Difference in frame concepts between GRASP and Adept One robot systems

In order to overcome this discrepancy, the tool attachment point of the robot has to be rotated about the Y axis by 180 degrees in the robot

simulation model, this leads to the programming of the robot TRACK (a specific name used in the GRASP system which comprises event-dependant motion sequences) to include the rotation of 180 degrees about the Y axis (except all locate statements).

In simulation, the robot end-effector's coordinate frame is the coordinate frame of any tool or object currently attached to the robot flange. However, in real robot systems, the robot end-effector is always referred to as the robot flange. The difference between these two frames should be corrected by using the TOOL statement in VAL II. Nevertheless, the translation between outputs from a robot simulator and a robot language can be made into a one to one conversion.

## 5.2 Theories Used in Post-processing for Different Generations of Robots

Early generation robots such as the ASEA IRB6, do not include a robot programming language, but use very simple command statements and condition looping. Such a robot can only take joint angles corresponding to individual robot joints and so the post-processor must be capable of converting simulation output from cartesian coordinates into individual robot joint angles. The robot controller is not sophisticated enough to deal with high level languages and the data communication capabilities for 'supervisory' systems are too simple for data transmission, although a microcomputer can be used as an external supervisor which can cope with high level to lower level robot language conversion. This approach has been adopted by Queen's University, Belfast [Wright, 1987].

Modern robot controllers are likely to incorporate a language processor such that the robot movement can be driven by language command statements and location data. This makes the development of a post-processor easier. This type of post-processing involves a conversion of the simulated sequences of motion into corresponding robot language statements. This approach was used for the development of the post-processor which is described later in this chapter.

## 5.3 Basic Approaches of Post-processing for Off-line Programs

Post-processing facilities can be system dependent, application dependent or generic. System dependent post-processors are the most commonly used, and function by interpreting and translating program statements of a robot simulator into a specific robot language. In other words, system dependent post-processors are specific to one robot simulator and one robot language. System dependent post-processing facilities are the building blocks for application dependent post-processors which are tailored specifically for a particular application with custom "macro" sequences. Generic post-processors are theoretically capable of translating the output of multiple robot simulators into robot languages for different robot controllers. Since robot languages differ from one another, the creation of a generic post-processor as described is not considered to be an achievable task. The most probable solutions for future implementation are thus expected to rely on standard data formats or enabling technology (see chapter three).

Assuming access to the source code of a robot simulator, post-processing software modules can be written to modify certain data structures used by

the robot simulator to provide new data structures as required by a particular robot. We will see from the following discussion that the person writing the post-processor may only have limited access to the various data structures used internally by the simulator. This utimately can limit the available functionality of the post-processor. For example, in certain cases the vendor of the robot simulator provides limited access data structures via a so called 'standard' interface. In such circumstances after a simulation exercise, internally created data structures can be post-processed into a robot-independent data format, which in the case of GRASP, is known as GRDATA. Here the post-processing function within the GRASP system outputs those entities that have been referenced in the robot TRACK, the program logic and sequence and the gripper or tooling to be used. This is effectively fast post-processing since it does not need to post-process every entity in the simulation model. However, the GRDATA format cannot be used for any other purpose except for off-line robot program translation and to convert a VAL II program into a TRACK. This robot-independent data format (e.g. GRDATA) is then translated into a specific robot language (VAL II in this study) using a specific robot language translator. The concepts involved are depicted by figure 5-3. This approach of post-processing has the advantage of more efficient use of data, reduced processing time, and, more importantly, a reduced chance of error (computation error) during the processing.

For the study purposes and without the supplier's source code, the author derived a system dependant post-processor (see figure 5-4) which was coded in PASCAL and created in a modular fashion. Access to the simulator's data structures was through GRASP model output. The first

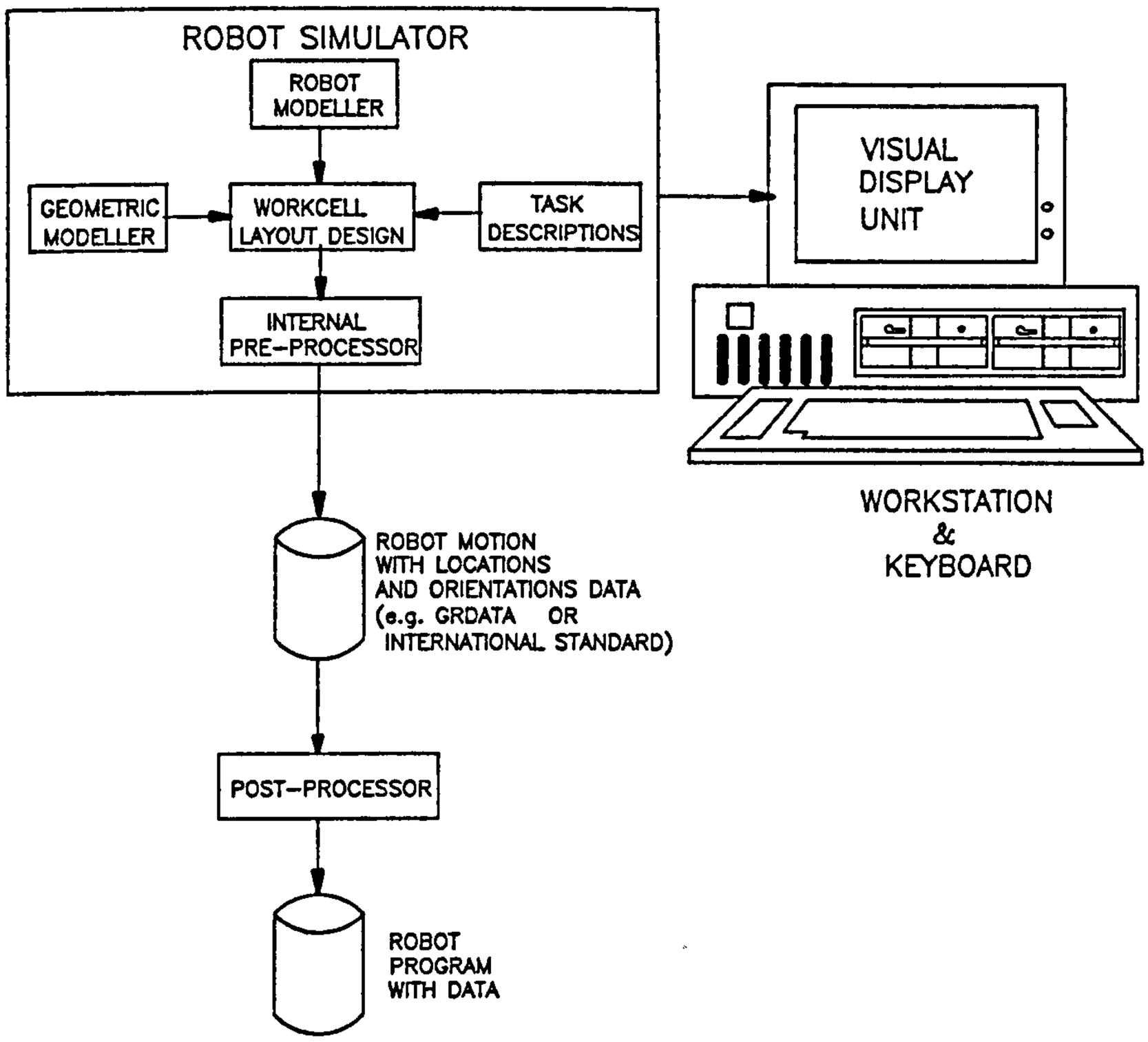



Figure 5-3 Post-processing methodology used by vendors of simulators

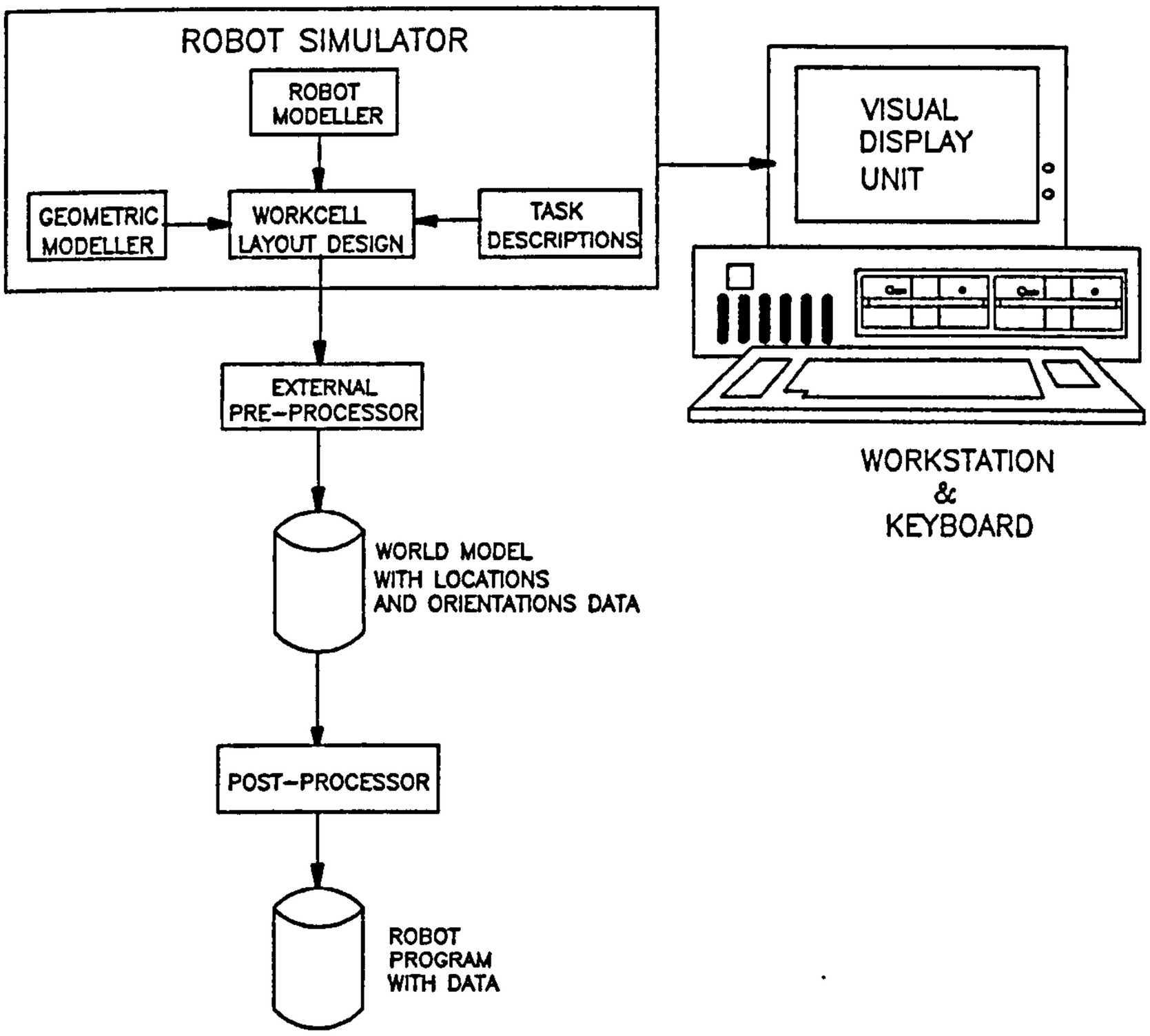



Figure 5-4 Post-processing methodology derived by the author

module extracts spatial relationships between entities in the workplace of a simulation model. This information is stored in a data file ready to be used as a reference data file. The second module of the post-processor translates motion sequences (TRACK statements) into a specific robot language (VAL II). During the study, the post-processing facility has been enhanced through several stages.

## (a) World State Post-processing

During the GRASP simulation, if the programmer stores the robot move positions relative to the robot base, the problem of post-processing the GRASP output source file can be simplified. It would actually be very easy within GRASP to ensure that this always happened on output of data intended for post-processing. However, there are drawbacks with this approach. For example, if any entity being referenced in the TRACK has been re-arranged in the simulation model then the TRACK will be invalid. Furthermore, calibration of robot programs generated off-line and conversion of robot programs into a TRACK are not possible with this approach. Assuming the robot base and global workplace coordinate systems coincide then complex transformations for processing the output source file are not necessary. At this stage, the post-processor consists of three modules (see figure 5-5a, b and c). Appendix A.1 shows an example of the original TRACK and the off-line robot program generated by this approach.

The first module, referred to as SELECTION, operates so that the programmer will be asked whether a TRACK is to be processed and is prompted to provide a name for it. Dependant on the operator responses,

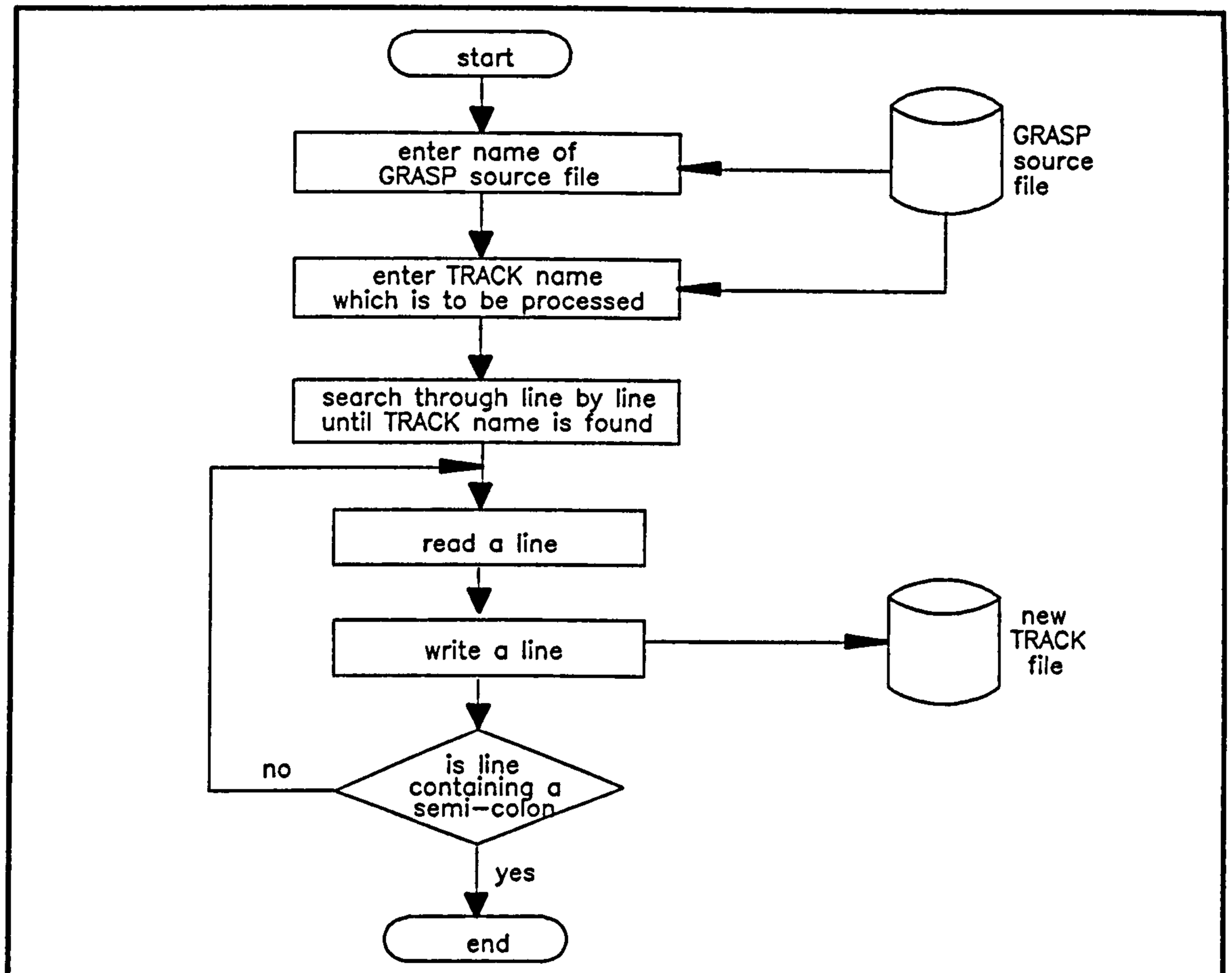



Figure 5-5a Flow chart of SELECTION module

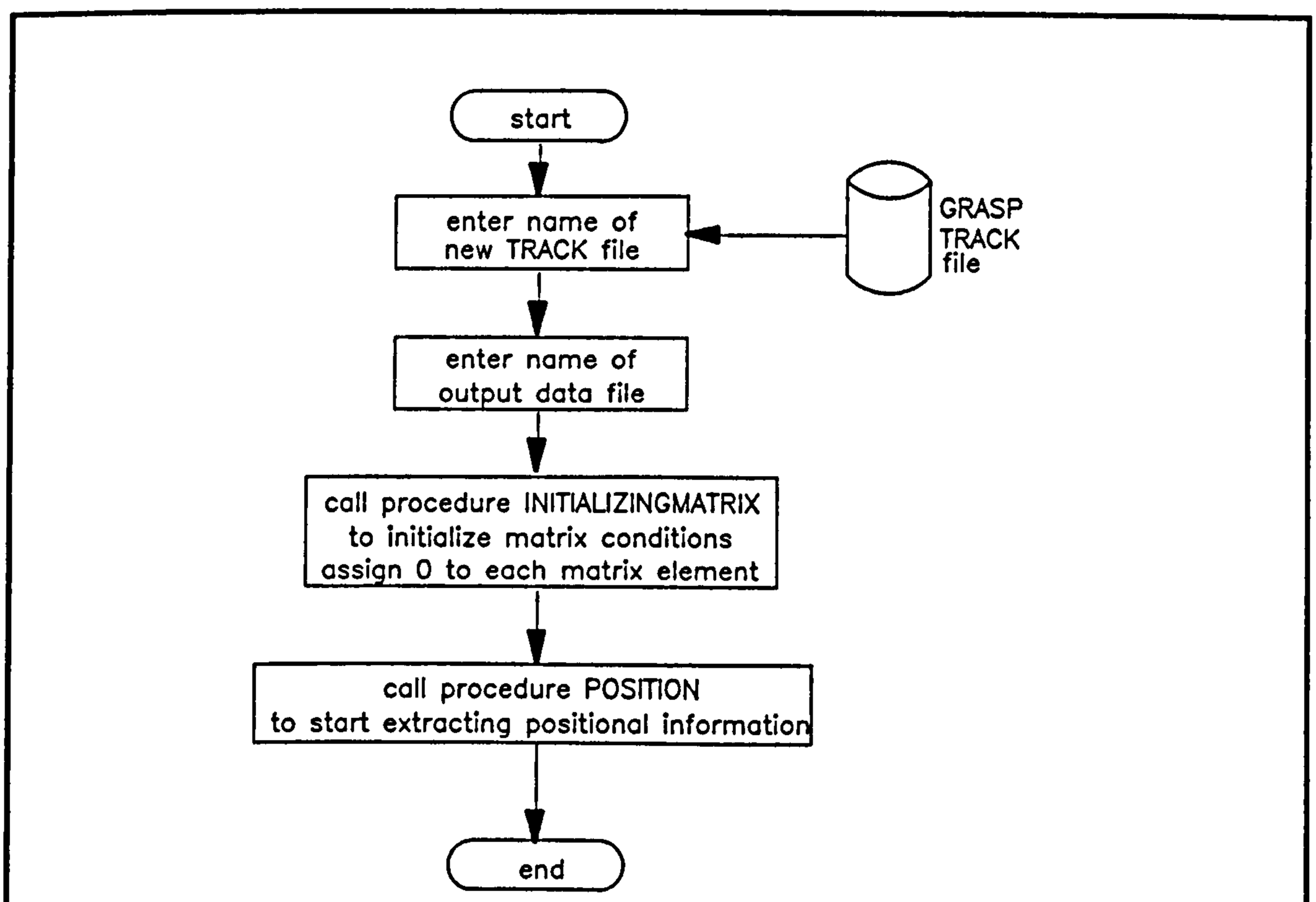



Figure 5-5b Flow chart of EXTRACTION module
- main program

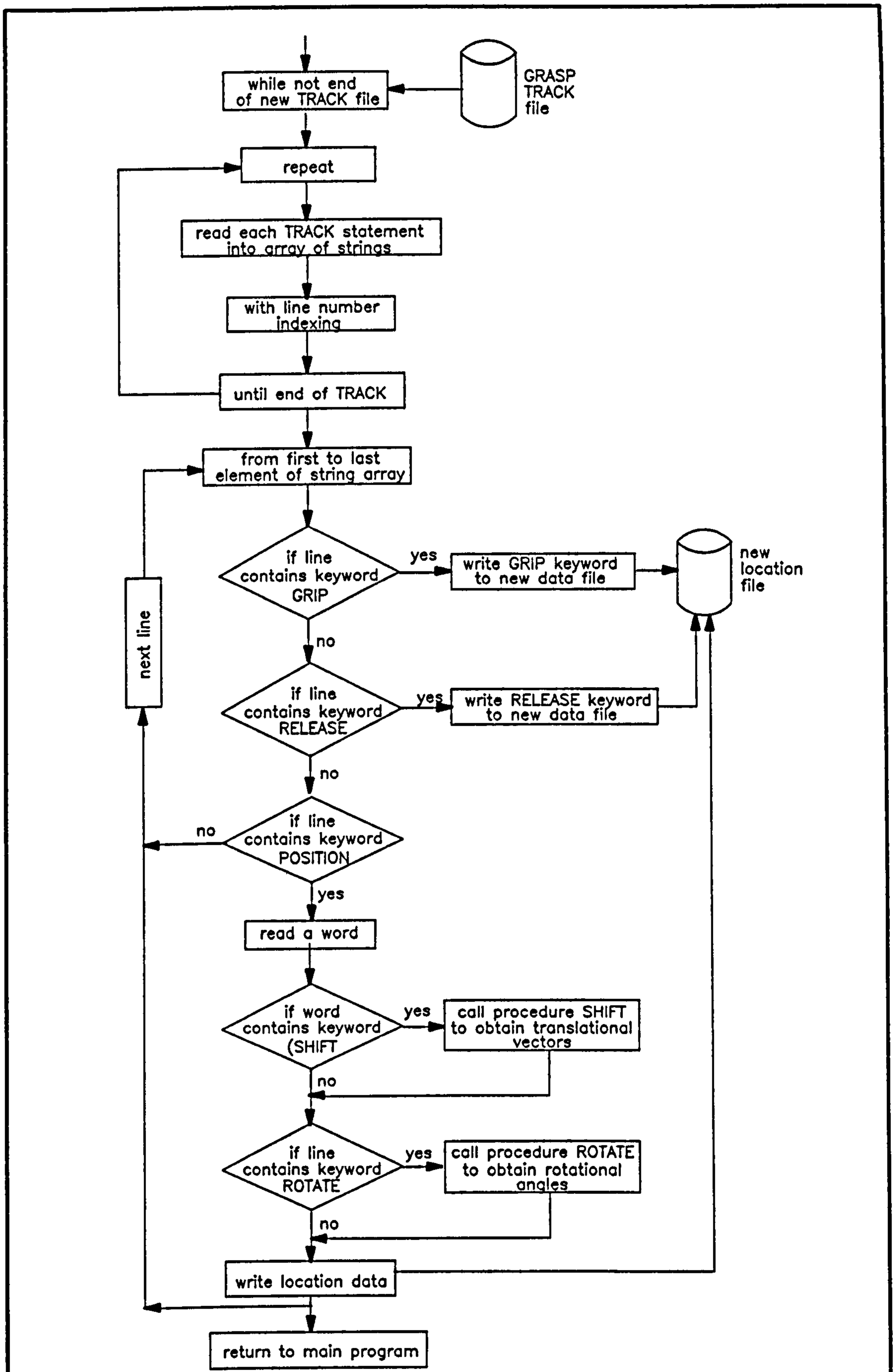



Figure 5–5b Flow chart of EXTRACTION module
– procedure POSITION

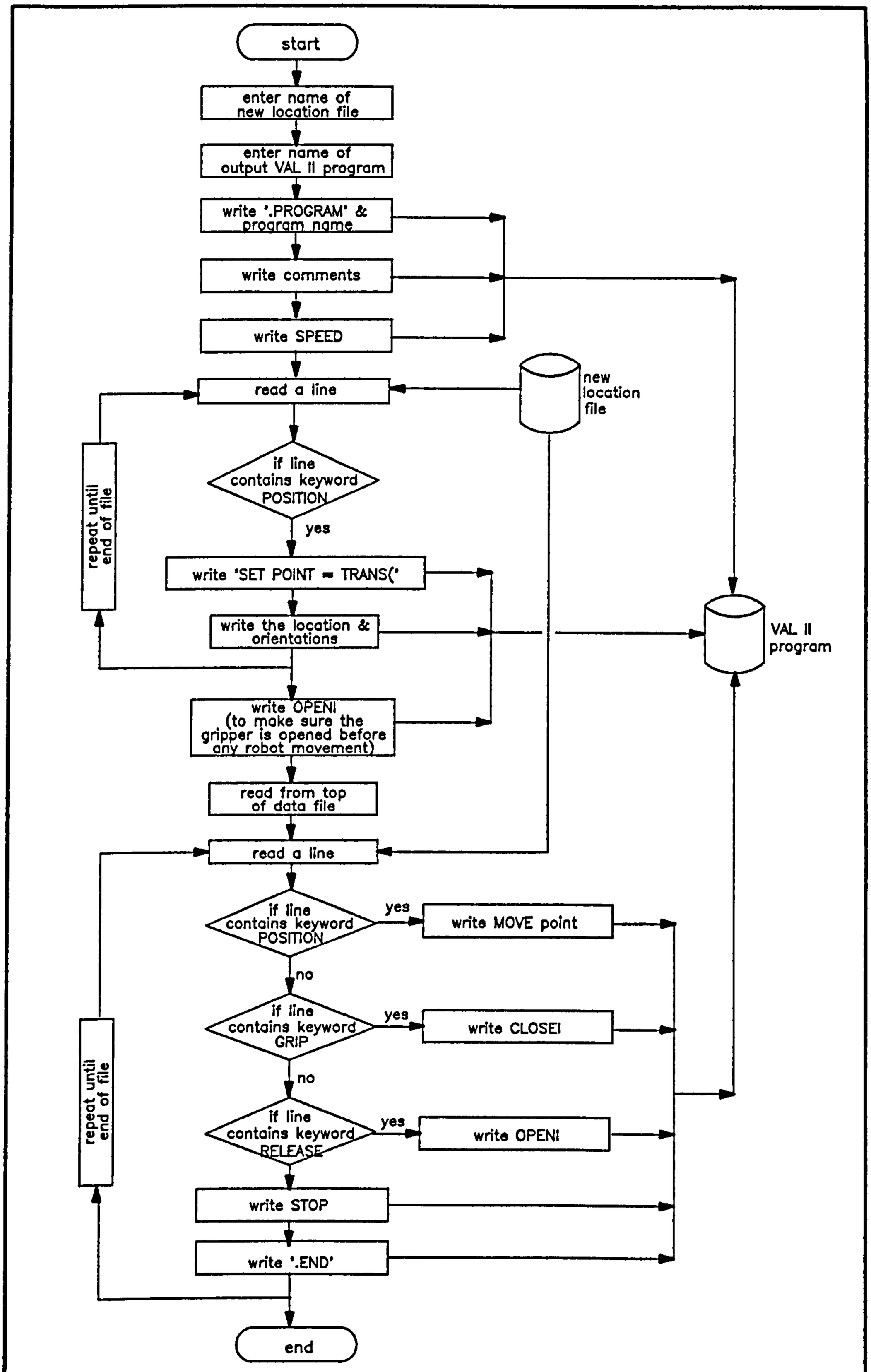



Figure 5–5c Flow chart of VALFORMATTING module

the post-processor will automatically select the appropriate TRACK from the GRASP output source format.

```
Example output from SELECTION
TRACK EXAMPLE
STEP1 : POSITION WORKPLACE (SHIFT X 200 Y 200 Z 800 ROTATE Y 180),
STEP2 : POSITION WORKPLACE (SHIFT X 200 Y 200 Z 780 ROTATE Y 180),
STEP3 : GRIP OBJECT,
STEP4 : POSITION WORKPLACE (SHIFT X 200 Y 200 Z 800 ROTATE Y 180),
STEP5 : RELEASE OBJECT TO BENCH,
STOP;
```

The second module, referred to as EXTRACTION, operates to enable the selected TRACK to be processed further to obtain move coordinates and orientations of the robot end effector and present the data in an appropriate format, as follows:

```
Example output from EXTRACTION
POSITION  200 200 800 0 180 0
POSITION  200 200 780 0 180 0
GRIP
POSITION  200 200 800 0 180 0
RELEASE
```

The third module, referrred to as VALFORMATTING, generates a VAL II robot program based on the accumulated data. This gives the flexibility of processing data and then using a different formatting program to convert into specific robot programs or codes.

```
Example output from VALFORMATTING
 .PROGRAM EXAMPLE_A
; Job number 888
SPEED 50 ALWAYS
SET POINT1 = TRANS(200, 200, 800, 0, 180, 0)
SET POINT2 = TRANS(200, 200, 780, 0, 180, 0)
SET POINT3 = TRANS(200, 200, 800, 0, 180, 0)
MOVES POINT1,
MOVES POINT2,
CLOSEI,
MOVES POINT3,
OPENI,
 .END
```

The World-state post-processing modules are simple but do not facilitate the conversion of VAL II robot programs into TRACKs. Calibration and CAD model updating are also not possible as there is inadequate information about the model. These shortcomings lead to the evolution of a Hierarchical-State Post-processing approach.

## (b) Hierarchical State Post-processing - 'Top-Down' Approach

The second approach to post-processing investigated by the author is similar to the first except that it can deal with more complex situations. For instance CAD programmers are allowed to record move positions in either absolute form (i.e. relative to the global workplace coordinate frame) or relative form (i.e. relative to the coordinate frame of an entity within the model). This has the advantage of permitting the modification of the relative positions of entities in the model without the need to reprogramme the task locations. This post-processor version comprises three modules coded in PASCAL (see figures 5-6a, b and c).

The first of these modules, called PROCESSOR1, is used to extract the spatial relationships between entities and objects in the model. This information is stored in a data file, ready for use in the second module. PROCESSOR1 extracts information using a top down approach (figure 5-7). The data file comprises random access data records, and each record stores the name of the object and the name of its owner object, with their spatial relationships defined within a simulation model presented in a 4 x 4 matrix. These data can be used for calibration and updating of the CAD model. They can also be used as a neutral data format for

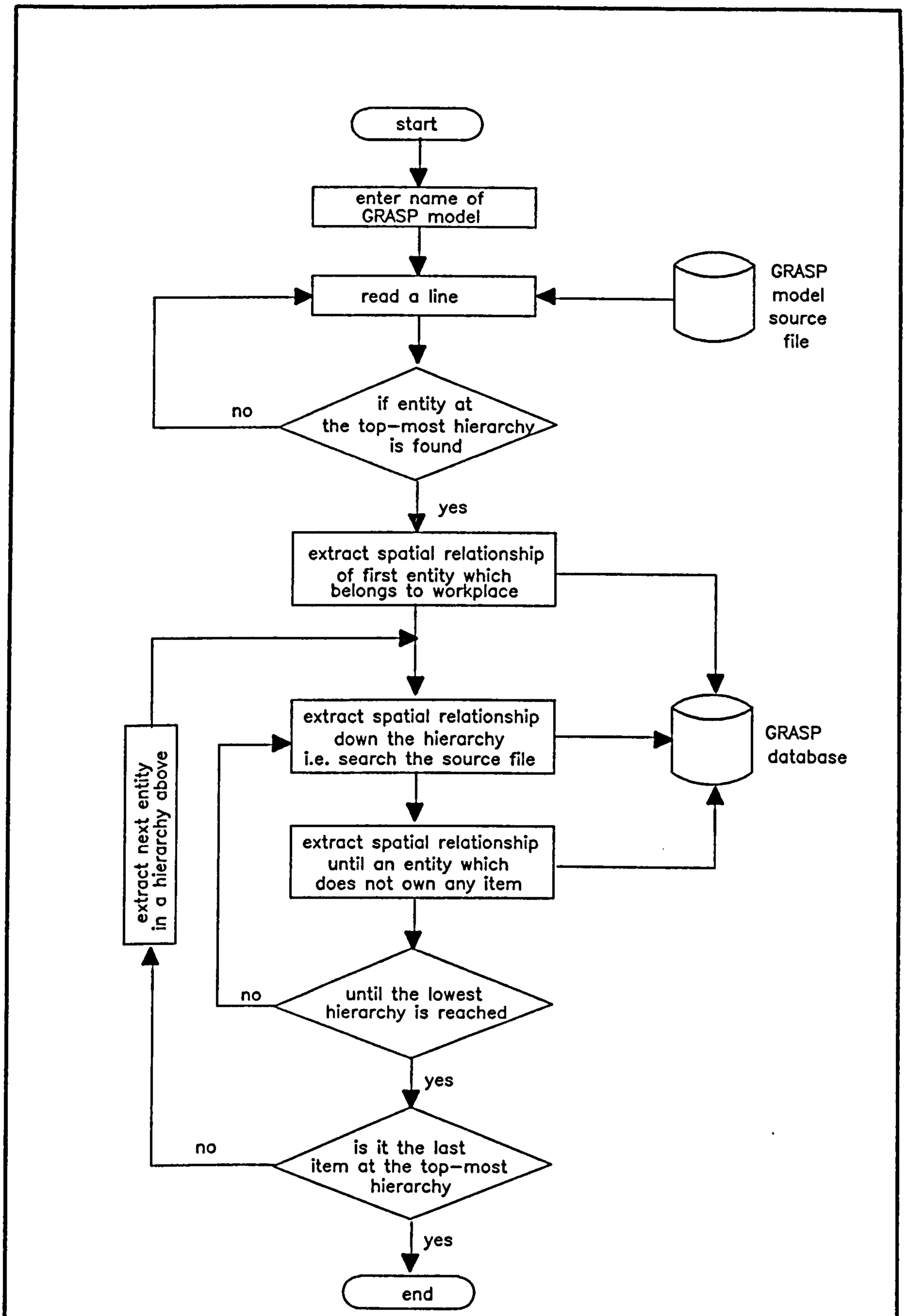



Figure 5-6a Pre-processor adopted with "TOP DOWN" approach

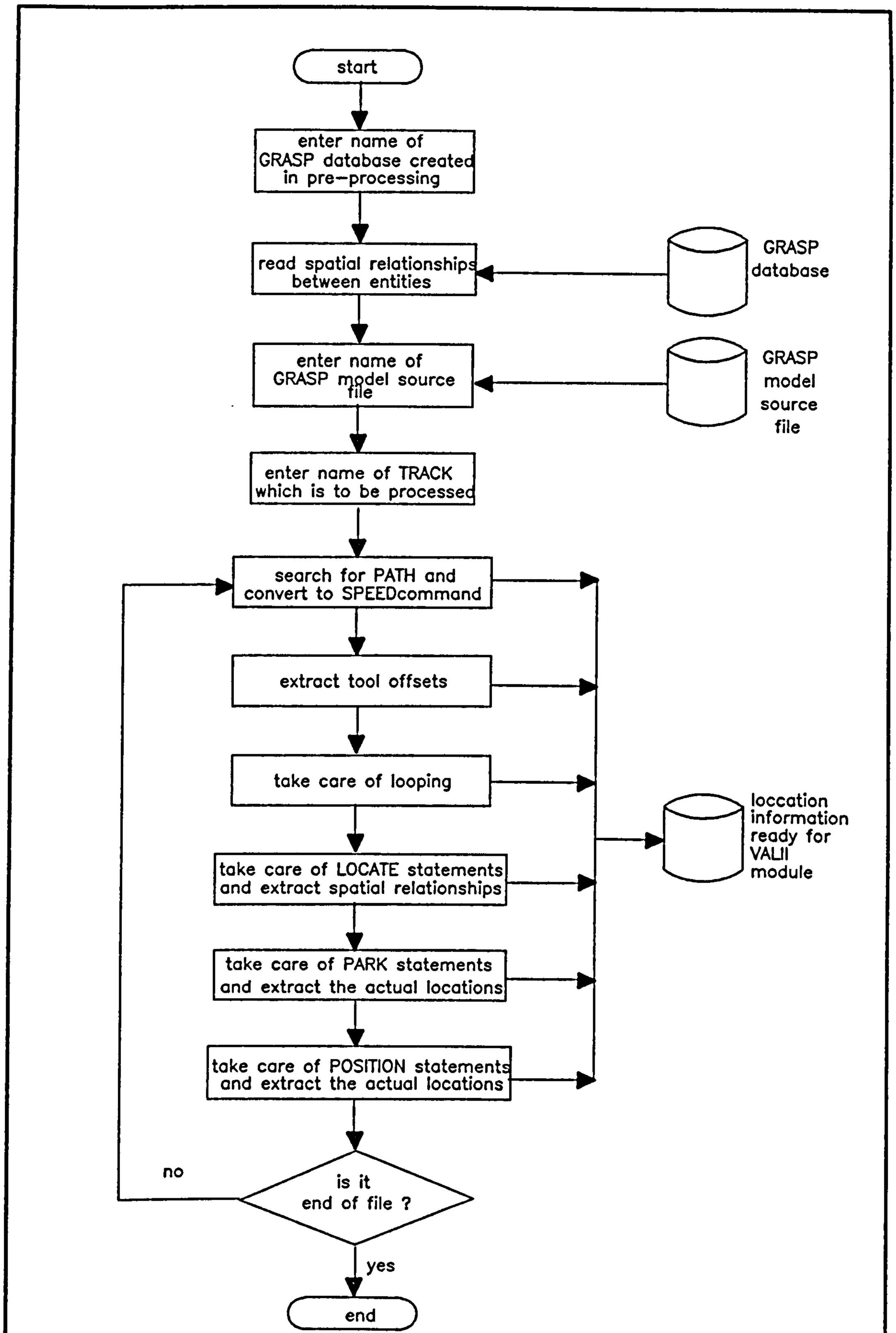



Figure 5-6b Post-processor for "TOP DOWN" approach

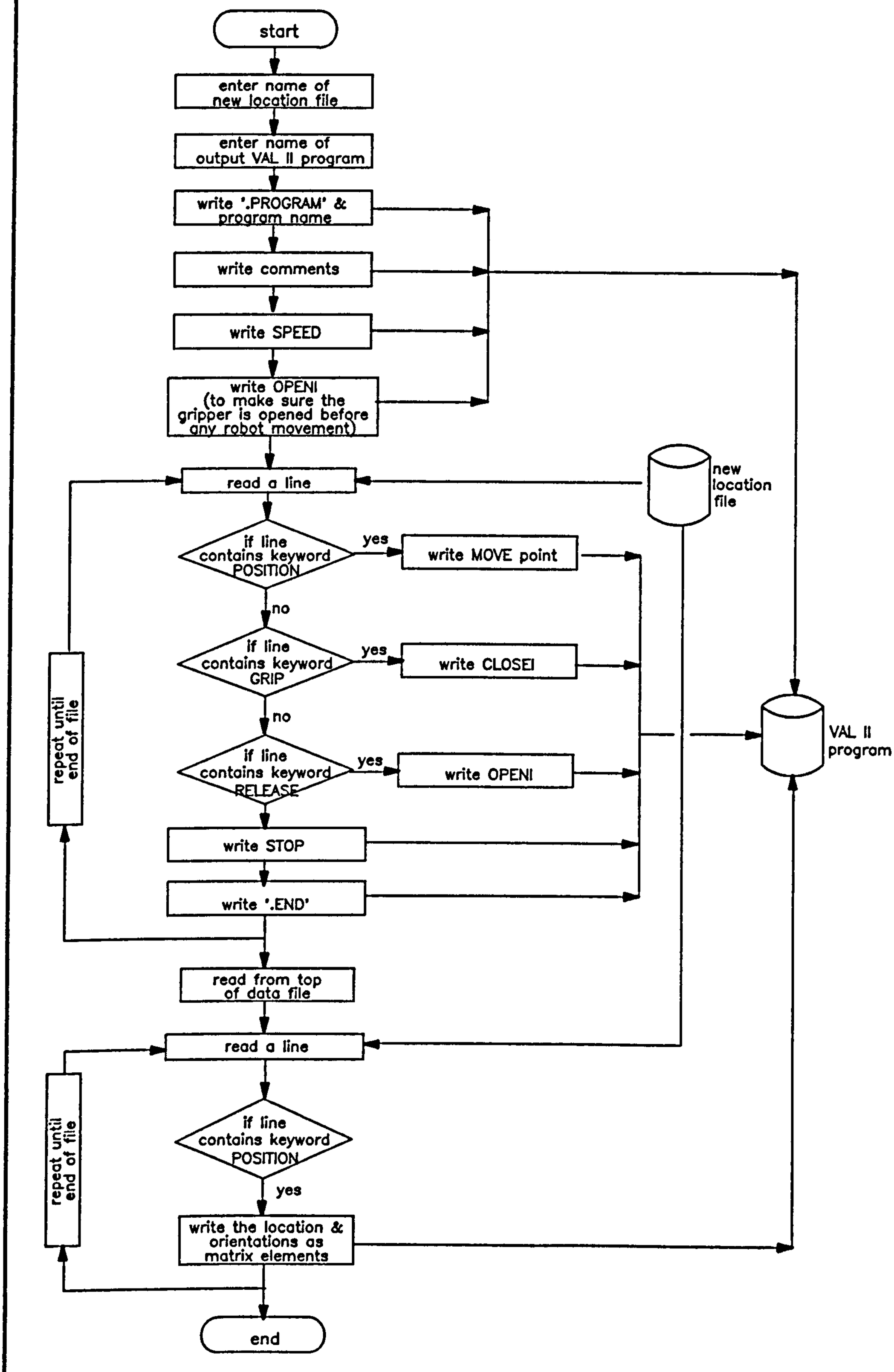
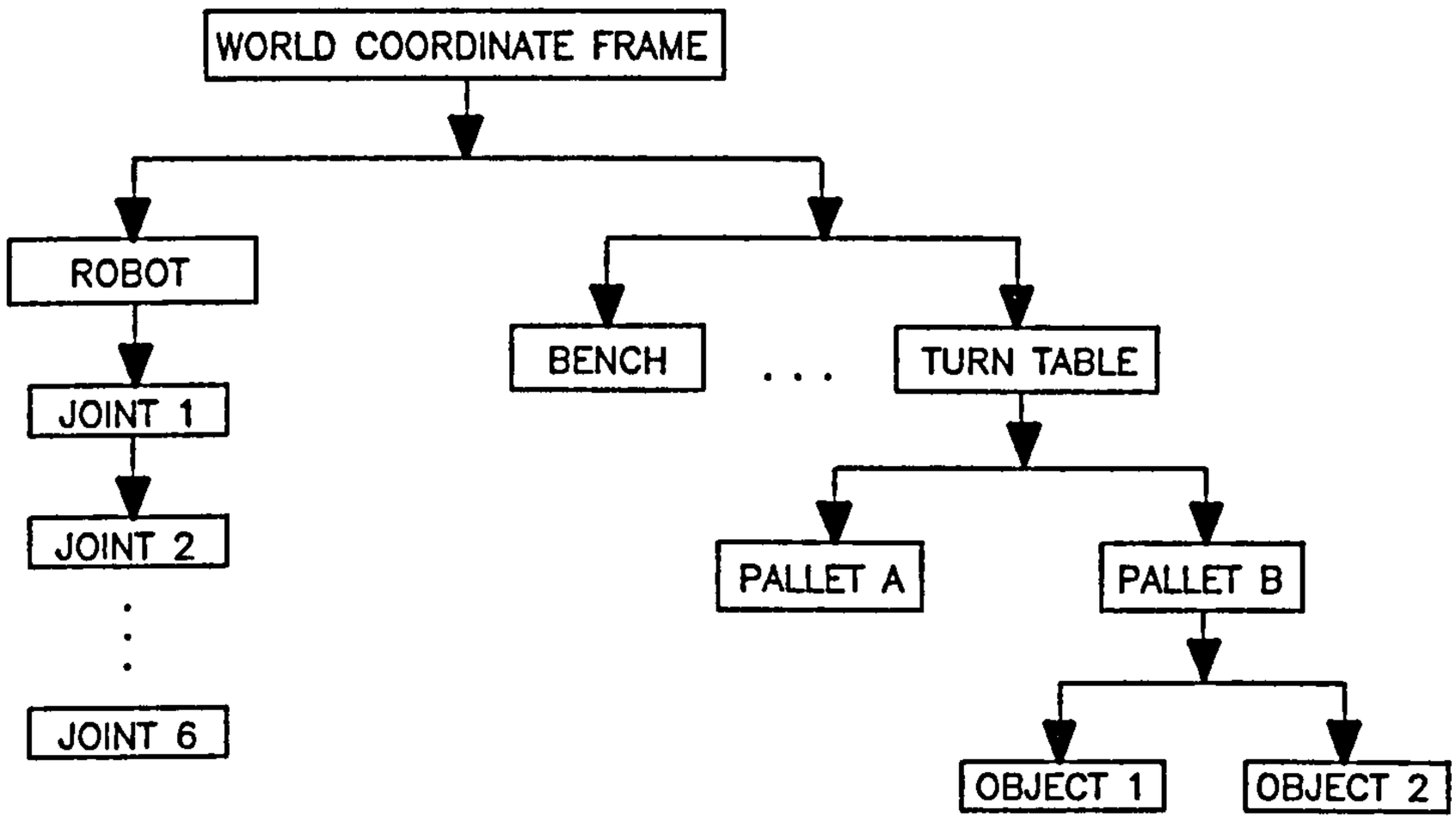
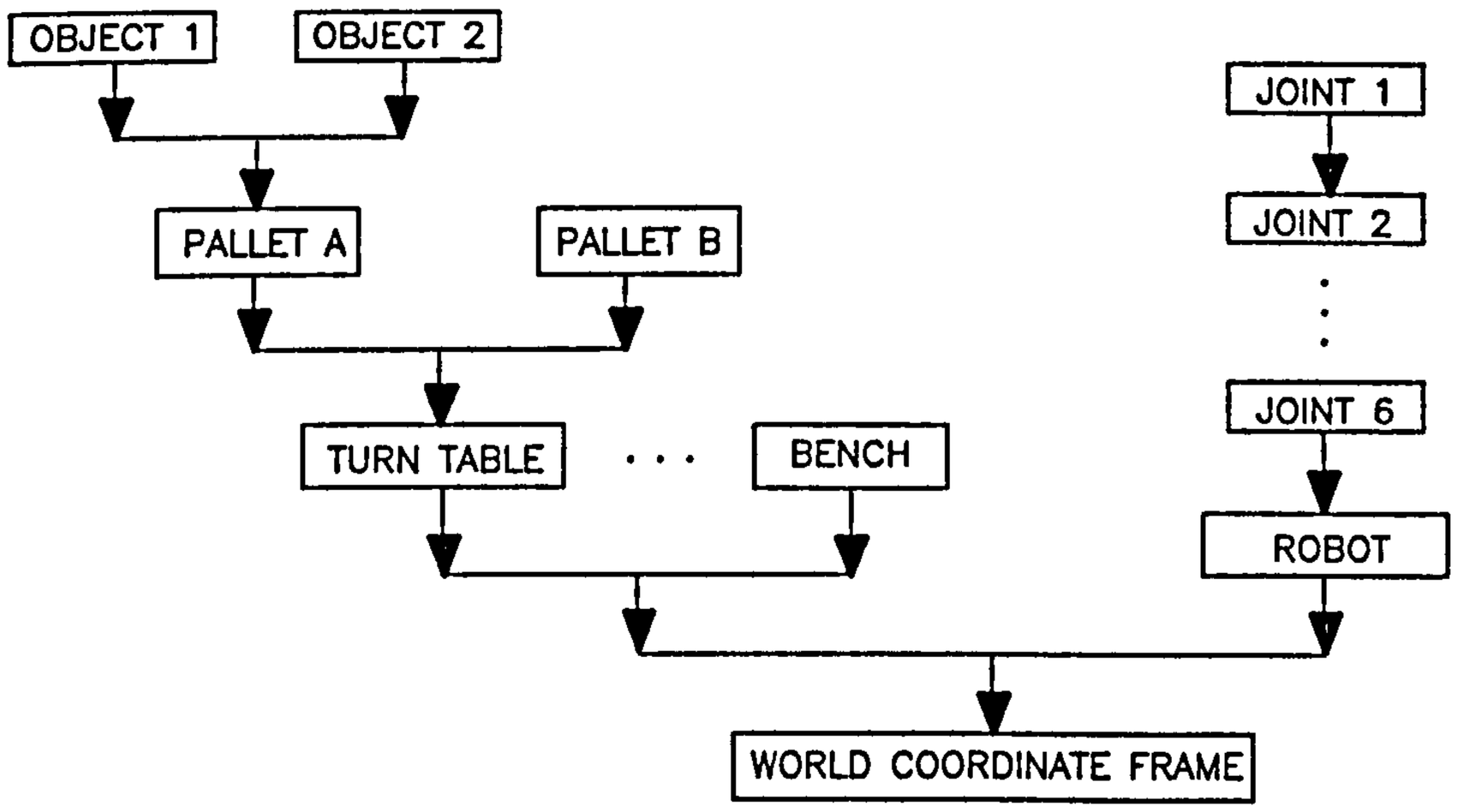



Figure 5–6c Flow chart of VALII module




N.B. In 'TOP DOWN' approach, the spatial relationships of entities are extracted from the top hierarchy to the lowest level. For example, Turn Table related to World Coordinate Frame, Pallet B related to Turn Table, then Object 1 and Object 2 related to Pallet B respectively.

Figure 5–7a "TOP DOWN" post–processing approach




N.B. In 'APPEARANCE' approach, the spatial relationships of entities are extracted according to their order of appearance in the simulation model. It is normally started with smaller items to build up larger entities in the model. For example, the spatial relationships of Object 1, Object 2 related to Pallet A is first extracted as they appear earlier in the model. The order of extraction followed is Pallet A related to Turn Table, Turn Table related to World Coordinate Frame respectively.

Figure 5–7b "APPEARANCE" post–processing approach

data exchange between different robot simulators with similar data structures.

Example of random access data file

| owner object name | object name | spatial relationship 4x4 matrices |
|---|---|---|
| WORKPLACE | PLATFORM | |
| PLATFORM | PLATFORM_TOP | |
| PLATFORM | STAND | |
| PLATFORM | ROTOR1 | |
| ROTOR1 | CYLINDER1 | |
| ROTOR1 | CYLINDER2 | |

The second module, called PROCESSOR2, is used to extract from the robot TRACK the motion sequences, locations and orientations of the move positions relative to a particular entity in the simulation model. Since we are interested in the location and orientation of a position in space to which the robot should be driven, the locations and orientations of such positions are defined with reference to the robot base, e.g.

$$T_{position}^{robot} = \left( T_{robot}^{workplace} \right)^{-1} * T_{platform}^{workplace} * T_{position}^{platform}$$

This module searches through the data file produced in PROCESSOR1 for the name of the object and retrieves the 4 x 4 spatial matrix. This is multiplied by the transformation used in the TRACK movement. The data file is repeatedly searched and the matrices concatenated until the highest level is reached. The final location information is written in absolute coordinates relative to the robot base. The information representing the robot movement is then converted into a robot specific language called VAL II via VALII module which is similar to VALFORMATTING module as described earlier. The VAL II robot program and the location data are produced in one text file (see appendix A.2 for example output).

Since the robot program uses numbers to identify positions instead of the actual entity name used in the CAD model, the conversion of VAL II robot programs into GRASP TRACKs is not possible.

Example of output from PROCESSOR2

absolute position for robot movement is used to drive the actual robot (calculated with respect to the Robot base coordinate frame, for example,

```
.PROGRAM EXAMPLE_B
SET POINT1=TRANS(200, 200, 800, 0, 180, 0)
SET POINT2=TRANS(200, 200, 750, 0, 180, 0)
SET POINT3=TRANS(500, 500, 800, 0, 180, 0)
SET POINT4=TRANS(500, 500, 750, 0, 180, 0)
SET POINT3=TRANS(500, 500, 800, 0, 180, 0)
SPEED 50 ALWAYS
MOVE POINT1
MOVE POINT2
CLOSEI
MOVE POINT2
MOVE POINT3
MOVE POINT4
OPENI
MOVE POINT4
MOVE POINT5
.END
```

## (c) Hierarchical State Post-processing -'Appearance' Approach

The version 3 post-processor produced in this study comprises two modules. The first module, called PROCESSOR11, is quite similar to the stage two PROCESSOR1 module (previous subsection), except that it is modified to increase the speed of processing such that the processing time is reduced to approximately 10 percent of that in stage two approach. This module extracts spatial relationships concerning each object and its owner in the order of their appearance. Figure 5-7 illustrates the difference between the "Top Down" and "Appearance" approaches.

The second module of the post-processor is called PROCESSOR22, which translates the robot TRACK into a VAL II robot program with all the move positions stored in a separate text file (see figure 5-8a and b). This allows compound transformations to be used in the robot program whereas in the previous stage only absolute positions were allowed. Appendix A.3 shows an example of the original TRACK and the off-line robot program with location data generated by this approach.

for example, Compound transformations

MOVE REFERENCE:TRANSFORM

where REFERENCE is the location of the referenced object with respect to the robot base, i.e.

$$T_{reference}^{robot} = \left( T_{robot}^{workplace} \right)^{-1} * T_{platform}^{workplace} * T_{reference}^{platform}$$

TRANSFORM is the object's location with respect to the referenced object

This method allows the off-line generated programs to be calibrated, the simulation model to be updated, data exchange between robot simulators with similar data structures may be implemented, and the actual robot programs can be reconverted into GRASP TRACKs. The format of actual robot programs can be reconverted into GRASP TRACKs by using the VALTOTRACK module created by the author. The VALTOTRACK module reads the robot program and its corresponding location data file which allows locate, path control, movements, and gripper control statements to be converted into GRASP syntax (see appendix A.4 for example output). This VALTOTRACK module allows robot simulation systems to be used in a relatively comprehensive manner. For example, the verification of a robot program, for different workshop layout configurations and for

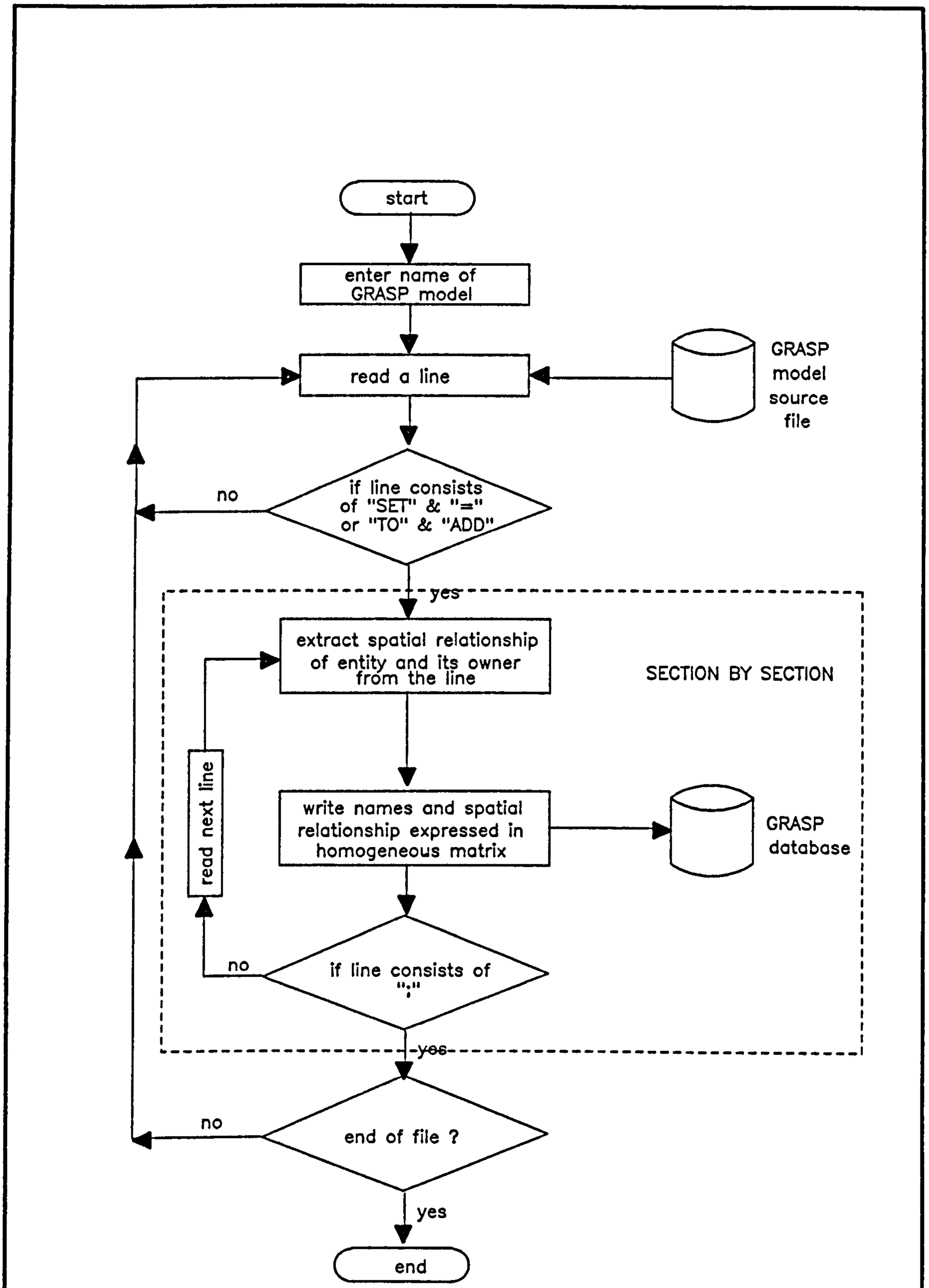



Figure 5-8a Pre-processor adopted with "APPEARANCE" approach

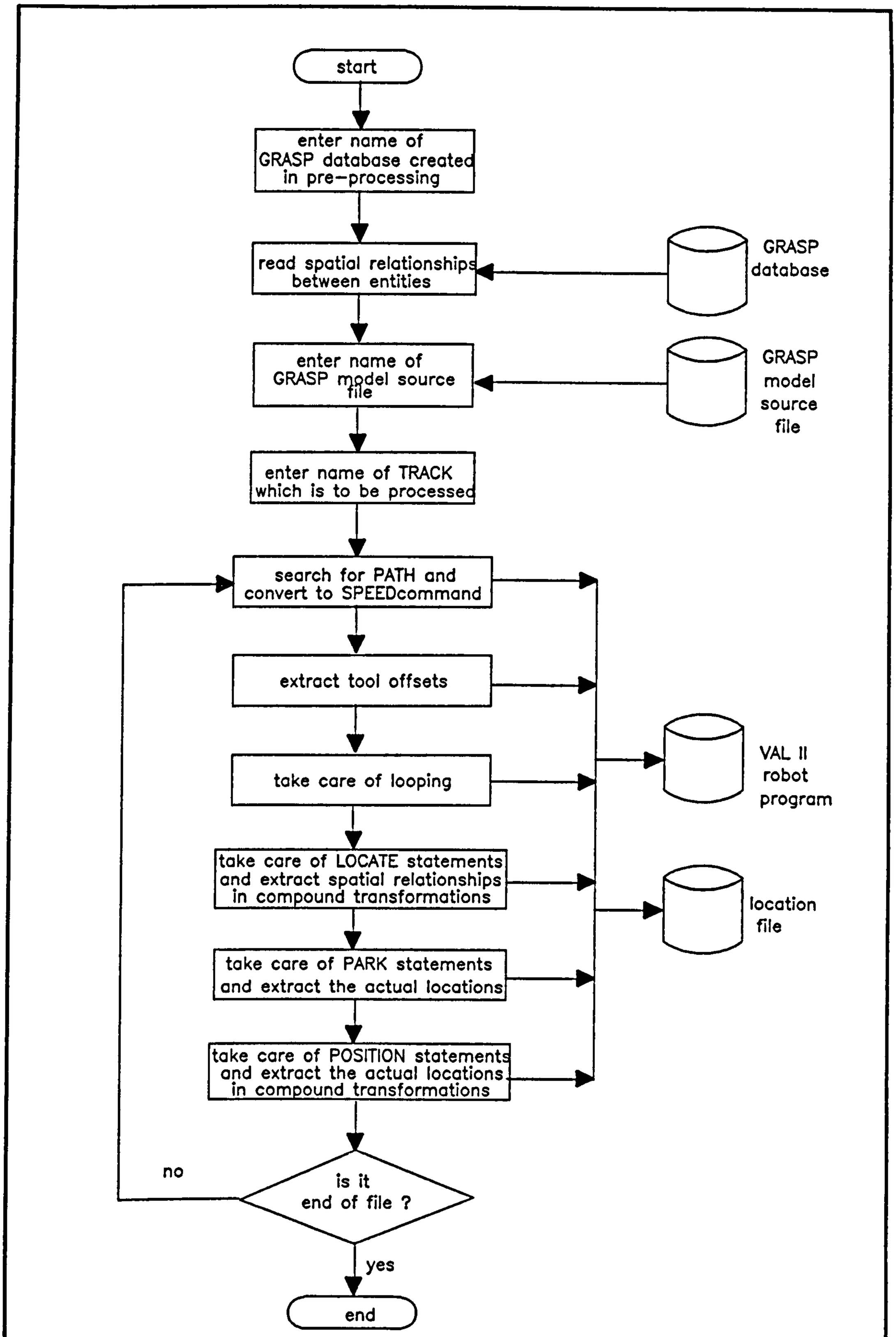



Figure 5–8b Post–processor for "APPEARANCE" approach

different robots to perform the same task, or the same robot for different applications to be simulated and edited. The flow chart of VALTOTRACK is shown in figure 5-9.

## 5.4 Methods of Downloading Off-line Programs to the Robot Controller

When post-processing is complete, the program may be transmitted to the robot controller in one of three possible ways:-

(a) The Robot program can be downloaded via a robot specific medium (such as magnetic tapes, paper tapes, floppy diskettes etc.) which can be transferred manually from the design office to the shopfloor. In the context of highly automated CIM systems, this may require unwanted human intervention and may lead to increased machine set up times.

(b) If it is desired to load programs electronically through an available robot specific medium interface, then a communication protocol must be implemented in software in order to interface to the robot controller. The use of an RS-232 link to facilitate the communication between the CAD system and the robot controllers is quite commonly used in industry. However, this approach is too rigidly fixed to the specific machines and will result in a low level of portability (or configuration flexibility) as CIM systems become more commonplace and/or complex.

(c) A more flexible approach could be based upon the use of the Manufacturing Automation Protocol (MAP). This new enabling technology provides control data, monitoring status transmission etc, and

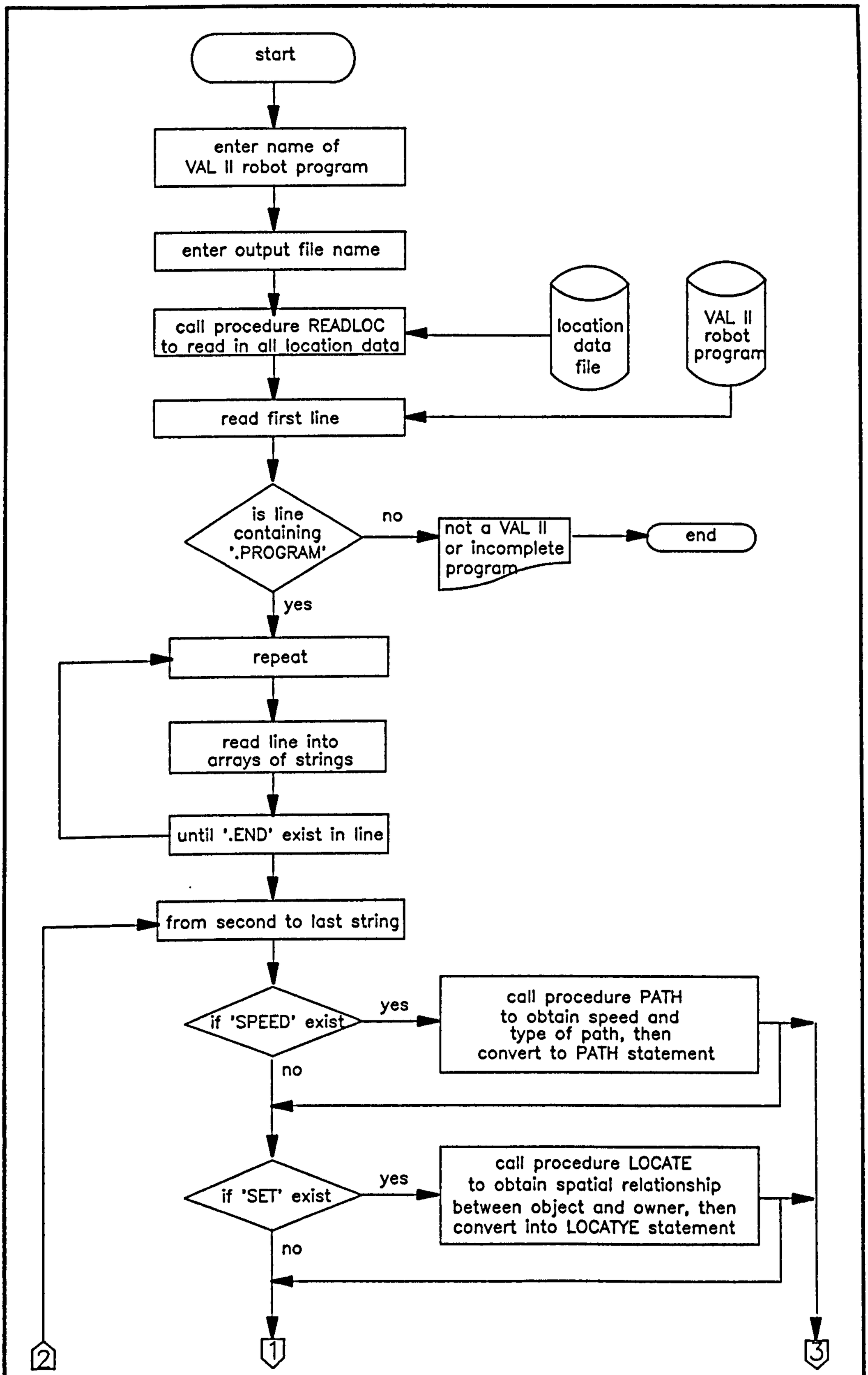



Figure 5–9 Flow chart of VALTOTRACK module

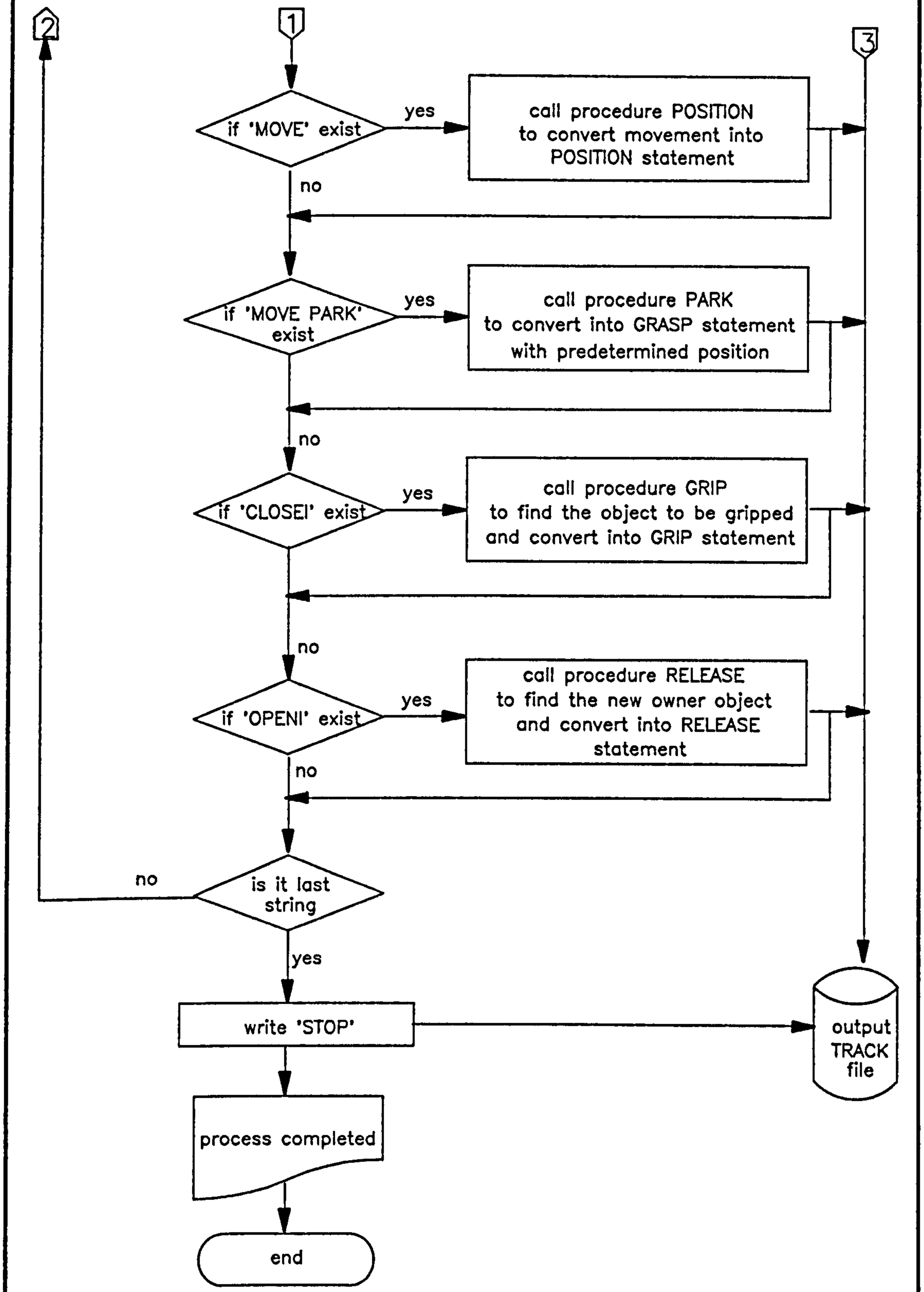



Figure 5-9 Flow chart of VALTOTRACK module (continued)

potentially there is a much reduced need to write specific protocol conversion for each robot. Furthermore, developments with respect to layer 7, the application layer, including the manufacturing message services (MMS) [MMS Draft 6 document, 1987] which is being evolved in the wake of the MAP/TOP initiative should be monitored and could form the basis of a third generation command language.

Although a MAP broadband network was available in the Department of Manufacturing Engineering during the period of this study, for various reasons (the major one being the Prime 550 computer, used by the author for prototyping was not connected to the MAP network) the approach (b) was used in this work.

## 5.5 Generalised Approach

Post-processors can be system or application dependent. Some of these post-processors have been designed for use with an application dependent robot simulator. These post-processors normally have "macro sequences" built in for the particular application. However, most of the post-processors are system dependent, which means they can only be used for one robot simulator and one robot language. If an internationally accepted standard data format existed, it would be wise to develop the post-processor in two modules (see figure 5-10), the first one being responsible for converting information from the output of a robot simulator to a neutral or standard data format, whilst the second module translates the standard data format into a robot language. This reduces the effort in developing a post-processor for each set of robot simulators and robot languages. This idea has been incorporated in the

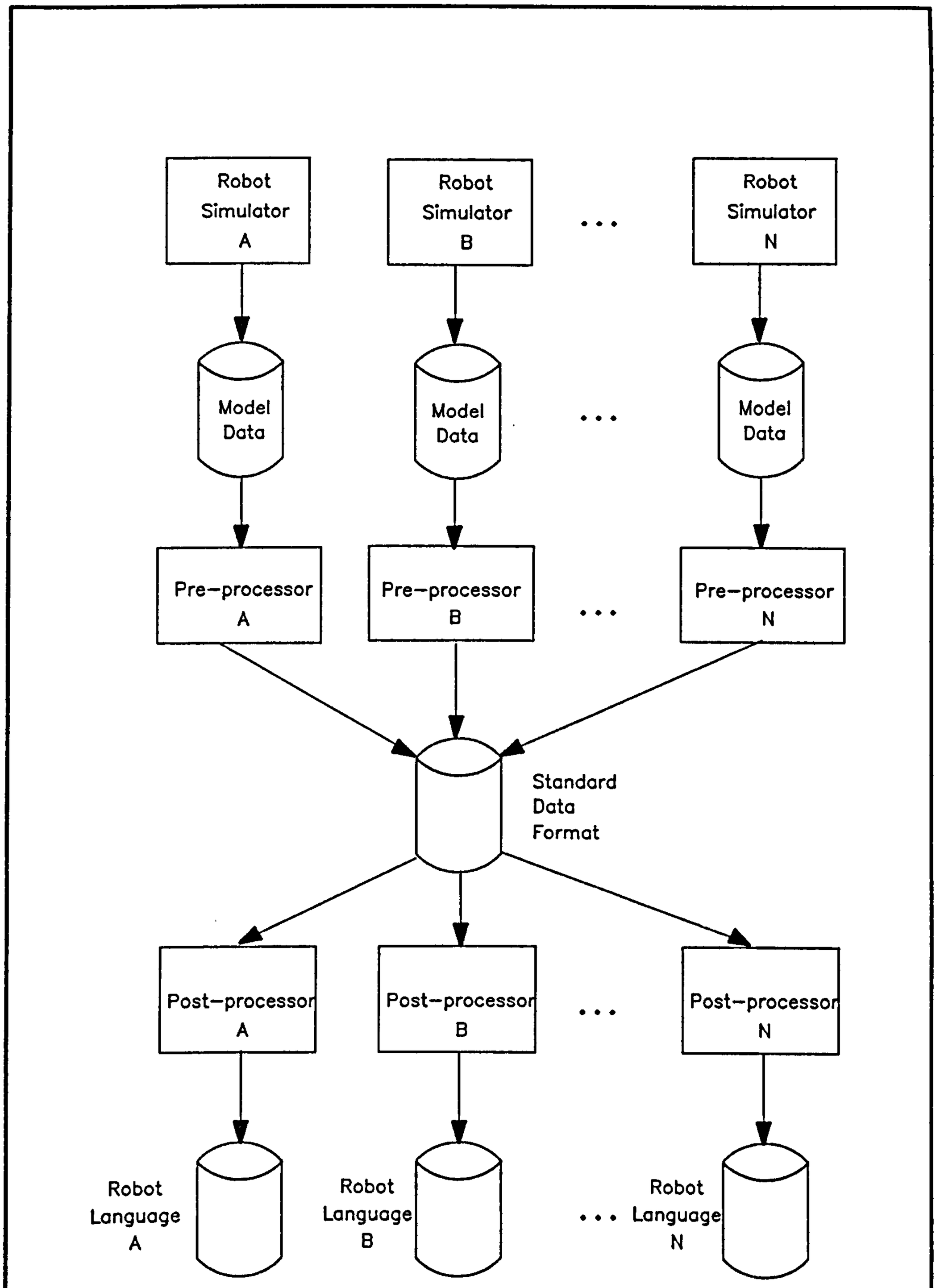



Figure 5-10 Generalised post-processing methodology with standard data format

processors created in this study and described in this chapter. However, standard data formats are not yet available. Thus a random access data file with consistent data structures has been used through providing a prototype neutral data format. Suppose, there are 10 robot simulators and 10 robot languages, then there is a need to write 10 pre-processors and 10 post-processors. However, with no internationally accepted standard data format (or neutral data format) there would be a need for 100 processors. This standard or neutral data format approach can be regarded as a generalised approach. There is a further possibility of generalising the post-processors so that common command statements are used in robot languages and post-processors.

The success experienced with this study indicates that similar effort can be applied to different systems. Of course, the total effort will be significantly reduced if standards exist. However, at the moment, there is no standardisation that has been successfully achieved in the field of off-line robot programming. Apparently, there is a lack of user enthusiasm for standardisation, and perhaps this is primarily due to the fact that industrial users apply robots to perform simple, repetitive, or mass-production tasks which do not require frequent reprogramming. Secondly, major users may well have developed their own interfaces/communication between their CAD system and their robotic devices. However, with the advent of CIM methods this situation is likely to change significantly over the next few decades.

When considering the 'level' at which standardisation is likely in the near future, it is concluded that the robot interface data format is the most likely initial target for standardisation with IRDATA and the MMS

robot companion standard both offering an important step in the right direction. When internationally agreed robot interface data formats are available, it will be the robot manufacturer who will be responsible for adopting appropriate controller protocols. For commercial reasons, manufacturers have appeared to resist standardisation initiatives and this has resulted in user driven initatives in the area, such as MAP. As technology advances, new enabling technologies such as MAP and TOP could provide tremendous capability for information transfer, linking different devices from different vendors, which would truly enhance Computer Integrated Manufacturing and the off-line programming of robots.

# CHAPTER SIX

# IMPROVEMENTS IN MAN-MACHINE INTERFACE

## 6.0 Introduction

The premise on which off-line programming systems have evolved and are marketed is that they are easy to use. Certain suppliers of robot simulators claim that their system is user friendly so that most operators, with no previous programming experience, can design workcells after one day's training. These suppliers have also claimed that an operator can become an expert in two to four weeks compared to six to nine months for a general purpose CAD system [Robotics World, 1986a]. This may in fact be the case in simple application areas but to date very little literature is available comparing off-line and conventional robot language programming procedures so that reliable conclusions cannot be drawn.

In the absence of information one can only suggest that the user interface to the robot simulator should be as simple as possible with an extensive use made of model libraries, containing not only the robots but also commonly used workplace elements constructed from standard parameterised primitive building blocks. Furthermore, where a product design interface cannot be established, facilities could be included for the user friendly description of products encompassing geometric and manufacturing/assembly sequential information.

As a generalisation one could conclude that off-line programming will be more easily justified in circumstances where complex robot tasks are required and/or where the batch manufacture of large product families are involved. However at the present time inadequacies in the simulators' capabilities in dealing with sensory feedback, exception handling and the debugging of task programs implies the continued use of on-line programming

for these aspects where a language such as VAL II combined with a teach pendant has the necessary flexibility.

The man-machine interface to proprietary robot simulators can be considered within two major groups of input facilities; one being related to the solid modeller, with the other related to the robot task description.

As discussed earlier, the solid modelling facility within GRASP is based on the boundary representation method using a combination of primitive solids and generalised polyhedra. A robotic device or workplace entity is modelled by defining geometric, spatial, and functional relationships. Definition of geometry using primitive shapes is straight-forward, but generalised polyhedra present a much more substantial problem which requires the assistance of screen interaction techniques. Spatial relationships between objects in three-dimensional space inevitably cause difficulties which can be addressed by an improved user interface or by improved, but potentially expensive, computer graphics techniques. Functional arrangement refers to the need to encapsulate kinematic relationships between model items within the data specification. For inexperienced users or complex situations this can be an extremely daunting prospect.

Once the model has been created there still remains the often complex task of building the kinematic simulation itself. Thus methods of enhancing the user interface to GRASP have been studied and software modules produced to assist both experienced and non expert programmers in dealing with solid modelling and task programming.

## 6.1 Computer Assisted Model Building

Building a three-dimensional model is a requirement which robot simulators have in common with CAD systems which are aimed at the more traditional aspects of engineering design. Simulators which are an extension of such systems are therefore likely to be able to call upon highly developed man-machine interfaces for the specification of geometry. A typical approach would be a parametric design programming language often used to ease the generation of models of families of parts. GRASP, a stand-alone simulator with its own modelling system, does not have such facilities and thus provides both the need and opportunity to study parametric design from the particular viewpoint of robotic equipment. The distinguishing feature here is the need to easily define the functional relationships between the components of the equipment. An example would be in specifying geometry of a gripper such that it will have the correct relationship to the robot on which it is to be mounted. (In the case of GRASP, this involves careful modelling about a set of axes which will be made coincident with those of the tool attachment point). Careful modelling is again required when the object being modelled is a mechanism which will be required to assume different configurations under various kinematic conditions. Modelling the kinematic structure of the robots themselves is of course a central aspect of the simulators and hence is well catered for. However, the kinematic modelling of grippers, assembly jigs, etc. can be extremely difficult, and a parametric approach could provide a solution.

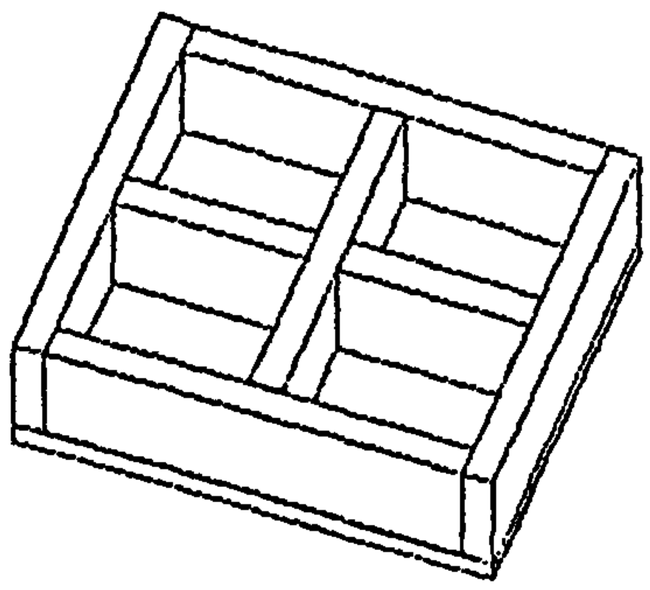
In this study, solid modelling software modules have been coded in Pascal which can assist both experienced and inexperienced programmers when

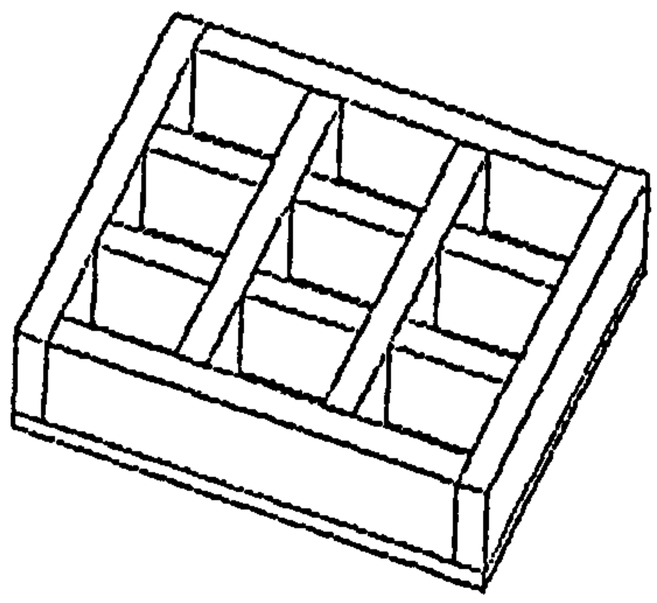
creating models of entities which are commonly encountered in a robot workplace, e.g. pallets (or racks), worktables (or benches), conveyors, turntables, tool magazines, etc. Here the programmer is required to input a limited set of parameters so that a text source file can be generated automatically and be coded appropriately for input to the GRASP solid modeller. The software modules produced have a common design structure.

To illustrate the principles involved, consider a specific software module which was produced to assist the user in creating objects in the pallet or rack category. The parameter values, such as the pallet dimensions and name, are supplied in response to prompts, and are used to generate a textual GRASP source file. Documentation of the software module is shown in Appendix B.1. The GRASP pictorial representation of typical pallets is illustrated by figure 6-1.

The same approach, of generating workplace entities based on user responses to simple input prompts, was used in creating other commonplace workplace entities.

Software modules were also produced for tool magazines and worksurfaces as well as the kinematically functional turntable and conveyor based on the same principles. Figure 6-2 illustrates workplace models created in this way where appropriate geometric, spatial and functional relationships have been defined. Clearly, the time taken by a user in creating a model depends upon the complexity involved and his/her level of expertise. For the examples treated here, worksurface modelling is the simplest, this also being reflected in the level of complexity in the





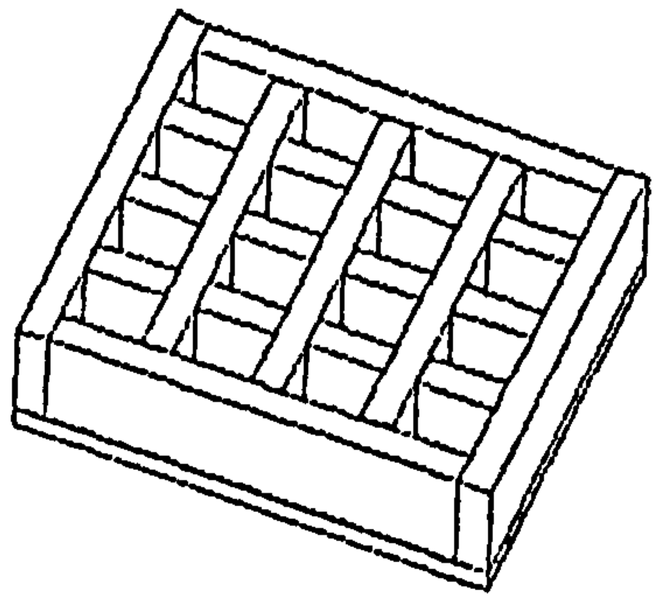


Figure 6–1 GRASP pictorial representation of typical pallets

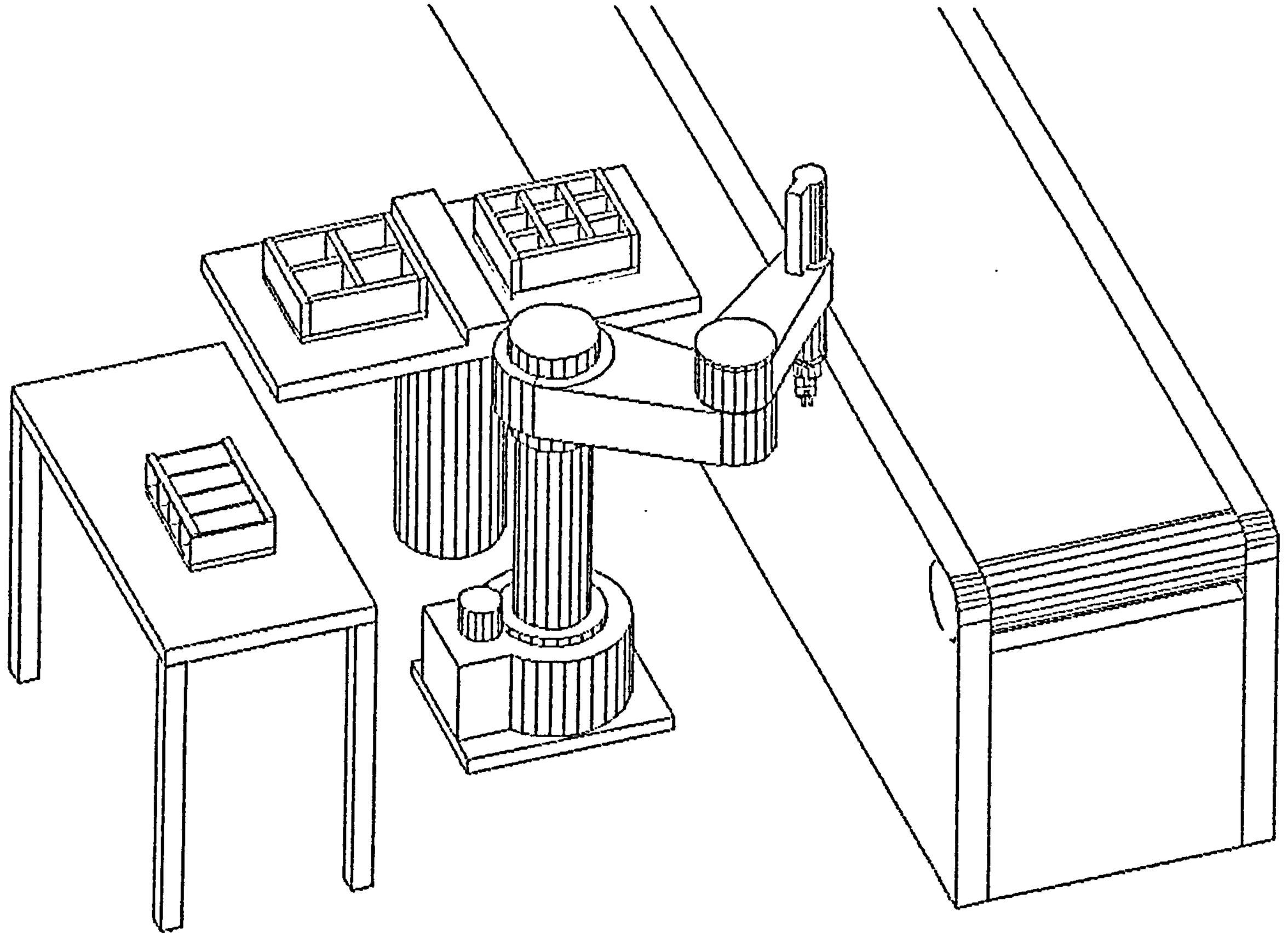


Figure 6–2 Example workplace configured using the model building facility

associated software module, whereas conveyor modelling is the most complex. When using the modelling facilities of a typical robot simulator a rough estimate of the time needed to create these solid models is between 10 and 30 minutes, the actual time taken being dependant on the experience of the user (although a significant part of the time required is usually spent on calculating the correct geometric location of the primitives). However, if the enhanced MMI software modules are used in generating an entity model, then the time required will be between 1 and 2 minutes. Thus significant time saving can be achieved. Perhaps of greater significance, however, is the confidence that can be placed in a parametrically defined model object. The parameterisation will have been subjected to rigorous testing of its suitability for the application and thus it is unlikely that model shortcomings will be discovered inconveniently late in the simulation process.

Using the system components created in this study, a limited range of items can be produced through the use of specific software modules, and this illustrates the general principles. However, it is recognised that eventual full-scale implementation is likely to be via a general purpose parametric language taking into account the special needs of this technique in robot modelling.

## 6.2 Computer Assisted Task Program Generation of Simulated Tasks

GRASP does have a language for the definition of a sequence of events within the simulation. This is very flexible, being accessed through screen menus or by a textual command language, and can be used to

explicitly define a wide range of tasks. It is however, still a complex task to create new sequences in a generalised way, and therefore software modules have also been produced which can simplify this activity. In particular, common material handling tasks have been studied and software modules produced for generating palletising, de-palletising and machine tending tasks which are representative of tasks for which robots are frequently used. Here the command inputs invoke commonly occuring sequences of motion when performing simulations. For example, it is not usual to pick up an object directly as collision between the manipulator's end-effector and other workplace entities may occur. Instead, the manipulator is normally commanded to move to a position above the object (if this is clear of obstacles), from which the gripper is moved in a controlled fashion onto the object, so as to be able to grip it and lift it up before the remainder of the task continues. Clearly there are many other "micro sequences" of this type, which when defined and parameterised, would make extremely useful building blocks for "macro sequences" or complete robot tasks.

## (a) Palletising

To demonstrate this principle a software module was produced to simplify the definition of a variety of palletising operations. This module comprising a number of functional elements as shown schematically in figure 6-3.

It is logical to create a GRASP model of a robotic device and its workplace before a task is defined. However, all workplace entities must have explicit names which are known to the user in order that the task

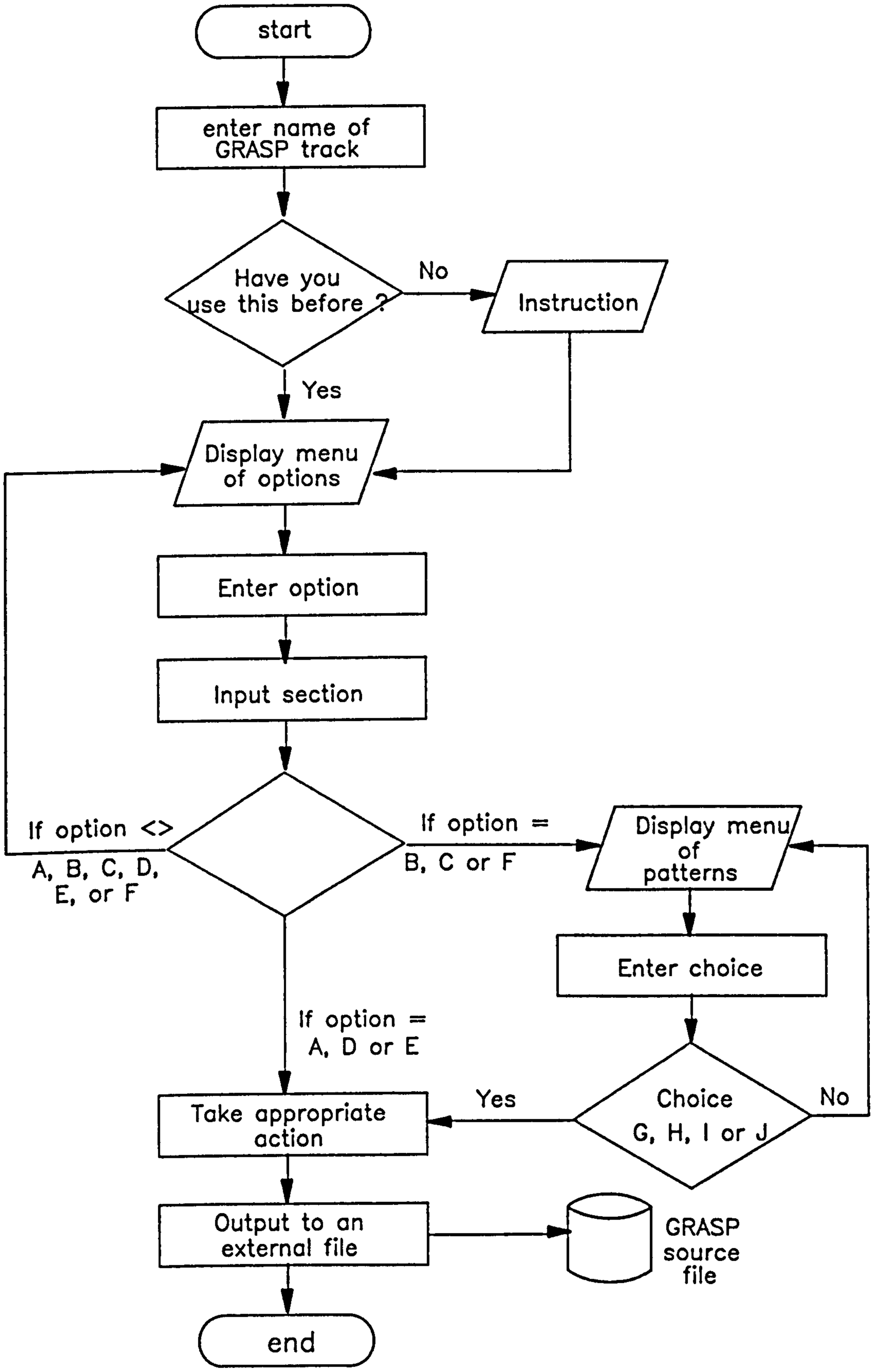



Figure 6–3 Flow chart of PROG_PALLET module
(module structure used to define palletising operations)

programming software can be used to create a GRASP track (i.e. a task program or motion sequences with reference to the various entities of the model).

GRASP users are provided with different programming levels (e.g. for simulation and off-line programming, for simulation only etc.) which in turn provide different options for generating palletising or depalletising patterns. The programmer is required to respond to terminal prompts with the user's responses, leading to the computer assisted generation of task programs.

A software module (PROG_PALLET) was produced to simplify the definition of a variety of palletising operations. An introduction for first time users, can be invoked. This section describes the assumptions made and the rules which must be followed (as listed below). Several assumptions have been made in automatically generating the patterns.

(i) Workpieces are assumed to be identical.

(ii) Workpieces are assumed to be palletised at the centre of each rack.

(iii) Sensory information is not incorporated within the task definition.

(iv) Reference points are assumed to be at the height of workpieces.

(v) Number of reference pick (and or place) points and the number of workpieces are to be equal.

A GRASP model which is built for programs generated by the software module is shown in figure 6-4. In simulating a task, it is necessary to define any reference points and these may take the form of an array. To

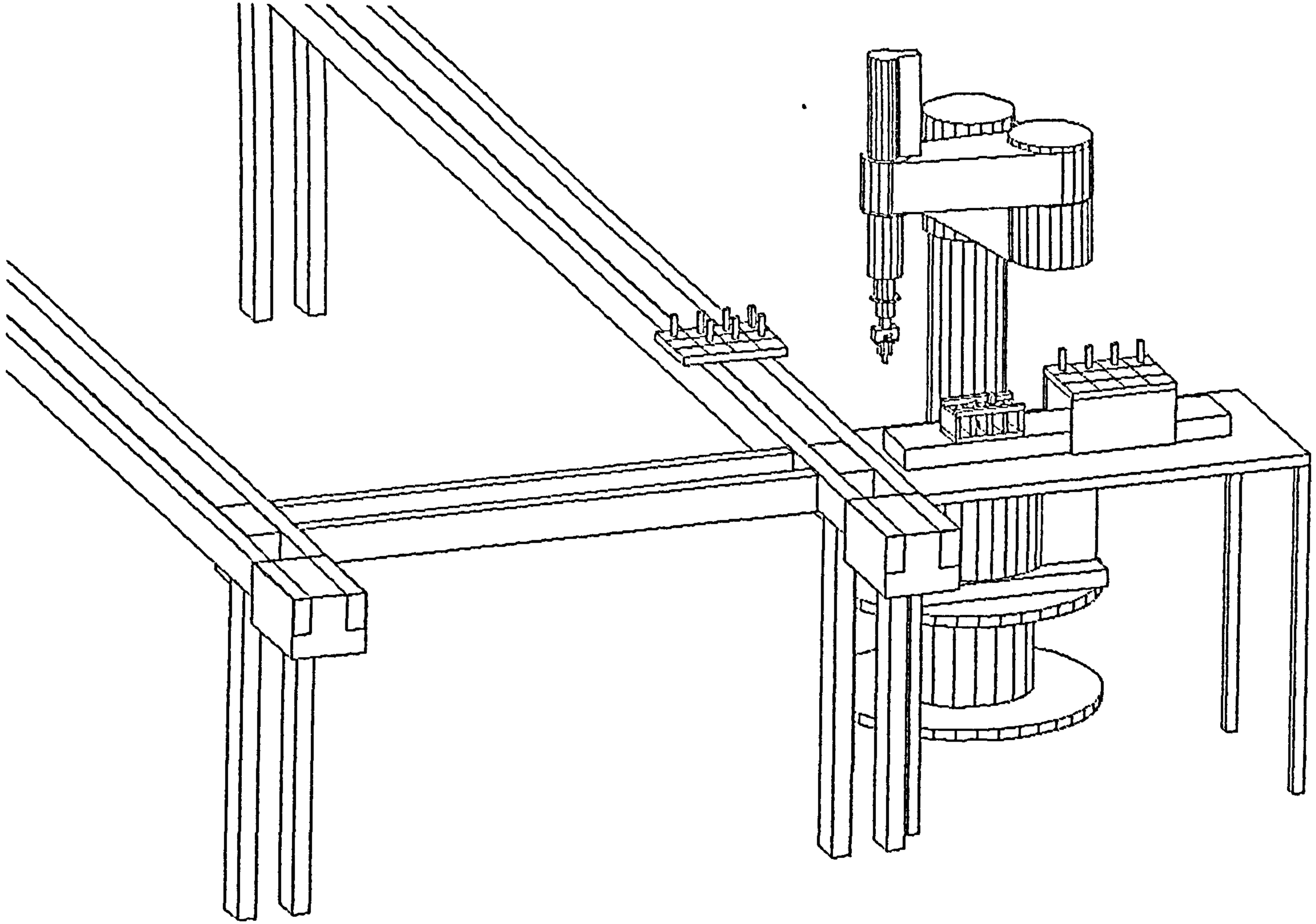


Figure 6–4 Example workplace configured for palletising operations

simplify reference point definition, optional software based procedures were written which accept the names of reference points. The resulting simulation task is presented in GRASP syntax, ready to be used in the simulation. To simplify/automate reference point definitions, a number of optional procedures can be invoked by the user specifically for palletising and de-palletising and these will find wider application in general assembly task programming. The available options are described below :

Option A is used where workpieces are to be picked up at fixed reference points assigned to a pallet and placed at fixed reference points assigned to another pallet. The sequence of these reference points describes the order of picking and placing, and thus a variety of robot movements are available. This option is suitable for both GRASP simulation and off-line program generation.

Option B is similar to option A except all the workpieces are to be placed at positions calculated from a fixed reference point on the destination pallet (the frame of the destination pallet) and a variety of palletising patterns are available (see later) at the destination pallet. This option produces a GRASP track in terms of real variables and it is to be used for GRASP simulation only, and is not suitable for off-line robot programming due to the restriction imposed by the proprietary post-processors (no real variables can be post-processed).

Option C is similar to option B, but, it is suitable for both simulation and off-line programming. No real variables are used for defining the location, but the program is effectively long winded in comparison with option B.

Option D is a de-palletising option in which all workpieces are to be picked up at fixed reference points on a pallet and are to be placed at a fixed reference point, for example on a conveyor or a bin that collects completed components. This option is also suitable for both simulation and off-line programming.

Option E is designed to be used for picking up workpieces at a fixed reference point (e.g. from a conveyor or from a feeding device) and to unload workpieces at fixed reference points on a pallet. Again this option is also suitable for both simulation and off-line programming.

In Option F, workpieces are to be picked up at a fixed reference point (e.g. a conveyor or feeding device) and placed at fixed reference points on a pallet. This option is also suitable for both simulation and off-line programming.

We are basically considering four different palletising or de-palletising patterns from a variety of choices. These patterns are to be used at the destination pallet and are the extension of the option B, C, and F. We consider the available patterns in terms of motions along the X and Y axes with reference to the pallet's own origin. The movement is incremental in both axes. The robot is commanded to move to the first row (Y axis) and then move along each column (X axis) until the final column of that row is reached. The robot then proceeds to the next row. This movement is repeated until the final position is reached.

For pattern choice G, the robot first visits Xmin, Ymin (with respect to the pallet origin) then moves to each point along the X axis. Subsequently the Y axis is incremented and the procedure repeated until pallet positions have been visited (see figure 6-5). This is similarly applied to choice H, I and J with one of the four corners as its starting point. Appendix B.2 shows an example output of result generated by this software module.

Some of the software modules were written so that they can be used in the high level GRASP programming mode where real variables are used to represent locations. Such an approach will yield a track that is simple but is only suitable for GRASP simulation and not off-line post-processing. In order to facilitate both simulation and post-processing, special modules were written.

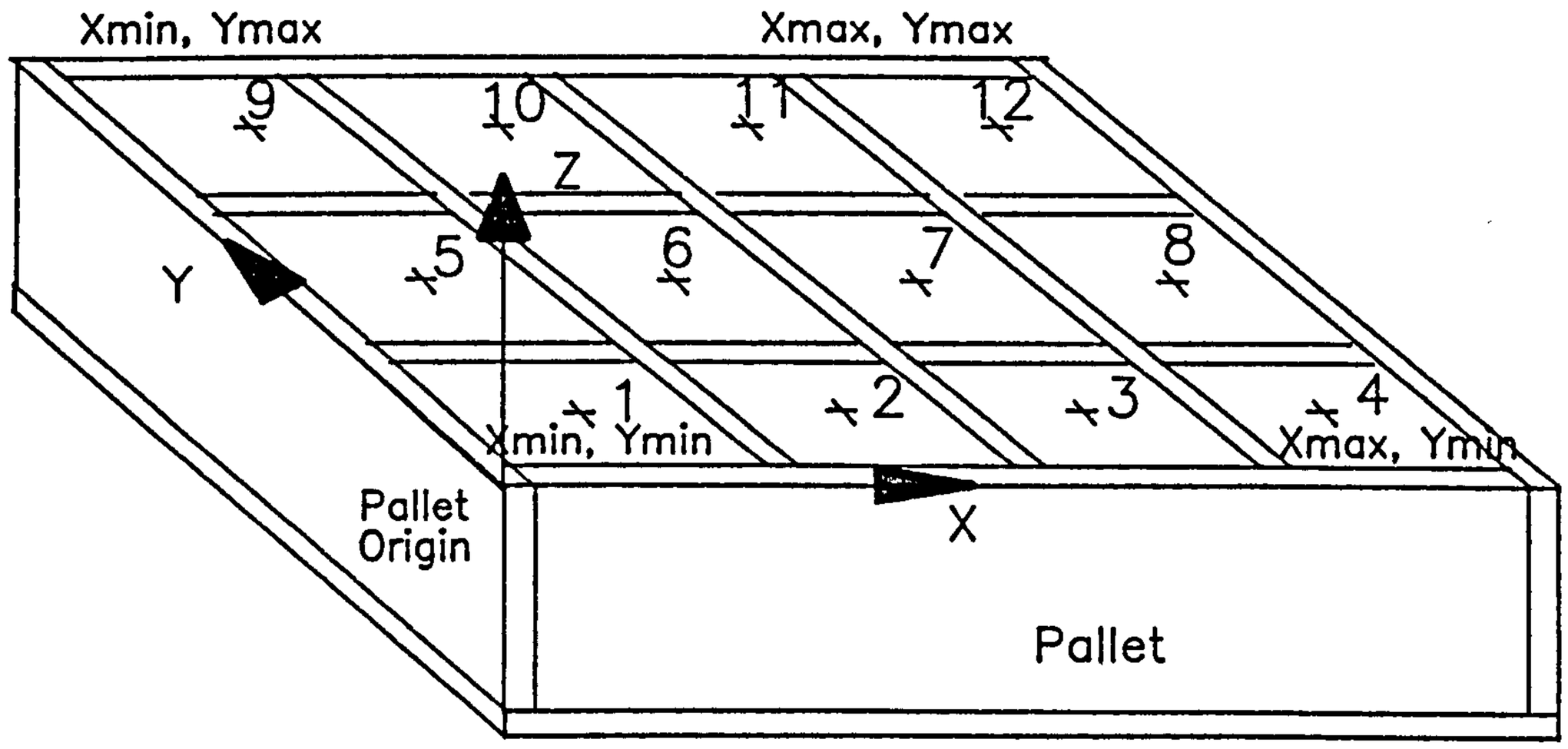
(b) Machine Tending

To investigate further opportunities for simplifying task definition, facilities for specifying machine tending tasks were included as such tasks are very frequently required. There are three basic types [Wilhelm and Sarin, 1985] of machine tending depending on the shopfloor configuration viz :

(i) Each part must be processed sequentially on all machines.

(ii) Each part requires one operation and could be assigned to any machine.

(iii) Each part requires one operation and must be assigned to a specific machine.




Palletising patterns are based on the four corner positions. Four different palletising patterns are implemented for illustration.

(i) Start from Xmin, Ymin to Xmax, Ymin and then repeat along Y axis towards Xmax, Ymax.

(ii) Start from Xmax, Ymin to Xmin, Ymin and then repeat along Y axis towards Xmin, Ymax.

(iii) Start from Xmin, Ymax to Xmax, Ymax and then repeat along Y axis towards Xmax, Ymin.

(iv) Start from Xmax, Ymax to Xmin, Ymax and then repeat along Y axis towards Xmin, Ymin.

Figure 6–5 Demonstration of palletising locations

Among these shopfloor configurations, programming type (i) is the simplest whilst type (iii) is the most complicated. Configuration type (i) and (ii) may look simple, but it is laborious to programme. To programme the shopfloor configuration type (iii), a vision system (or by other means such as bar-code reading) is required to identify the part and then assign the part to the appropriate machine. The loading, unloading and production cycle times for each machine have to be evaluated and used in the production scheduling. Since this research study is not concerned with production scheduling, scheduling problems for flexible manufacturing systems are not considered. Software modules were coded for illustrating the principles involved in enhancing the man-machine interface.

Machine tending software was developed to implement these three basic configurations. The software is generic, such that variations in the number of machines and parts, and the sequence in which these machines require tending, can be varied according to the priority specified in the input. In each case, programs for robot tending are generated together with shell programs for each attendant peripheral machines. These programs include simple sensory signal input/output for synchronising the robot tending workplace.

PROG_TEND is a software module written to assist programmers in defining such machine tending tasks. Appendix B.3 shows an example robot tending simulation program for configuration type (ii). When using this facility the user will require a GRASP model and must have knowledge of the model i.e. the number of machines that will need to be tended, the names of these machines, the order in which the machines require tending, the

number of workpieces to be manipulated, the names of workpieces, the required sequence of manipulations for each workpiece, the pick up reference point(s) and place reference point(s). The machines will share a common pick up reference point (e.g. a bin which contains raw material or a conveyor belt that drives material into the workcell) and a common place reference point (e.g. a bin which collects finished workpieces or a conveyor belt that drives workpieces out of the workcell). This arrangement is shown in figure 6-6. The machine tending software assumes that all workpieces are to be inserted in a clamping device or a jig with reference to an assigned target (named with the machine name suffixed by '_TARG'). The approaching movement is described in terms of the Z axis of each target. For instance, if any machine requires a horizontal feeding movement, for example, a lathe, the target will need to be rotated about its Y axis by 90 degrees (such that the Z axis of the target is horizontal) for the purpose of inserting workpieces horizontally. The design structure of this software is depicted in the flow chart of figure 6-7.

Subsequently during simulation, any joint violation and object collisions can be detected. The computer assisted model building and task operation facilities have been used in programming a number of tasks for two Adept One robots, the off-line program generation being accomplished through the use of a VAL II post-processor. These off-line programming exercises have proven the validity of the approach and suggested many possible enhancements.

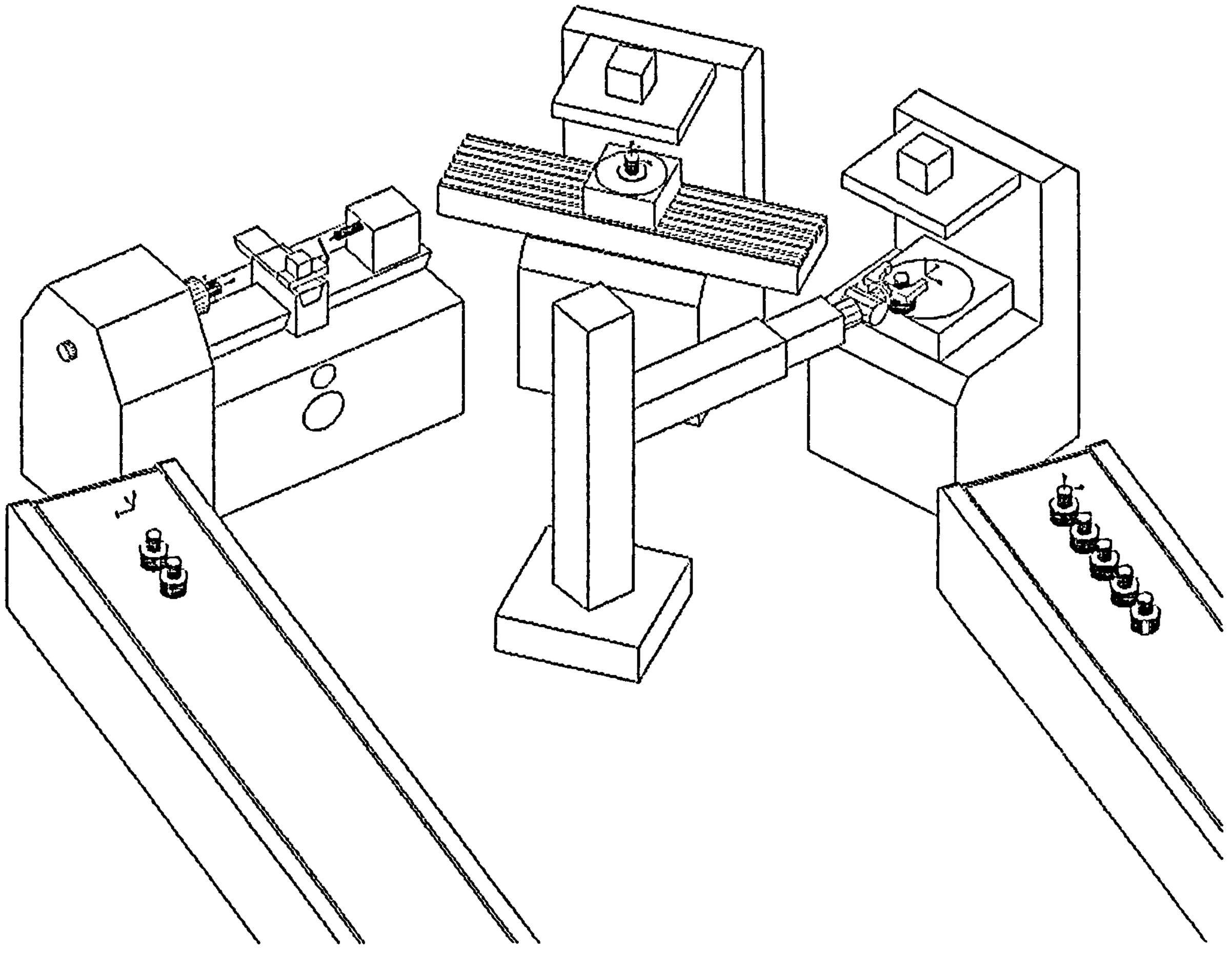


Figure 6–6 Example workplace configured for machine tending operations

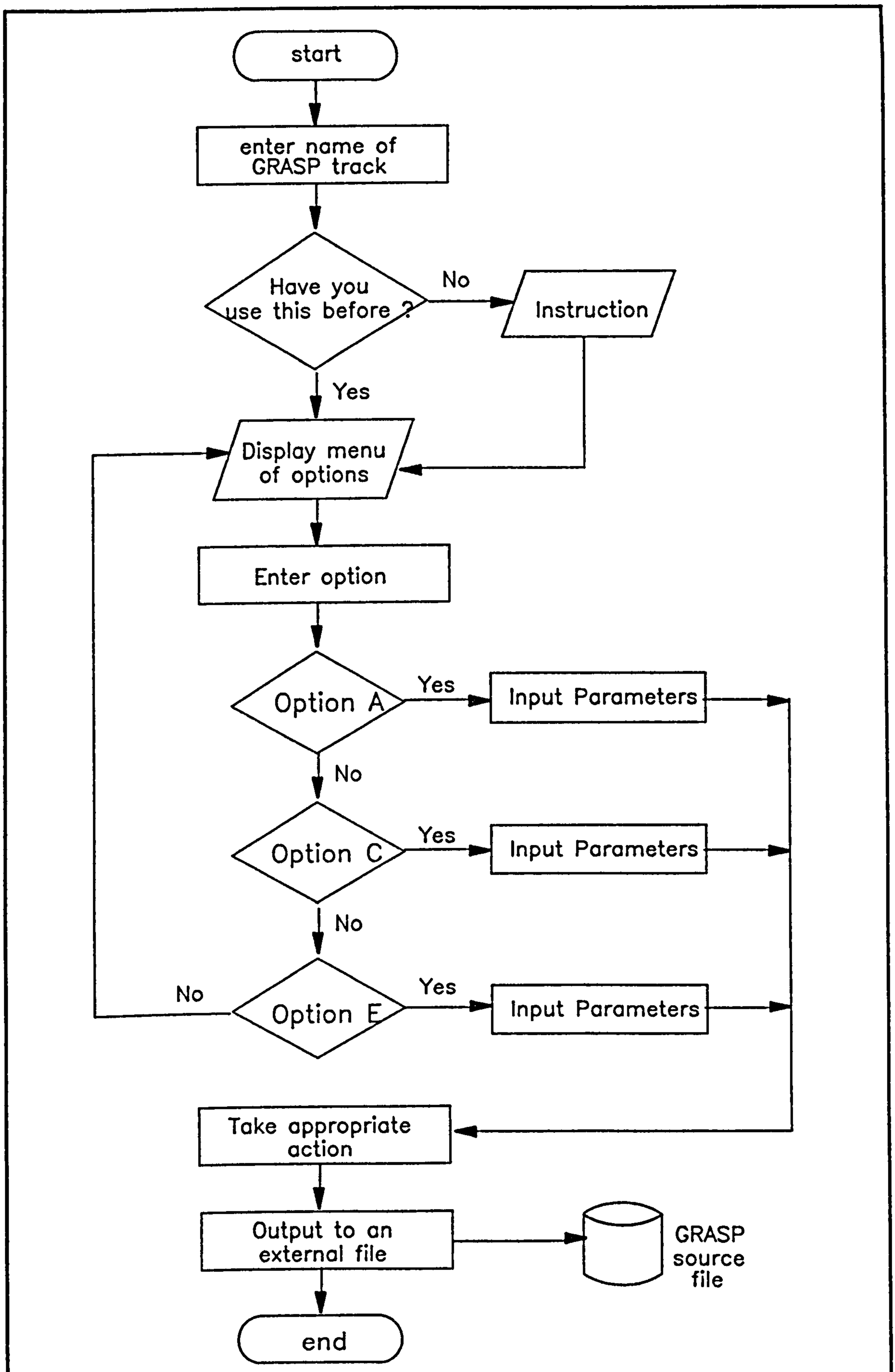



Figure 6–7 Flow chart of PROG_TEND module
(module structure used to define machine tending operations)

## 6.3 General Considerations

At the present time, most general robot simulation systems are too complex to learn and when programming a simple robot task it may be more efficient to use manual guiding methods. On the other hand, application-dependent systems, although inflexible, are very efficient for task programming of the specific application. In order to gain advantages from these systems, a parameterised task language should be made available such that different types of robot motion can be programmed without losing too much flexibility.

Most robot applications in manufacturing industry can be grouped into pick and place, palletising, machine tending, assembly (including mechanical and electronics assembly), and welding (such as arc and spot welding) operations. The robot motion required to carry out these operations can be broken into subgroups (see table 6-1) and each subgroup can be dealt with using a parameterised task language. In order to generalise this idea, an analysis of the robot task movements has been conducted to arrive at a common ground on which the parameterised language is to be based.

As a general recommendation, the implementation of this type of parameterised task language should be done according to the type of standard motion. The proposed language should comprise several application modules, each module being responsible for each category of task commonly encountered in manufacturing industry. Each category, in turn, is divided into individual functions for carrying a specific part of the task. The proposed generic task language is illustrated through the examples in sections (a) to (f).

| COMMON APPLICATIONS | | BASIC ROBOT MOVEMENTS |
|---|---|---|
| Pick and Place | | APPRO pick up position<br>ONTO pick up position<br>GRIP object<br>DEPART from pick up position<br>APPRO target position<br>ONTO target position<br>RELEASE object<br>DEPART from target position |
| Machine tending | | APPRO bin1 / ONTO object1 / GRIP object1 / DEPART from bin1 / APPRO target machine / ONTO target machine / RELEASE object1 / DEPART from machine / :<br>APPRO machine / ONTO object1 / GRIP object1 / DEPART from machine / APPRO bin2 / ONTO place position / RELEASE object1 / DEPART from bin2 / : |
| Palletising | | APPRO pick up position<br>ONTO pick up position<br>GRIP object1<br>DEPART from pick up position<br>APPRO target position<br>ONTO target position<br>RELEASE object1<br>DEPART from target position |
| Assembly | Mechanical | APPRO object1<br>ONTO object1<br>GRIP object1<br>DEPART from pick up point<br>APPRO object2 (assembly in fixture)<br>ONTO object2<br>RELEASE object1<br>DEPART from object2 |
| | Electronic | APPRO feeding device1<br>ONTO chip1<br>GRIP chip1<br>DEPART from feeding device1<br>APPRO target position on pcb<br>ONTO target position with a transformation<br>RELEASE chip1<br>DEPART from pcb |
| Welding | Arc welding | START applying current/voltage<br>APPRO initial position<br>ONTO initial position<br>MOVE along to 2nd position<br>:<br>DEPART from workpiece<br>STOP applying current/voltage |
| | Spot welding | START applying current/voltage<br>APPRO initial position<br>ONTO initial position<br>CLOSE jaws<br>RELEASE jaws<br>:<br>DEPART from workpiece<br>STOP applying current/voltage |

Table 6-1 Common robot applications and basic robot task movements

## (a) ARC WELDING COMMANDS

APPLY VOLTAGE/CURRENT <number/number>

WELD_STRAIGHT_LINE { NAME <name1> TO <name2> }
{ COORD <point1> TO <point2> }

name1, name2 { name given to global x, y, z coordinates and orientations }
point1, point2 { global x, y, z coordinates and orientations }

WELD_CIRCULAR { NAME <name1> <name2> }
{ COORD <point1> <point2> } RADIUS <number1> { ANGLE <number2> }

STOP VOLTAGE/CURRENT

name1, name2 { name given to global x, y, z coordinates and orientations }
point1, point2 { global x, y, z coordinates and orientations }
radius of arc or circle
number1 { +ve real number indicate clockwise }
{ -ve real number indicate counter-clockwise }
angle is angle of welding path
number2 { +ve real number indicate clockwise }
{ -ve real number indicate counter-clockwise }

## (b) SPOT WELDING COMMANDS

SPOTWELD { NAME <name1> <name2> ... }
{ COORD <point1> <point2> ... }

SPOTWELDS { NAME <name1> TO <name2> } { PITCH }
{ COORD <point1> TO <point2> } { INTERVAL } <number1>

SPOTWELDC { NAME <name1> TO <name2> } { PITCH }
{ COORD <point1> TO <point2> } { INTERVAL } <number1> RADIUS <number2> { ANGLE <number3> }

name1, name2 { name given to global x,y,z coordinates and orientations }
point1, point2 { global x,y,z coordinates and orientations }
straight line pitch number1 always +ve towards 2nd point
circular pitch number1 { +ve indicate clockwise towards 2nd point }
{ -ve indicate counter-clockwise towards 2nd point }
interval is the number of points between the start and finish spot welds
radius of arc or circle
number2 { +ve real number indicate clockwise }
{ -ve real number indicate counter-clockwise }
angle is angle of welding path
number3 { +ve real number indicate clockwise }
{ -ve real number indicate counter-clockwise }

## (c) TRANSPORT COMMANDS

SEPARATE <object1> <object2> ... STREAM <path1> <path2> ...

MERGE <object1> <object2> ... STREAM <main path1> <path2> ...

TURN <object> BY <angle>

ORIENTATING <object1> BY <angle> { RELATIVE TO <object2> }

ALLOCATING <number> <object> AT <path1>

object1, object2 are names given to objects
path1, path2 are the path of movement
number is the number of objects to be allocated to a path

## (d) ASSEMBLY COMMANDS

MOVES { TO } { <point> }
       { <object> }

MOVEC BY { X } <number>
         { Y }
         { Z }

APPROS { NAME <object> } BY { X } <number1>
       { COORD <point1> }   { Y }
                            { Z }

APPROC { NAME <object> } BY { X } <number1> RADIUS <number2> { ANGLE <number3> }
       { COORD <point1> }   { Y }
                            { Z }

DEPARTS { NAME <object> } BY { X } <number1>
        { COORD <point1> }   { Y }
                             { Z }

DEPARTC { NAME <object> } BY { X } <number1> RADIUS <number2> { ANGLE <number3> }
        { COORD <point1> }   { Y }
                             { Z }

PICK AT { NAME <object> } BY { X } <appro dist> <depart dist>
        { COORD <point> }    { Y }
                             { X }

PLACE AT { NAME <object> } { COORD <point> } BY { X } { Y } { X } <appro dist> <depart dist>

ALIGN <object1> TO <object2>

INSERT <object1> in <object2> { BY <depth> } { TO END } TURN <object1> ABOUT { X } { Y } { Z } BY <angle>

SCREW <object1> ONTO <object2> BY <angle>

RIVET { NAME <name1> <name2> ... } { COORD <point1> <point2> ... }

RIVETS { NAME <name1> TO <name2> } { COORD <point1> TO <point2> } { PITCH } { INTERVAL } <number1>

RIVETC { NAME <name1> TO <name2> } { COORD <point1> TO <point2> } { PITCH } { INTERVAL } <number1> RADIUS <number2> { ANGLE <number3> }

EXTRACT <object1> FROM <object2>

ASSEMBLE <object1> ONTO <object2>

DISASSEMBLE <object1> FROM <object2>

## (e) PALLETISING COMMANDS

PALLETPTP { FIX <start frame> <option> } { FLEX <name1> <name2> ... } TO { FIX <finish frame> <option> } { FLEX <name1> <name2> ... }

PALLETPTP is a command used to generate lower level commands for the specific use of palletising objects from one pallet to another

<start frame> is the name given to the coordinate frame of the starting pallet

<finish frame> is the name given to the coordinate frame of the finishing pallet

{ PATTERN } is an optional command, once chosen the appropriate option should be entered

PALLETSPTP <start frame> TO { FIX <finish frame> <option> } { FLEX <name1> <name2> ... }

PALLETSPTP is a command used to generate lower level commands for the specific use of palletising objects from one single position to a pallet.

<start frame> is the name given to the coordinate frame of the starting position

<finish frame> is the name given to the coordinate frame of the finsihing pallet

PALLETPTSP { FIX <start frame> <option> } TO <finish frame>
{ FLEX <name1> <name2> ... }

PALLETPTSP is a command used to generate lower level commands for the specific use of palletising objects from a pallet to a single position.

<start frame> is the name given to the coordinate frame of the starting pallet

<finish frame> is the name given to the coordinate frame of the finishing position (e.g. conveyor, bin etc).

## (f) TENDING COMMANDS

LOADING <object> <MC1>

UNLOADING <object> <MC1>

OPEN_FIXTURE <fixture> AND WAIT <time in sec>

CLOSE_FIXTURE <fixture> AND WAIT <time in sec>

TENDMC <MC1> <MC2> ...

TENDMC is a command used to generate lower level commands for the machine tending operation where machines are identical.
<MC1> <MC2> ... represent machines' names

TENDMCS <MC1> <object type> <MC2> <object type> ...

TENDMCS is a command used to generate lower level commands for the machine tending operation where machines are not identical and object of certain type can be assigned to a specific machine.
<MC1> <MC2> ... represent machines' names
<object type> an integer represent object type

TENDFS <MC1> <MC2> <MC3> ...

TENDFS is a command used to generate lower level commands for the machine tending operation where machines are constructed to perform flow production with the machines specified in order of tending. <MC1> <MC2> ... represent machines' names

## 6.4 Limitations of Computer Assisted Solid Modelling and Task Program Generation

When defining objects, GRASP users must assign a unique name to that object. If two or more of these objects or entities (created by using software modules) are to be used in a GRASP simulation, there may be name clashes in the dictionary which causes the simulation to fail. The problem that has been identified here is that models and tasks cannot really be defined in isolation from the larger model and set of tasks that they become part of. One of the consequences is this problem with names. The proposed solution must be further pre-processing on input to GRASP or alternatively the use of contextual (or tree) names. Such operational procedures are also necessary with other simulation packages and robot simulators.

The major limitation of using the software modules to generate object models is that the flexibility of the solid modelling function of a robot simulation package is reduced. However, it is only to a limited extent since a certain part of an entity is often of particular importance with respect to off-line robot programming. Another generic problem with parameterisation is the infinite number of possibilities. Arguably, it

is not practical to provide an enhanced MMI facility for all possible object classes, however, the user can resort to using the usual GRASP syntax to define an entity which does not fall into the MMI classes implemented.

Without manipulator joint violation checks and collision detection, the task programs generated by computer assisted task generation software module may include errors of this type. Thus users are advised to check their generated programs through GRASP simulation, this being a normal procedure for checking joint violations and collisions. Pre-processors should not attempt to duplicate facilities which are more properly part of the main system.

Once this type of task language is fully developed, programmers do not need a graphical simulation, i.e. it would be good enough for post-processing as an off-line program. But realistically the simulation is there precisely for the kind of reasons identified in the above paragraph. Therefore, these parameterised task language should be included in the general purpose robot simulation system as built-in macro sequences.

Although off-line task programs are generated together with all the location information, the accuracy of these locations represent an idealised situation and generally not good enough for the real robotic arrangement. Furthermore, robot repeatability and accuracy may not be adequate for precise operations. The accuracy problems are dependent upon many factors and therefore off-line generated robot programs require calibration - a topic discussed in the next chapter.

# CHAPTER SEVEN

# OFF-LINE ROBOT PROGRAM CALIBRATION

## 7.0 Introduction

In common with any simulation process, the usefulness of a robot simulation is substantially governed by the available "accuracy" of modelling.

The present generation of computer systems provide sufficient processing power to achieve very high precision modelling of any manipulator system, albeit that "realtime" simulation may not be possible. However, in creating such a model, specific input data concerning the manipulator's kinematic and dynamic behaviour must be available from some source. When simulating robots two obvious sources of this input data are:

(a) from the robot manufacturer, and

(b) through using manipulator and workplace measurement devices.

If the robot simulator is used only in choosing a robot and designing a suitable workplace layout then the first of these sources can be appropriate. This approach is the one favoured by current suppliers of robot simulators. However, robot manufacturers usually supply only limited statistically averaged kinematic data concerning their manipulator with often little or no data relating to its dynamic behaviour. Thus when compared with simulated results, a specific industrial robot will demonstrate both dimensional variations (resulting from manufacturing tolerances, backlash, deflection in manipulator links, control system resolution and deadband, etc) and unpredicted dynamic behaviour (viz: damping, following error, etc) as it is moved through its working envelope.

Example data sheets of robot kinematic characteristics are shown in appendix C.1. Clearly, any significant inaccuracies or omissions in specifying the robot and workplace model can lead to lack of confidence in the simulation results.

This situation is further exacerbated when attempting to use the simulator to achieve the off-line generation of robot task programs. Here it is not sufficient for workplace elements to assume nominal positions, shapes and, where appropriate, dynamic behaviour. Nor can significantly large dimensional and dynamic behavioural errors in the manipulator simulation be tolerated.

To date very little detailed technical information is published in the literature which documents the use of robot simulators in off-line programming applications. However, two approaches have been used in attempting to overcome modelling errors, (a) through two stage programming, the second involving the use of a teach pendant [BYG, 1988], and (b) through the use of workplace sensor(s) to calibrate the robot and it's workplace [El-Zorkany, 1984; Paul, 1983; Tarvin, 1980].

The use of a teach pendant to overcome modelling errors must be viewed as a retrograde step, unless the pendant is used only to establish a limited number of reference or datum points, whereas the use of workplace sensors will incur a significant increase in complexity and additional cost. In fact generic robot calibration procedures have yet to evolve in an internationally accepted sense, resulting from the complexity involved in accurately measuring the position and orientation of workplace elements, let

alone their dynamic characteristics [ISO/DP8373, 1986; ISO/DP9283, 1986]. However, the McDonnell Douglas Robotics Software suite includes an "ADJUST" module as an example of a specific solution to this problem through the use of a calibration probe, mounted on the face plate of the robot's gripper (the probe supplying feedback/input data to the robot/workplace model).

Another common approach is the use of a trained vision system for identifying objects. The vision system will compare the objects in its view with the object models stored. When the vision system recognise an object, it calculates its orientation and the location of its centroid. This approach will be discussed in more detail later in this chapter.

## 7.1 Sources Of Error in Off-line Robot Programming Systems

In practice, the robot does not go to the commanded location as predicted by the model or conversely the entities comprising the workplace are not precisely at the locations as defined in the CAD model. These discrepancies can be attributed to the following [Jeyachandra et al 1986, Yong et al 1985, Lau and Hocken 1984].

### (a) Geometric/Static Errors

(i) Numeric Accuracy of an Off-line Programming System

The predicted location can be seriously affected by the numeric accuracy of the off-line programming system which is influenced by the algorithms used. In this way, for example, a small discrepancy in angle can lead to

a significant change in linear distance at a distance from the reference frame.

(ii) Accuracy of an Off-line Robot Program

The accuracy of an off-line program can at best be as good as the simulation model.

(iii) Robot Parameters

Insufficiently tight tolerances used in the manufacture of robot linkages can give rise to variations in joint offsets. Small errors in the structure can compound to produce quite significant errors at the robot end effector.

(iv) Part Defects

The same off-line generated robot program can function perfectly in most cases, but may occasionally fail resulting from the use of defective parts.

(v) Part and Tool Misalignment

Part misalignments can cause problems requiring static calibration. However, tool misalignment is a serious problem which can not simply be calibrated. For example in an off-line programming system, the tooling or gripper is perfectly aligned with the robot end effector centre line.

The main problem is the actual tooling or gripper mounted on the robot end-effector. This can be very different each time a new tool is mounted.

(vi) Difficulty in Determining Object Locations

The difficulty in determining precise locations of objects with reference to a datum within the workplace. The robot usually encompasses a large working envelope (i.e. variation in robot accuracy) but small tolerances on components require very good resolution.

## (b) Kinematic Errors

(i) Nominal Robot Parameters

The nominal geometry of a robot model (data obtained from design drawings) is normally used to derive the nominal robot kinematic functions (forward and inverse kinematic transformations). These nominal kinematic functions are often used in the robot controller for robots of the same model without taking into account variations in the assembled robots [Azadivar, 1987].

(ii) Insufficient Feedback Information

Insufficient feedback information (e.g. position, velocity, acceleration and torque) processed by the robot controller. This is seriously affected by resolution which is defined as the smallest measurement

increment achievable by the feedback sensor(s), e.g. encoder, resolver, tachogenerator, force sensor [Birk, 1976; Ho, 1982; Benhabib et al, 1987].

(iii) Joint Encoders Misalignment

Misalignment of joint encoders can cause serious problems, as eccentric movement of joints can produce unpredictable results.

(iv) Numeric accuracy of the controller

This is affected by the effects of quantisation, roundoff, sampling rate and other characteristics of the realtime control algorithms used i.e. not overcoming effects such as limit cycling, deadband, changing robot characteristics.

## (c) Dynamic Errors

(i) Incompatibility between robots

No two robots of identical configuration and model will behave exactly the same, and hence an off-line program for a robot with reasonably good accuracy may perform differently with another robot.

(ii) Lack of rigidity of robot structure

Lack of rigidity of robot structure (caused by the clearance in bearings,

bent and twisted shaft under different loading conditions may change the robot arm parameters). This can cause serious errors under heavy loading conditions and/or at high speeds.

(iii) Numeric accuracy of the robot controller

This is the same as described in (b)(iv).

(iv) Steady State Error of Servomotors

Steady state errors of servomotors caused by factors such as hysteresis, backlash in transmissions, and drift.

<u>(d) Transient errors</u>

(i) Stabilization

Robots with hydraulic drives do not stablize until after a certain amount of warm-up time [Warnecke et al, 1982].

(ii) Environmental Effects

Environmental effects such as temperature can adversely affect the performance of the robot e.g. heat generated locally in an arc welding operation. The effects of severe temperature changes may require the programmed path to be reprogrammed. In the example given by Nissley (1983), just the thermal expansion of the floor between the robot and

workpiece positioner amounts to about 0.18mm over a distance of 1.5 m due to a 10 degree C temperature change.

The compounding effects of these errors across the whole off-line programming system can lead to a significant magnitude of errors particularly where dynamic characteristics are to be modelled. For off-line programming to become a practical tool it must at least be possible to accomplish final positioning adjustments automatically for a limited set of application scenarios. To achieve this a combination of approaches will be required:

(i) the positional accuracy of the robot or a knowledge of inaccuracies has to be improved.

(ii) more reliable methods for determining locations of objects within a workplace need to be applied.

(iii) relaxations of the tolerances of the components on which the robot has to work to improve the overall performance of the system.

(iv) the incorporation of sensory technology should cater for the remaining discrepancies within a system but their inclusion will inevitably contribute penalties of cost and processing time.

## 7.2 Methods Of Calibration

Errors in off-line robot programming can be overcome by many methods, each of which has its own value dependent upon the application. In general, calibration can be categorised into three main groups viz: Simulation Model Calibration, Robot Calibration, and On-line Calibration.

### 7.2.1 Simulation Model Calibration

Calibration is required to cope with static discrepancies (i.e. geometric discrepancies) between the idealised CAD model and the real working environment. The simulation model calibration can be subdivided into two sections, firstly, to measure correct location data, and secondly to update the CAD workplace model.

#### (a) Measurement or Data Collection

##### (i) On-line Editing

This can be regarded as the simplest approach. The off-line programming system is used to generate the robot's sequence of motion and location data. The off-line robot program is verified by running it to check each programmed location. If any alterations are required, the robot is driven to the correct position by use of the teach pendant and the corrected locations are stored. This method is applicable where a robot is being transferred from one workplace to another. The main purpose of off-line programming in this context is to generate the correct sequence of motions and logic with the correct

locations data taught by an on-line method. This method still satisfies the requirement to improve robot downtime. Clearly however, the 'percentage improvement' will be related to characteristics of the application and may not alone justify the increased sophistication introduced through utilising off-line programming.

(ii) Datum Point Calibration

In this approach, the robot is moved to a number of important reference points which are required to be as accurate as possible. As for on-line editing, the robot can be used in defining these locations with the locations recorded and used to refine the original CAD model. Off-line programs generated by this approach should not require any further on-line programming.

This method is particularly useful for flexible manufacturing and assembly systems with a limited number of reference points. Palletising is a good example (figure 7-1) as the palletising positions are referenced to a single reference point. The teaching of this datum and a limited number of points, to define the orientation of the reference frame should be sufficient to ensure that the whole program will perform correctly as predicted.

(iii) Sensor Assisted Static Calibration

This involves a simple calibration to obtain the locations (relative to the robot base) of the entities in the workplace by the use of

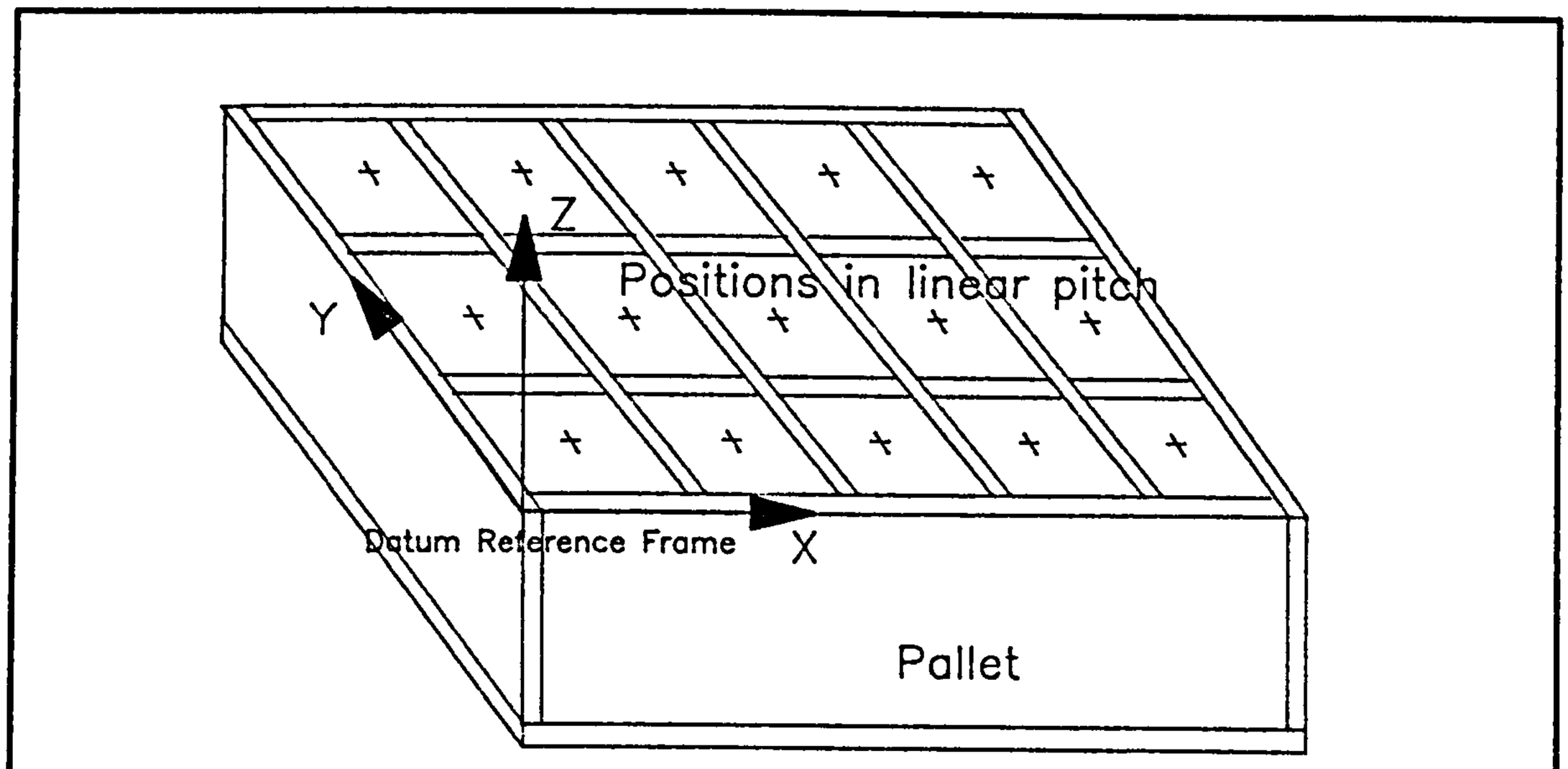



Figure 7–1 Datum point calibration used in palletising

sensors without manual intervention. The location data is then uploaded to the CAD system and the original model updated using a calibration program. This has the advantage of producing a model which more closely represents the real workplace. This calibration method is accurate enough for other classes of application which are not properly dealt with using previous two approaches. Most of the tasks programmed off-line (using this updated model) will be accurate enough and there will be no need for inaccuracy correction (except in the case of dynamic effects).

Spatial relationships between objects can be obtained through different methods ranging from the simplest to the most sophisticated. In one method, the robot is used as a digitiser to obtain the location and orientation of objects. However, with this method special tooling is required in order to obtain good accuracy. There are various ways

of achieving this calibration, and only two common groups of tooling are considered here:

- probing/force sensing
- vision systems

First, we consider the simple form of probing. A pin with a square top is made which fits in the gripper to provide adequate squareness and alignment with the centre of the robot mounting flange. A force sensor installed at the robot wrist ensures that when the pin touches the object in calibration, the charge amplifier provides indications such that the x, y and z dimension can be monitored. This method remains simple whilst giving reasonably accurate readings. An even better approach is to use a probe that can provide indications as it attains the location required. However, these approaches can provide calibration in 2D only.

A more sophisticated method might involve the use of a vision system whereby the location information of certain objects can be obtained. Robotic vision can broadly be classified into 2D and 3D.

2D robotic vision is generally based on visible band lumination data and mainly deals with analysis, recognition and interpretation. In some cases, the shape of the object is not important, and only the features of the marks placed on these objects are desired for their unique identification. These marks can be handwritten, printed letters or geometric marks (i.e. fiducial marks). Also in some

applications, the locations of these marks are not desired, but the interpretation of the content of the mark may yield part identity. For example, a camera can be mounted on the robot arm at a fixed spatial relationship to the robot arm. The spatial relationship of the entities with respect to the robot base would be calculated through the compound transformation. This method is far more accurate than the previous method. However, the vision system has to be trained to recognise objects or entities in the workplace before it can pick up the location information of the entity concerned. This can be very time consuming unless the vision system can be programmed through integration with product design (i.e. access to product information contained in manufacturing or design databases).

3D robotic vision is of growing importance in many applications. A major requirement of any robotic vision system is "robustness" which when properly enhanced will enable wider application of the technique. An important operational requirement for robotic vision is that it should be capable of dealing with incomplete image or data due to glare or shadow, partial occlusion by other objects or even the robot arm itself. A desired attribute of a general robotic vision system would be the ability to develop at least some partial information about the situation such as a "hypothesis verification" so as to identify the part type, location and orientation. This hypothesis verification should be offered along with a measure of the system's confidence regarding the verification. Positioning accuracy is typically on order of 1 to 2 mm [Rueb and Wong, 1988].

Error in vision tasks can translate into errors in physical manipulations. Manipulators, parts, and fixtures could be damaged or destroyed. High "value added" assemblies might need to be scrapped, a costly result of a sensory error.

## (b) Static Error Correction Through Updating CAD Model

Currently, most of the available off-line robot programming systems are open loop systems. Thus no feed back information is available to the off-line robot programming system to refine the simulation model. If any discrepancy is found between the real system and on idealised simulation model, correction can only be made on-line without updating the simulation model. If the off-line robot programming system is used for simulating manufacturing or assembly operations of families of parts or products, the inability of the system to update its simulation model may cause high inefficiency, especially in a CIM context. Due to this obvious reason, calibration and updating software modules were under consideration as one the necessary means. Software modules were coded in PASCAL to deal with calibrating and updating the simulation models, and thus a closed loop off-line programming system is attained. Two separate approaches have been used in this enhancement: single level approach and hierarchical approach.

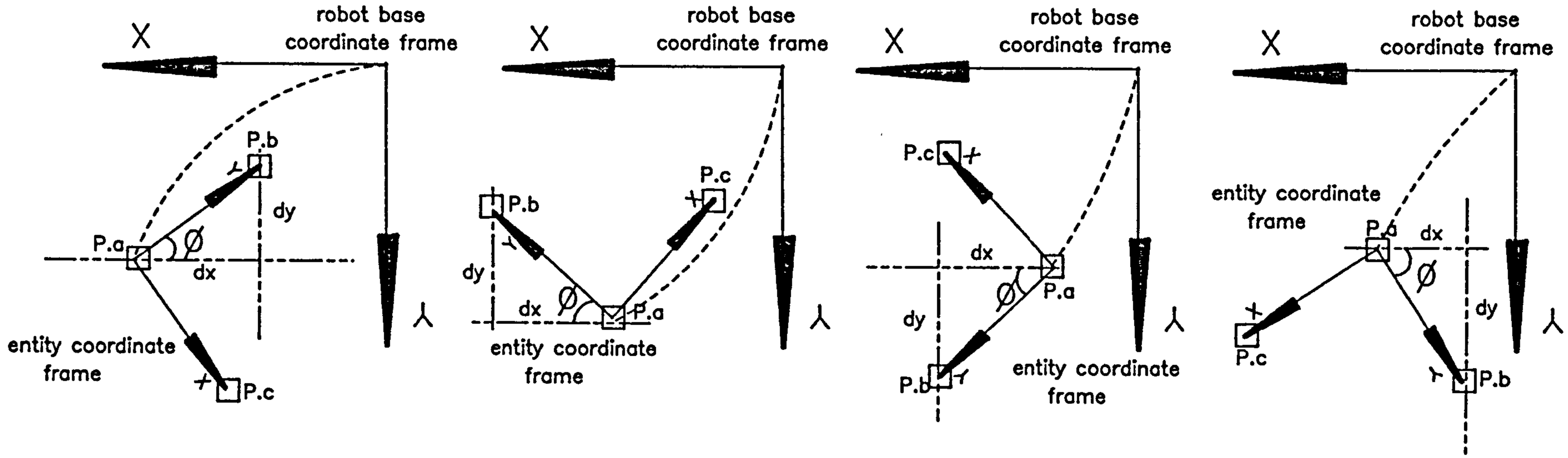
### (i) Hierarchical Approach

The hierarchical approach is designed for calibrating and updating a specific robot workcell with a four axis Adept One robot. The

hierarchical approach involves two separate software modules, namely CALIB and UPDATE. The former reads in the location data file. The location file contains the names of the locations and their positions and orientations. Each location can be identified by one touch point if the object is so small (normally used for small workpieces), particularly in the case of an entity that is symmetrical i.e. the orientations of the entity does not create any orientation problem. The orientation of the robot gripper is used as the orientation of the workpiece. The robot controller does not accept two locations of the same name, and hence to keep more than one touch point for the same entity at the same time, the name given to the locations must be the same as used in CAD model but each with a suffix ".a", ".b", ".c" etc. The calculation of the orientation of the entity frame is illustrated in figure 7-2. Alternatively, the orientation of an entity can be obtained through the FRAME function which is specific to VAL II or could be done with equivalent functions that are available in other robot languages. This function requires three positions to identify the orientations of the entity relative to the robot.

The CALIB module reads in the location data file and subsequently identifies each unique object name with its corresponding location data. The location information stored in this file represents spatial relationships between entities and the robot base. Since the CAD model stores information in a hierarchical order, the CALIB module searches through the original CAD model database to obtain the name of the owner object and its relationship with respect to the robot base. It calculates the new spatial relationships between entities and their owners at a level above in the hierarchical structure. The results

N.B. Positions P.a, P.b and P.c are calibrated positions, and each position contains x, y and z coordinates. P.a is the origin of the Frame.
The calculation of orientation of frame relative to robot base is illustrated.




| if X2 < X1 and Y2 < Y1 then | if X2 > X1 and Y2 < Y1 then | if X2 > X1 and Y2 > Y1 | if X2 < X1 Y2 > Y1 |
|---|---|---|---|
| $\emptyset = \tan2^{-1}\left(\frac{dy}{dx}\right)$ | $\emptyset = \tan2^{-1}\left(\frac{dy}{dx}\right)$ | $\emptyset = \tan2^{-1}\left(\frac{dy}{dx}\right)$ | $\emptyset = \tan2^{-1}\left(\frac{dy}{dx}\right)$ |
| Orientation with respect to robot base coordinate frame | Orientation with respect to robot base coordinate frame | Orientation with respect to robot base coordinate frame | Orientation with respect to robot base coordinate frame |
| $\emptyset = 90 + \lvert\emptyset\rvert$ | $\emptyset = -90 - \lvert\emptyset\rvert$ | $\emptyset = -90 + \lvert\emptyset\rvert$ | $\emptyset = 90 - \lvert\emptyset\rvert$ |

Figure 7–2 Illustration and explanation of finding entity frame

are stored in a new location file ready for updating the original simulation model. The software design is described in figure 7-3.

The UPDATE module reads in the updated location and orientation information and searches through the original GRASP model in order to update the entity's location and orientation. A new GRASP model with new entities is produced. Figure 7-4 illustrates the functionality involved.

## (ii) Single Level Approach

The single level approach involves a PASCAL module called VALTOGRASP. This module reads in the locations and orientations of entities that have been calibrated, and the appropriate syntax for GRASP is produced (with transformations between the entities concerned and the robot). GRASP is automatically invoked, and the original GRASP simulation model loaded. Finally the generated GRASP syntax inputs are fed into the simulator. The simulator accepts the GRASP transformations which are used to recalculate the spatial relationships between the robot and the entities within the workplace model. Figure 7-5 is a functional diagram representation of the logic design of the software module. Appendix C.2 shows an example output from this module. The newly created model should closely represent the real environment and thus the inaccuracies are reduced.

There is another possible approach which is potentially a more interactive method of updating a CAD model, but it is possible only if the robot simulation software has the ability to read in calibrated data directly from the robot controller via an interfacing unit.

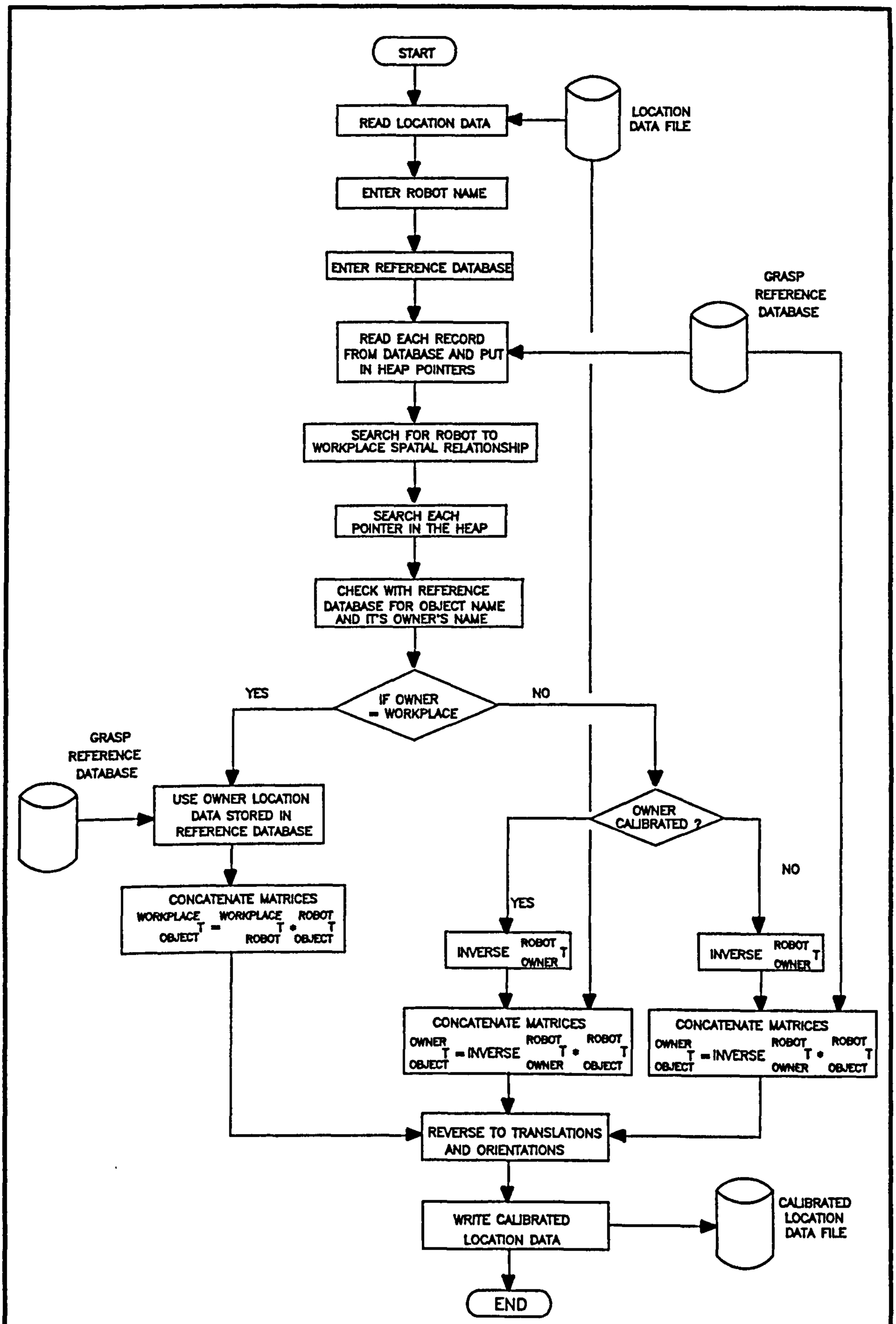



Figure 7–3 Flow chart of CALIB module

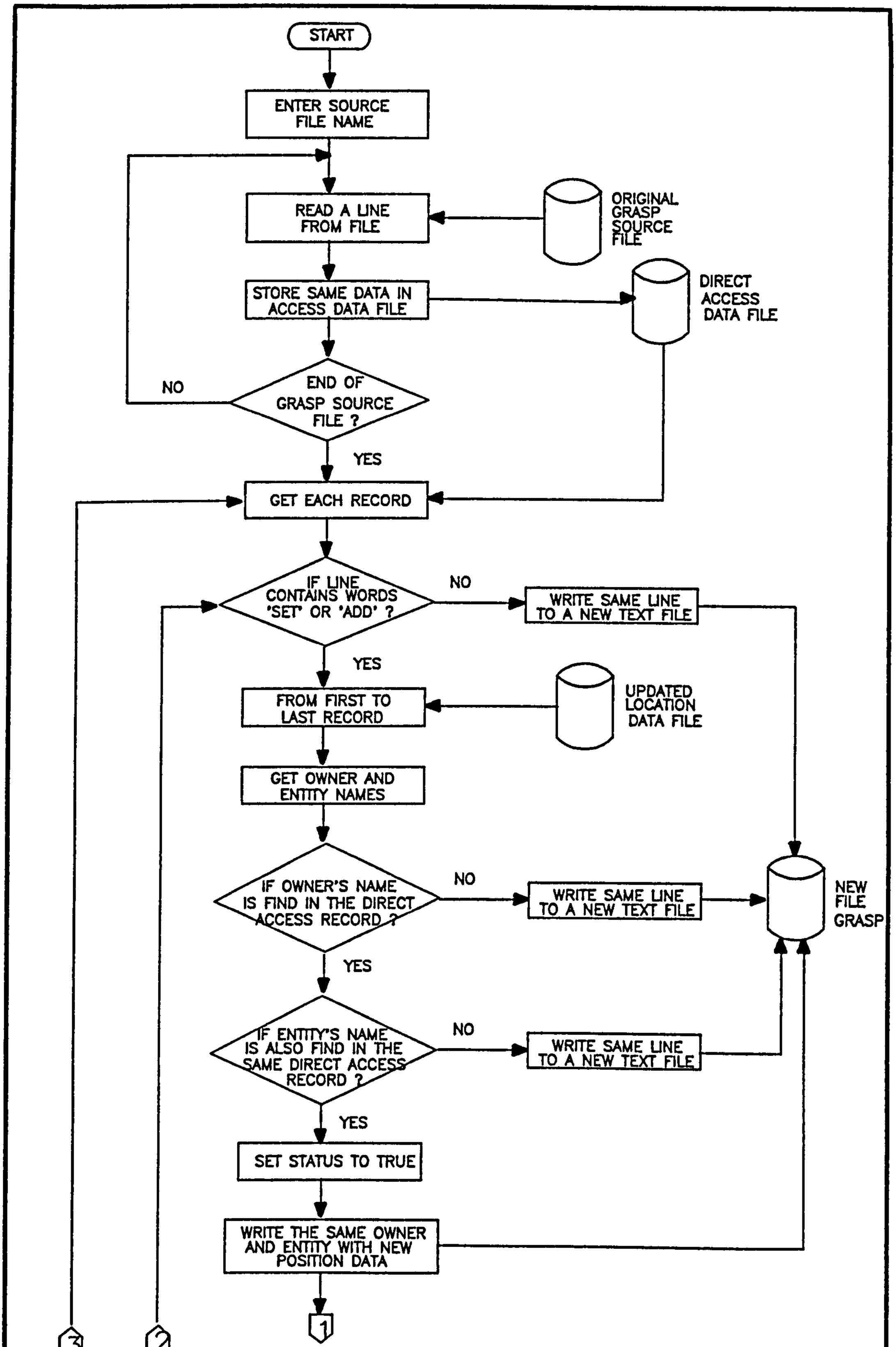



Figure 7–4 Flow chart of UPDATE module

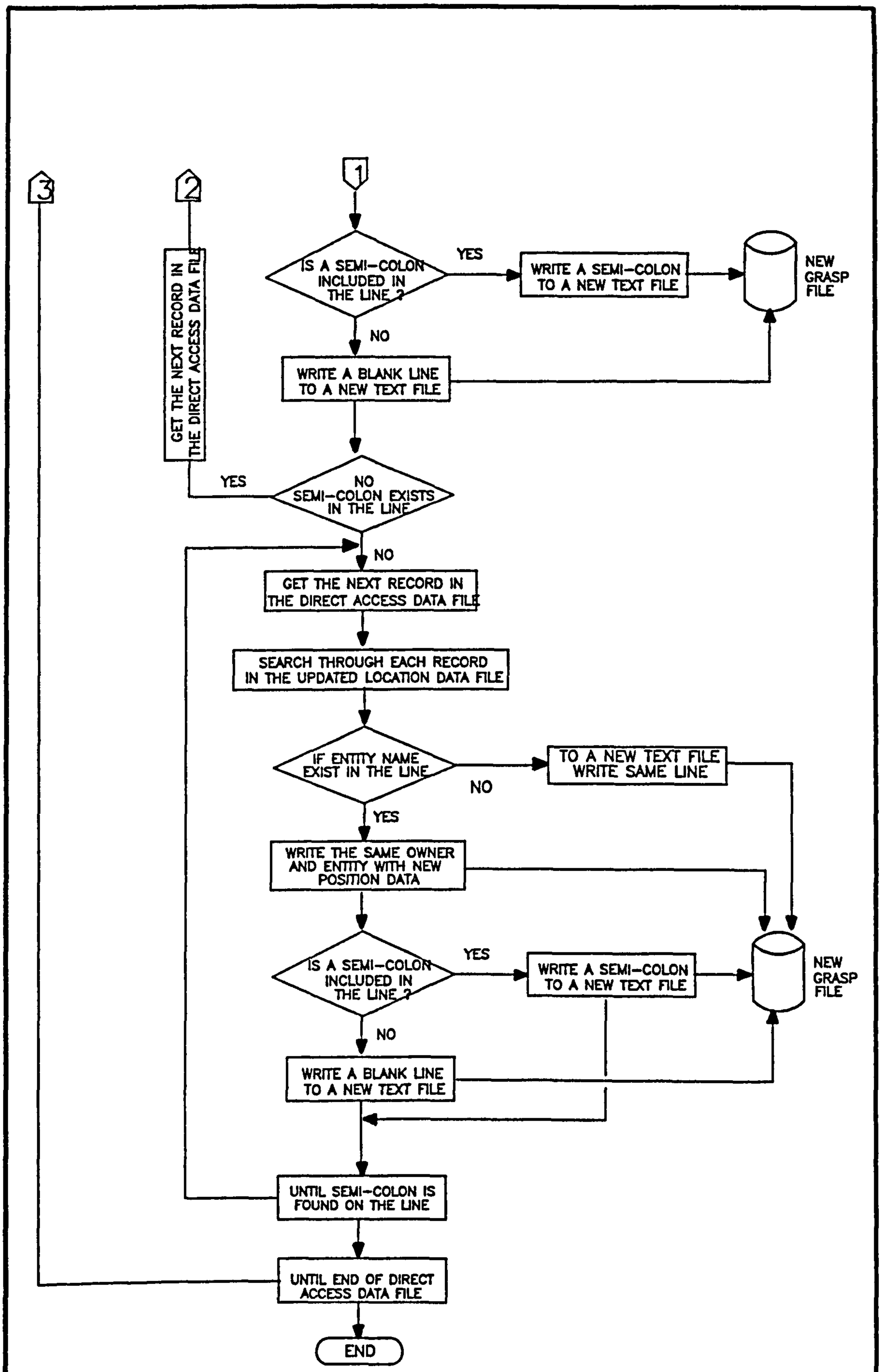



Figure 7–4 Flow chart of UPDATE module (continued)

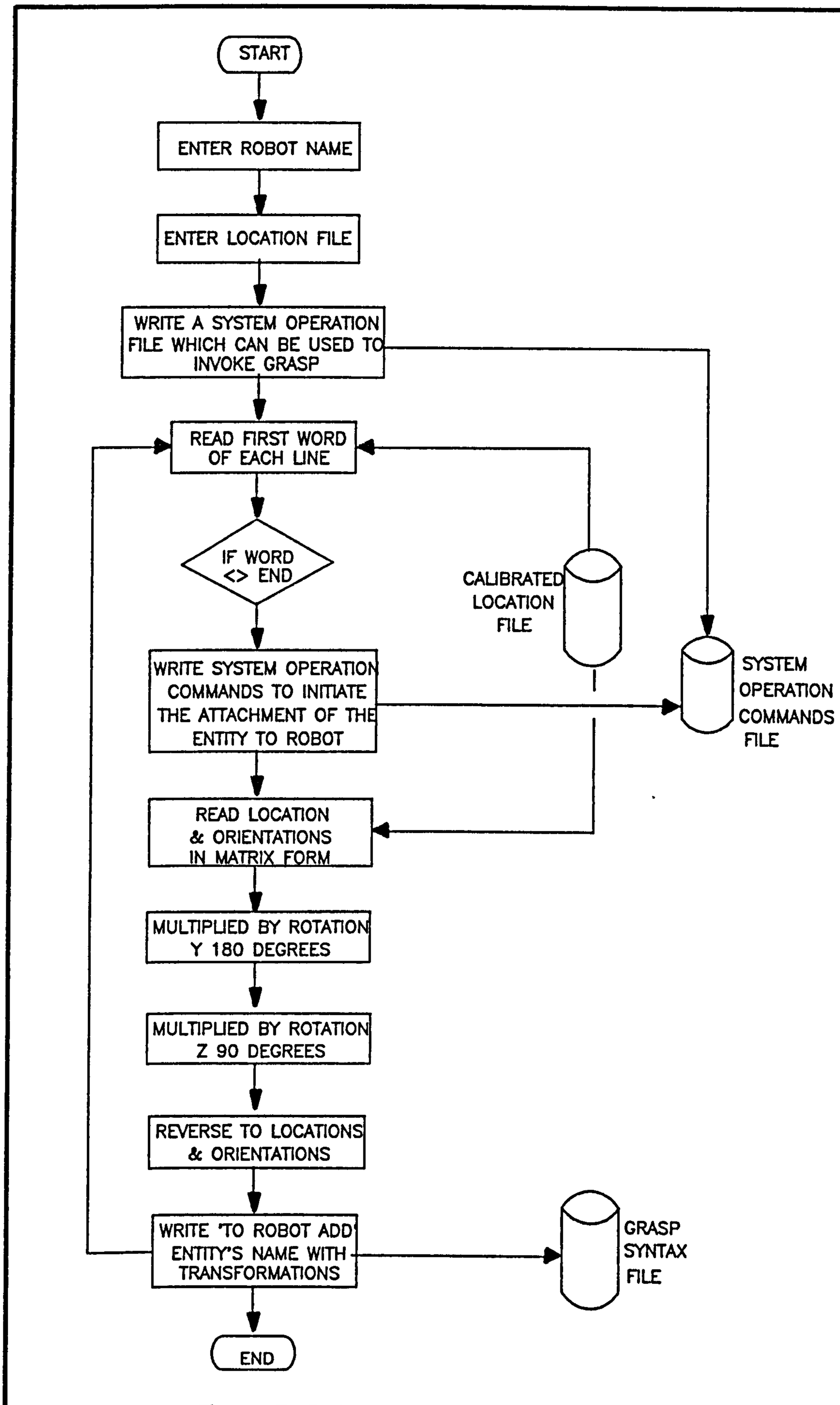



Figure 7–5 Flow chart of VALTOGRASP module

Theoretically, once the calibration is accomplished at any one point, the calibrated position and orientation would be transferred directly and automatically into the robot simulation.

Calibrating and updating a CAD simulation model with the use of the CALIB and UPDATE approaches can provide a new simulation model in a source file (text) without any changes in the hierarchy of the data structure in which the entities' geometric relationships are stored. This provides the advantage of always keeping existing simulated robot task programs in a valid condition. However, this method requires the use of an existing reference database which has been generated when the simulated robot task program has been pre-processed for off-line robot program generation as described in chapter five. This method restricts the calibration to be accomplished from top down i.e. the entity at a higher hierarchy should be calibrated first and then entities belonging to this entity. In the CALIB module, it is assumed that there would be no change in the owner objects location and orientation for any owner objects that have not been physically calibrated. The geometric relationship of the owner object stored in the database is used for transformation matrices concatenation. This may introduce error due to incomplete information obtained during the calibration.

The VALTOGRASP approach is flexible, and it utilises existing facilities residing inside the robot controller and the GRASP robot simulation package. For example, there is a frame function in VAL II which can return the location and orientations of a datum position. This approach is more efficient than the hierarchical approach.

The calibration procedures and software modules can also be used with the on-line editing and the datum point method described above.

### 7.2.2 Robot Calibration

Industrial robots are generally applied to routine work and on-line teaching methods are normally used to programme these robots. Since the robots or manipulators are required to perform their programmed tasks repeatedly, they are heavily reliant on their repeatability. Due to the nature of the work and the programming methods, robot designers have placed great emphasis on robot repeatability. Most of the currently available robots show satisfactory repeatability but poor accuracy [Driels and Pathre, 1987] (this being the de-facto situation unless Cartesian robot coordinate systems are utilised).

Many robots can also be programmed by language commands. Moreover, with the advent of Computer Integrated Manufacturing, robots must be programmed off-line; hence robot accuracy is becoming important to utilize off-line programming techniques based on CAD databases. This means the robot end effector is commanded to attain numerically specified positions and orientations with respect to a reference frame or datum. In other words, with this type of programming, positioning based on commanded locations requires a high degree of robot accuracy. However, the actual attained position may be quite different from that which the programmer desires, for example, as Baird and Lurie (1983) observed with the PUMA 600 manipulator, inaccuracies of up to 10 mm exist over a 200 mm straightline path.

In the past, there was a lack of an explicit mathematical model to analyze the errors. This means that the kinematic design of a robot manipulator could not be optimized. There has been growing interest and awareness in the importance of robot accuracy, and considerable improvements have been made. Still, there are accuracy problems which need to be drawn to the designers' and users' attention. These accuracy problems are attributed to the kinematic functions (not perfect) and the inaccurate knowledge of the workpiece reference frame in relation to the robot reference frame. The nominal robot kinematic functions are derived from the nominal geometry of a robot (with information obtained from the design drawings). These kinematic functions are often used in the robot controller for robots of the same model without taking into account the variations (manufacturing tolerances of robot components) in the assembled robots.

For off-line robot programming, the success or otherwise of the programmed task relies on the accuracy of the robot used. Because of the accuracy problem, programming a robot with off-line programming methods (i.e. without teaching the desired locations) will not permit the robot to perform satisfactorily. For this reason, it is important that the nominal functions can closely fit in all of the robots of the same model. This means that the tolerances specified in the design drawings have to be tightened. The manufacture of accurate robot components would make it a potentially expensive approach to improving positioning accuracy and may be considered as commercially not possible. An alternative approach (and likely to be a more economically viable approach) is robot calibration. Robot calibration is used to analyze the exact geometry of each individual robot to establish its unique kinematic functions [Chen

and Chao, 1987]. Once these robot kinematic functions are established, its inverse kinematic functions can also be obtained. These forward and inverse kinematic functions can replace the nominal ones used in robot controllers. In its simplest form, robot calibration can be described as a process whereby robot accuracy can be improved by modifying the robot positioning software (i.e. forward and inverse kinematic functions). The improved robot accuracy makes off-line robot programming methods more readily applicable and suitable. This means that a robot task can be programmed based upon (entirely) numerically specified locations or only a few reference frames (data) have to be taught and all other positions can be specified as relative coordinates with respect to these taught positions. This also makes robot task programs more readily transportable from one robot to another (either a robot of the same or different model).

## (a) Robot Calibration Levels

Although the procedures involved in robot calibration vary widely in their complexity (some deal with joint transducer information, some consider the entire kinematic model, and some even consider the dynamic model), most of the current robot calibration approaches can be categorised into three levels [Whitney et al, 1984; Roth et al, 1987] viz: joint, entire robot kinematic model, and dynamic model.

### (i) Joint Level Calibration

The purpose of calibration is to determine the correct relationship between the signal produced by the joint displacement transducer and

the actual joint displacement. This usually involves calibration of the kinematics of the drive and the joint sensor mechanisms [Azavidar, 1987]. The actual calibration procedure could be carried out using laser interferometry or similar techniques [Driels and Pathre, 1987]. The error motions can be tabulated and stored as a function of the joint variable. These error motions are likely to change very gradually with change in the value of the joint variable and so will be constant over a small range of motion of the joint.

(ii) Entire Robot Kinematic Model Calibration

The goal is to determine the basic kinematic geometry of the robot as well as the correct joint angle relationships. With knowledge of the manufacturing tolerances, the Cartesian error envelopes for the designed kinematic parameters can be predicted [Stone et al, 1986; Veitschegger and Wu, 1986]. Though a calibration method at this level can correct the kinematic errors of a robot, it complicates the controller's task in solving joint transformations.

(iii) Robot Dynamic Model Calibration

This applies to robots under dynamic control. If any changes in dynamic conditions of a robot are identified, then a correction for the changes will be made in the dynamic model. The dynamic errors considered here, are also referred to as "non-geometric" or "non-kinematic" errors in the literature. These non-kinematic errors in positioning of a robot end effector are due to effects such as those discussed earlier in section 7.1(c). At the present, not many

attempts have been made at this type of calibration.

## (b) Robot Calibration Procedures

In general, the robot calibration process can be considered [Roth et al, 1987; Okada and Mohri, 1985] to include four major steps viz:

### (i) Modelling Step

The first step in the calibration process is to choose a suitable functional relationship (i.e. nominal kinematic functions).

### (ii) Measurement Step

This involves the collection of data (position and orientations) from the actual robot that relate the input of the model to the output. There are three common methods available for measuring robot accuracy and repeatability. The cost of the measuring equipment varies according to the accuracy required which in turn depends on the application. Some of the more commonly used methods can be found in Lau and Hocken (1984).

### (iii) Identification Step

The collected data from the measurement step are mathematically processed to identify the coefficients in the model. This means the process is to determine the expected error in the identified coefficients due to measuring error.

(iv) Correction Step

After the coefficients of the model are identified, a new model of position control software of the robot can be implemented.

## (c) Possible Extensions of Robot Calibration

### (i) Kinematic Error Mapping

Robot kinematics can be considered as consisting of an infinite number of static positions. If this assumption holds, kinematic errors can be considered as a series of static errors i.e. kinematic errors frozen at points in time. In this situation kinematic error mapping can become a valid tool for error correction as an alternative to the kinematic equation formulation and calibration.

The mapping involves the accuracy testing of a certain area of the workplace where the robot is to perform its tasks. Typically this could be the assembly bench. The accuracy test is carried out at different speeds within the allowable operating range. The results are stored as a database which subsequently can be used in correcting inaccuracy of the off-line generated locations and orientations. This method also involves the writing of a software module for the search through the database to obtain the required offset correction. Suppose, the robot is commanded to move to location x at a certain speed specified in a robot simulator, the correction software should search through the database and use it as a look up table for the offset at the right speed range and the approximate position. The

offset is then added to the off-line generated locations.

(ii) Adaptive Control - Dynamic Error Compensation At Robot Controller

The method proposed by Lee et al (1989) was implemented by modifying the resolver feedback signal by the predicted error. A new trajectory control of a robot is proposed based on the off-line trajectory error analysis of the system (see figure 7-6). An Adaptive Linear Modelling algorithm [Astrom and Eykhoff, 1971; Eykhoff, 1974] based on the Recursive Least-Squares method [Strejc, 1980] is used to determine the approximation model. The principle of the implementation is that resolver signals are converted into joint angles ($\theta_i$). The world coordinates (Px, Py, Pz) corresponding to the measured joint angles ($\theta_1$, $\theta_2$, $\theta_3$) are calculated using the direct kinematic transformations. These world coordinates are then modified by the predicted error (Ex, Ey, Ez). The modified coordinates are finally converted back to resolver signals and returned to the robot controller. This method has been shown to achieve a 70 % reduction in errors.

This method tackles the problems of control, and requires experimental data to determine the approximate behaviour of the robot model. In this respect, it is specific to the off-line programming system concerned, and hence not general enough for different robot models. Although the claims of improvement in error seems quite promising, there are still problems of resolution and deadband.

Figure 7–6 The modified structure of signal feedback
(Adapted from Lee et al, 1989)

## (iii) Dynamic Robot Simulators

This type of robot simulator considers dynamic effects as well as the kinematic inaccuracy of a robot. At the present time only a very small number of dynamic robot simulators are commercially available (more detail was described in chapter two) including STAR, ROSI and ROBOT-SIM. Using this type of robot simulator, the dynamic effects of a robot can be predicted and therefore dynamic errors or inaccuracies can be corrected. Unfortunately, dynamic robot simulators can only be used to predict the performance of a limited number of robot types due to the many difficulties involved in the formulation of correct algorithms. However, the algorithms themselves are subjected to errors due to measurement errors and error sources that are probablistic in nature. Thus, dynamic robot simulators provide a solution to the specific problem and the generic solution has yet to be developed. The development of a generic algorithm for predicting dynamic behaviour of robots of different configurations can be considered as an impossible task.

It is not simple to assess the relative significance among all dynamic error sources. The significance varies from model to model. Experiments can be attempted to yield some measured values, but it may still be difficult to distinguish among the contributing sources [Chen and Chao, 1987]. Furthermore, it will be very time consuming to perform measurements for the individual robots to obtain error values which will be accurate enough to achieve the desired accuracy in position computation.

The simulation model calibration relies on robot repeatability and hence the data collected is not perfect. In addition, the robot calibration itself is also subject to inaccuracy due to error in the measuring devices and some unpredictable errors. On-line calibration may therefore be necessary to deal with the remaining inaccuracy. Normally, the simulation model, robot calibrations, and the refined kinematic/dynamic functions should be good enough for most applications.

### 7.2.3 On-line Calibration Methods Mapped Onto Operation Classes

As described earlier in chapter six, robot applications can be broadly grouped into pick and place, palletising, machine tending, assembly and welding. The existing calibration methods have been discussed in previous section and these methods are to be mapped onto existing application areas.

#### (a) Pick and Place Operation

Pick and place is the fundamental robot movement that other applications are based upon. Pick and place operation can be regarded as the simplest and therefore a simple calibration method is suitable (in terms of cost and accuracy). In most circumstances, the on-line editing approach is suitable for this type of application where the number of operation positions are limited.

## (b) Palletising Operation

In palletising operations, the robot operation positions are normally referred to a datum point such as the coordinate frame of a pallet, and thus the datum point approach is more appropriate. This is simple and yet accurate enough to provide good calibration. This calibration method can only be applied to palletising operations where the objects which are to be manipulated need not be distinguished from one another. If the palletising operation is dependent upon the object type, then some means of identification such as bar coding is required. The bar code reader is then used before the palletising operation is to begin. An alternative approach is to use a vision system to distinguish the object type as well as for calibration purposes. This will however generally result in higher cost.

## (c) Assembly Operation - Electronics

In electronics assembly, two-dimensional vision systems can be used for calibrating the workcell arrangement. Here the datum point approach with the assistance of a two-dimensional vision system is the most appropriate method of calibration.

In printed circuit board (pcb) assembly, there are two major types of entity that require calibration. With the off-line program, calibration is required to locate the delivered location of the electronic components and the location of the pcb where these components are to be assembled. Force sensing on the pcb coordinate frame is not possible due to the fact that a small external force disturbance can cause misalignment of the pcb. The alternative is to apply vision techniques to locate the pcb

coordinate frame. However, in the absence of a multiple camera vision system and for economic reasons, it would be better if the calibration of the coordinate frame of the electronic components (delivered by the feeding device) was carried out by teach without force sensing. The calibrated locations and orientations can be stored at the robot controller for use in assembly. Alternatively, in the case of assembling families of products then the locations and orientations of the electronic components delivered should be fed back to the simulation system for updating the simulation model.

(d) Assembly Operation - Mechanical

The use of sensory information can provide a method for correcting both static and dynamic errors. Let us consider the three possibilities of this approach where the batch size is the determinant factor. The three possibilities are: one off, batch and large product families solutions. The starting point in task calibration is that one is given a robot program for a given task along with all location data. All locations where accurate positioning is necessary are identified and assigned unique names in the location data file (unique names are the same as used in a CAD model). It is assumed that the robot will be stationary for some duration during normal task execution. For accurate assembly tasks, such as peg in hole insertion, assembly might require the assistance of a force sensor at such stationary points during program execution (i.e. at the points of insertion). The forces and torques exerted on the robot gripper can be sensed assuming that the correct component is being correctly fed and gripped. If this force is greater than the pre-set threshold value then the robot control software will conclude that insertion is not being attempted at the right location and therefore the

normal assembly routine is interrupted and a VAL II program with search pattern is called up to find a location where the force exerted on the gripper is less than the accepted value. From then the location is either used for insertion or the location is updated and the normal assembly routine is resumed. The choice is dependent upon the batch size involved (i.e. dependent upon the period of repetition). Three possibilities are open for consideration: one off, batch and large product families solutions.

## (i) One Off Solution

In this method, an off-line program is linked up to the on-line sensory search program through the use of a compilation program. This involves writing different VAL II (or other programs) to do a variety of search patterns in which the precise locations can be found. Another VAL II program is used to combine the off-line program with the search program. These location search programs when completed should be stored to build up a library. As more programs are written, the library will be expanded so that it will be easier for the programmer to retrieve the appropriate program for the type of job required. Since the robot is to be used once, the correcting of locations and orientations may not be necessary (as the sensor search program may be used each time a task is performed). This may take a long time and is inefficient for batch operations.

## (ii) Batch Solution

This is similar to solution (i) above, except that the locations and orientations are updated and stored in the robot controller. In this way

the search routine is not invoked for the second and subsequent operations of the task, except possibly through exception handling invocation. This significantly reduces the cycle time.

**(iii) Large Product Families Solution**

This is very much the same as solution (ii) above, but the updated locations and orientations are fed back to the robot simulator to modify the original simulation model. This solution has the advantage of being capable of accommodating large families of products where frequent reprogramming of the robot task is required. This raises interesting issues as to whether the modelling/simulation tools could be used with advantages in the robot controller where high speed information access/processing can be utilised.

For any simple mechanical assembly with parts locations and orientations pre-determined, force sensing would be just good enough. However, for complex assembly where parts and subassemblies are not always delivered in a fixed sequence nor located at fixed locations and orientations, compliance force sensing is not capable of identifying parts. Three-dimensional vision can be advantageous in this application, which can be used at a distance to identify and locate the parts in the field of view. The assembly process is monitored by compliance force sensing while the parts and subassemblies are in contact. This is particularly relevant when the parts and or the subassemblies are seriously misaligned and the compliance force sensing cannot accomplish the task alone [Russell et al, 1989].

(e) Welding Operation

A welding operation also requires calibration before the welding process is executed. Tactile sensors can be utilised to calibrate weld start and finish locations and orientations and to guide the robots in performing the welding task. In this application vision systems are rarely used for two major reasons viz: the severe lighting generated by the arc welding can seriously affect the performance of the vision system and the field of view of the vision system can only cover a part of the object or objects and hence it is not possible to train the vision system for recognition.

## 7.3 General Conclusions

In general robot accuracy depends upon the rigidity of the robot structure, link parameters and other factors earlier discussed. A prismatic joint will generally be more rigid than a revolute joint and hence the error or inaccuracy introduced by a revolute joint is considered to be more significant.

It is evident that many of the error sources identified here are not easily quantified. Sometimes the best that can be done is to make a carefully judged estimate of error magnitude. It is probable that many users of robotic systems do not have a complete understanding of the positioning error sources involved. Even when error sources are well understood, it is often difficult to get the required accuracy at an acceptable cost. These factors lead many systems to operate on the outer fringes of acceptable accuracy. Such systems require constant monitoring

by the programmer and frequent "touch-up" is necessary to maintain the path programmed.

In general, simulation model calibration is a sensible approach to modify discrepancies between the idealised CAD model and the real world environment. Although the robot calibration methods of formulating an accurate kinematic function or static error mapping can be useful in correcting kinematic error (assuming the robot is stationary at a certain point in time), the variables involved which may affect the robot performance makes the calibration too time consuming. In precise operations where accurate robot movement is required, static error mapping does not provide the necessary information on how a robot will behave dynamically. For this reason, methods of dynamic error compensation are required, in the form of dynamic robot simulators or dynamic effects prediction. However, each robot model can only represent a particular robot since no two robots are identical in every aspect even if they were of the same model and supplied from the same manufacturer. Their dynamic behaviour will also suffer variation from time to time due to maintenance, temperature changes, etc. This makes the dynamic equations difficult to develop and the advantage of this approach will not be realised until these barriers can be overcome. Perhaps, the residence of a robot simulator in a robot controller would provide the necessary functionality.

Although, geometric, kinematic and dynamic accuracies can be improved, it is expected that there will always be residual inaccuracy inherited in the whole off-line robot programming system. On-line sensory feedback can be used for the remaining inaccuracy. Force sensing can assist the

robot in carrying out assembly tasks, but the searching time (for ensuring the assembly is attempted at the right position and orientations) involved are usually quite significant (depending on the precision of the operation) even in a simple assembly operation. An alternative solution is to use vision systems to locate the centroid of each of the assembly points concerned. Although the search time can be improved, recognition time can become significant and should require further consideration. Furthermore, uncertainties arises in the centroid positions (normally it is small and may be regarded as second order).

If one-off assembly with many assembly operations (i.e. assembly points) is to be carried out, force sensing and vision systems are inadequate as the searching/recognition becomes a large portion of the total cycle time. This problem is particularly prevalent where families of products are to be assembled. A typical example is in electronics components insertion, where hundreds of items are to be inserted in a printed circuit board. In addition to the long search time required, force sensing could disturb the pcb location. An alternative is the integration with CAD product design data so that there is no need for searching, instead, one teach point (i.e. the frame of pcb) can be used as a reference point, to which other electronics components are referred. This integration approach is discussed in chapter eight.

# CHAPTER EIGHT

# INTEGRATION OF PRODUCT DESIGN DATA WITH AN OFF-LINE ROBOT PROGRAMMING SYSTEM

## 8.0 Introduction

At the present time most CAD systems operate as isolated islands of computer technology. However, as previously discussed, the goals of CIM cannot be fully realised without utilising electronically the information created at product design for administering, organising and programming the manufacturing machines. Thus the output from a CAD database will have limited practical value unless it can be used to support decision making processes and ultimately facilitate the generation of machine programs (including NC programs, robot programs etc). Thus future CAD/CAM systems will be more fully integrated with other factory computer systems with design information forming part of an integrated database which is likely to be distributed [Rui et al, 1988]. Thus for each product, design engineers will develop a product model and enable manufacturing engineers to reference it using a CAD/CAM system to perform operations such as manufacturing planning, assembly and inspection. Each of these functions will access the same product model and add its own information to the database.

As indicated earlier, future generations of robot simulator are expected to have an increasingly important role in the evolution of CIM systems. To provide an insight into one fragment of this problem, the possible form of an interface to product design will be considered.

Consider the specific case of an electronic manufacturing environment where a robot is to be used to insert "odd form" components into a family of printed circuit boards (pcb's). Suppose also that the pcb's are to be manufactured in small batches and have been designed using a proprietary CAD facility. In such an instance, various information concerning the artwork

on the pcb's, the components types and their geometry and the required location of the components on the pcb's will be stored within libraries in the pcb design system. Although this information is likely to be stored using proprietary data formats, it will exist in a machine readable form and can be utilised in generating the product model for robot simulation. Clearly, however, a data link of some form (which may be automated or involve manual intervention) must be established and processing facilities provided to reformat the information into data structures which can be integrated with the workplace model. Such an arrangement could yield significant advantages where the importance of better integration into CIM is evidenced as a measure of productivity [Groover, 1980 and 1987]. As reported in Computing Equipment (1985), a productivity ratio of 4 to 1 exists in favour of using CAD in comparison with manual methods. Furthermore, according to this study when CAD is fully integrated into CAM, the productivity increase is of the order of 40 to 1. Although the productivity increase attainable will be 'industry' and 'application' specific in nature, the opportunities for lead time savings is a major driving impetus for this study, and indeed other studies worldwide, where robot/workplace simulation and off-line programming are involved.

The specific study presented in this chapter serves to illustrate that for robot simulators the modelling process can be considered to comprise three constituent elements, namely a model of (i) the robot manipulator, (ii) the robot workplace and its tools, and (iii) the product. Having chosen a robot and determined its attendant workplace equipment, the arrangement so formed will remain fixed in most situations and so too will their associated model elements in any simulation. However, when batch manufacturing is involved, changes in the product occur which may or may not have implications with

regard to tools, feeders and fixtures. Thus where a large family of products are involved, or where complex geometric descriptions of products are concerned, a link to product design can be of significant benefit.

Furthermore, significant benefit can be gained by establishing data links to other computer based manufacturing activity areas. For example an interface to an automated process planning system could access valuable information concerning the sequence of manufacturing operations to be performed. In our pcb assembly example process planning information might already exist, in machine readable form, describing the sequence in which sub-tasks should be performed in the simulation process and subsequently in the robot task program.

The success of any integrated solutions as described will utimately rely heavily on the success of standards initiatives such as MAP/TOP, EDIF, IGES, PDES, IMDAS and AUTOMAIL as described in chapter three. Such specifications will allow manufacturers to supply their automation products with standard interfaces, thereby allowing standard product descriptions to be stored and transmitted across these interfaces.

Integration of a printed circuit board design system (REDBOARD) with an off-line robot programming system (GRASP) has been implemented as part of this study to illustrate possible problem areas in establishing integration and thereby to evolve methodologies inherent in such integration exercises. A printed circuit board design system will provide the designer with facilities for pcb layout design, but it should also reduce the need to manually produce a mask for pcb artworks, define electronic components insertion locations etc. This linking of design and manufacture/assembly

information is of course not restricted to the electronics industry, but also applies to the whole engineering spectrum. In many ways, however, pcb application areas provide a sensible starting point for study as such products are simple in the sense that their functionality is not usually related to their 3D shape as is often found in many conventional electro-mechanical and mechanical products.

The particular printed circuit board design system used in this study was the REDBOARD software which was produced and supplied to run on an IBM PC/AT by RACAL-REDAC Limited. When designing the layout of a printed circuit board with REDBOARD, the module called PCB is invoked. A library of electronic components is available within the system, and includes a 2D description of most common integrated circuit chips (IC's) available today. From this library, specific electronic components can be selected and located with respect to a 2D graphical representation of the printed circuit board. After all the electronic components have been placed at their desired locations, connections can be routed either manually or automatically using the AUTOROUTE function.

The completed design is stored in a binary format which is only meaningful for this particular system, i.e. the information is stored using proprietary data structures and formats. At the present time, the RACAL-REDAC company supply various postprocessors to enable the design output to be used for plotting the pcb artwork for photographic processing, generating NC programs for drilling and cutting out the outline of the pcb, and generating assembly programs e.g for Automatic Component Insertion (ACI) machines. Although REDBOARD provides output information which can be used in the programming of a variety of processes associated with pcb manufacture

and assembly, there are many other CIM activities which require reformatted fragments of this information. Examples of such entities are robots, vision systems, cell controllers (i.e. computer systems used to support decision making and control functions for a group of manufacturing machines or assembly workers) and shop controllers (i.e. providing decision support and control functions for a shop, such as an assembly or drilling shop). Here we are primarily concerned with the use of the information for the programming of robots (which can of course be classified as a general purpose programmable machine).

The binary output from the REDBOARD design system is not directly suitable for further post-processing for the creation of robot programs. The information can however be converted into a text (ASCII) file, using a REDBOARD system software module (called PASCII). The text file generated contains information which can be processed via the author's processing modules to produce three dimensional solid models (based on Boundary representation) of electronic components, and the pcb for use in the GRASP robot simulator. Significant potential, in terms of time savings in the design to product cycle, is promised if robot-independent programs can be generated for electronic component insertion, based on the use of existing design data.

## 8.1 Discrepancies Between the Design and Off-line Robot Programming Systems

Necessarily, this study has centered on the use of specific design and off-line robot programming systems through the use of REDBOARD and the GRASP simulator extended by the author. The study has highlighted six

major discrepancies between the specific pcb design system and that of the chosen off-line robot programming system. Although those discrepancies listed below are thus related to specific properties of the combination of GRASP and REDBOARD, they illustrate features of the general integration problem.

(a) Purpose

Generally, the current generation of pcb design software tools are clearly intended for design purposes. Hence, apart from producing masks for the artwork and NC tapes for drilling through holes, they have not been conceived to serve other purposes. In particular there is no information concerning assembly sequences or pick up points. A process of iteration is considered by the author to be inevitable here, with the needs of CIM specifying the characteristics of future design tools.

(b) Model Representation

The REDBOARD system produces a two dimensional pcb layout design whilst GRASP uses a three dimensional solid model representation. There is a lack of information in REDBOARD concerning the third axis (component height) and generic conversion from a two dimensional to a three dimensional model or representation is not possible. In the pcb assembly application examples studied, this problem can largely be solved by assigning individual parameters to the third dimension of components with the thickness of the pcb usually being constant for a given product (this being the method used in this study), although pcb's of different thicknesses and types (e.g. multi-layered boards) must be catered for in

general case.

## (c) Spatial Relationships

Major discrepancies relating to spatial relationships may occur. For example with REDBOARD, the locations and orientations of electronic components on the pcb are referred to the current pins at the bottom left hand corners of the electronic components after any rotation. However, with GRASP the objects' locations and orientations are defined by relating the objects' frames to the frame of the pcb. The principles embodied here are illustrated by figure 8-1.

## (d) Resolution

The increment in dimension (i.e. the resolution with which modelling occurs) must be consistent. For example, the parameters given in REDBOARD are normally (but can be modified as required) in units of 0.635mm (4 units being equivalent to 0.001 inch, i.e. the pitch between pins on many IC's) compared to the 1mm unit of GRASP.

## (e) Dimensioning

A further problem identified relates to pcb design systems where generally the outline dimensions of components are the nominal sizes rather than the actual dimensions. The difference in dimensioning is shown in figure 8-2, and clearly can have significant implications when accomplishing the actual assembly functions.

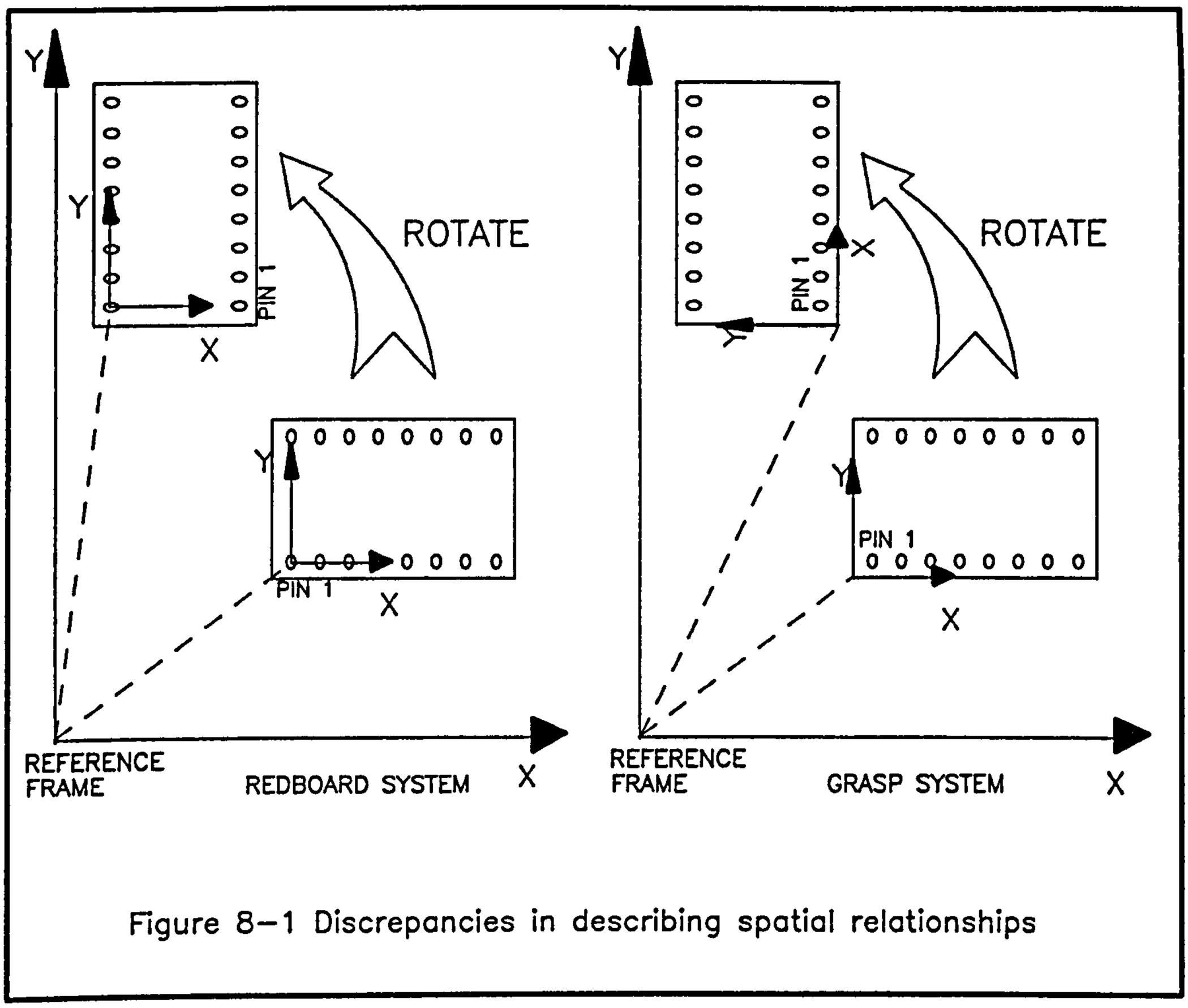



Figure 8–1 Discrepancies in describing spatial relationships

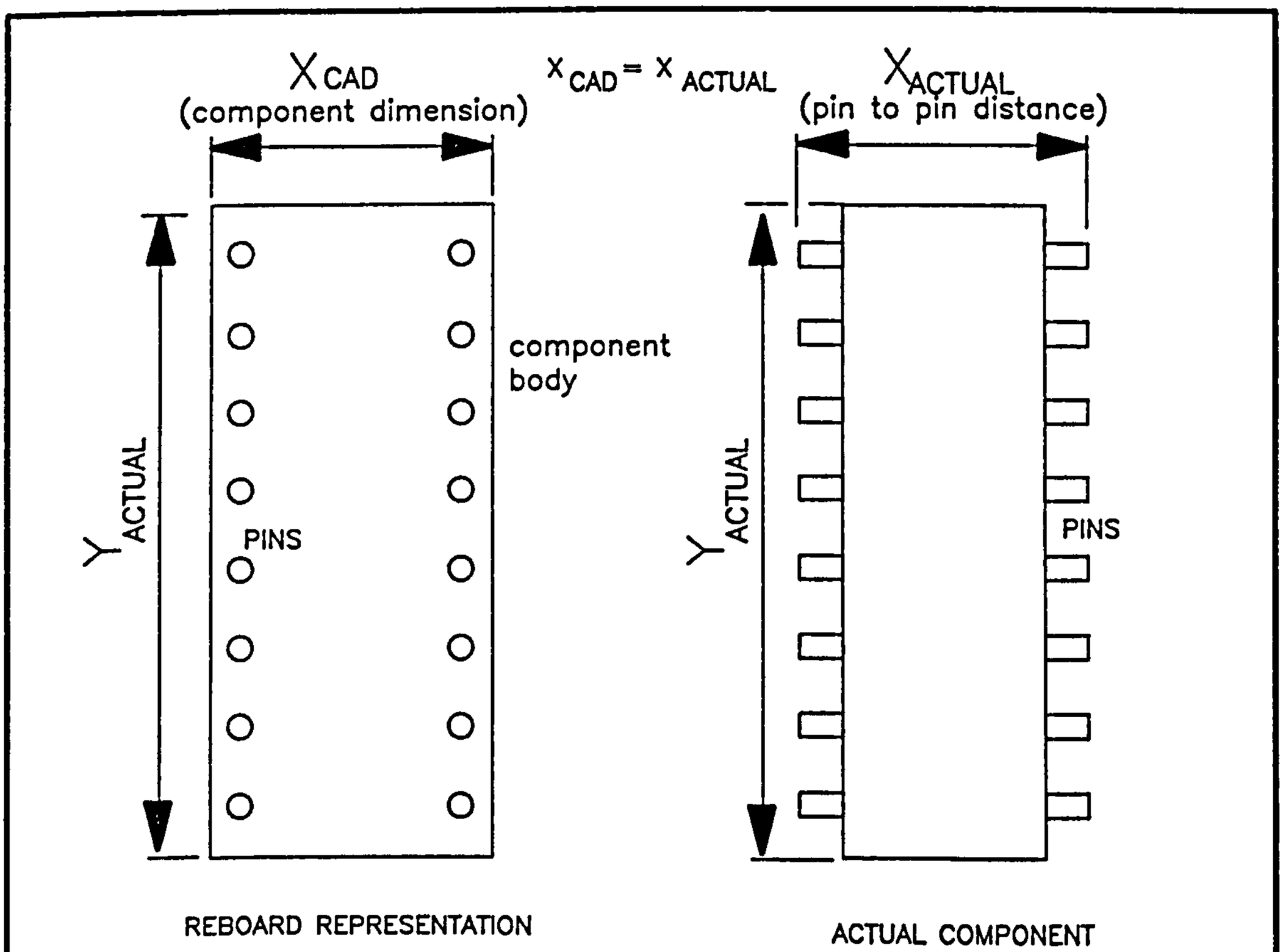



Figure 8–2 Redboard representation of an IC chip

(f) Data Format

The way in which information is represented and stored will generally be different between the design and programming systems. For example, the syntax or data format used in the REDBOARD design software is significantly different to that of the GRASP robot simulator and reflects many of the observations made in (a) to (e) above. Appendix D.1 provides examples of the formats of the two systems.

## 8.2 Design Approach Used for Integrating a CAD System with an Off-line Robot Programming System

The purpose of the integration software evolved in this project is to bridge the discrepancies between CAD and off-line robot programming systems and to provide a platform for generalisation. Again the actual software implementation which achieves this is specifically designed to integrate REDBOARD and enhanced GRASP entities, but the underlying rationalle was to create a mechanism for generalisation. This section describes the software produced.

Since the REDBOARD design system can only provide two dimensional information concerning electronic components, this information must be converted into three dimensional form for subsequent GRASP solid modelling and for robot task simulation. The required locations and orientations of the components on the pcb are used as the destinations at which the components should be assembled. However, no information is available within REDBOARD to define the pick up positions (i.e. the position of feeders for the electronic components) which

itself can present complex problems. The integration software evolved generates text files which contain GRASP syntax describing (i) solid models of pcb's and the associated components, (ii) pick up location assignments and (iii) task simulation programs.

A program describing the robot assembly task is generated which is suitable for the specific robot simulator.

In the case of creating a new model, spatial information may not be available concerning the workcell arrangement. That is, the location of pick up points and the working frame of the pcb board may not be known initially, and the practice of teaching reference points of the model was conceived, based on "on-line" teaching methods. When using this practice human intervention takes place to calibrate the workplace such that positional and orientational information of key elements (obtained using the on-line teach methods described in the previous chapter) are fed back to the simulation system. In this example the required feeder positions and the pcb coordinate frame are updated, ready for simulating the assembly performance and eventually producing an off-line robot program for the assembly task (figure 8-3 conceptually illustrates the principles involved).

The methodology derived does not exclude the approach of allowing the simulation to proceed with key elements located at arbitrary predetermined positions and orientations, thus permitting an initial evaluation of the model. Subsequently, however, the exact locations and hence the "on-line" teaching of locations and frames will be required. In certain circumstances teaching can be viewed as a "fine tuning"

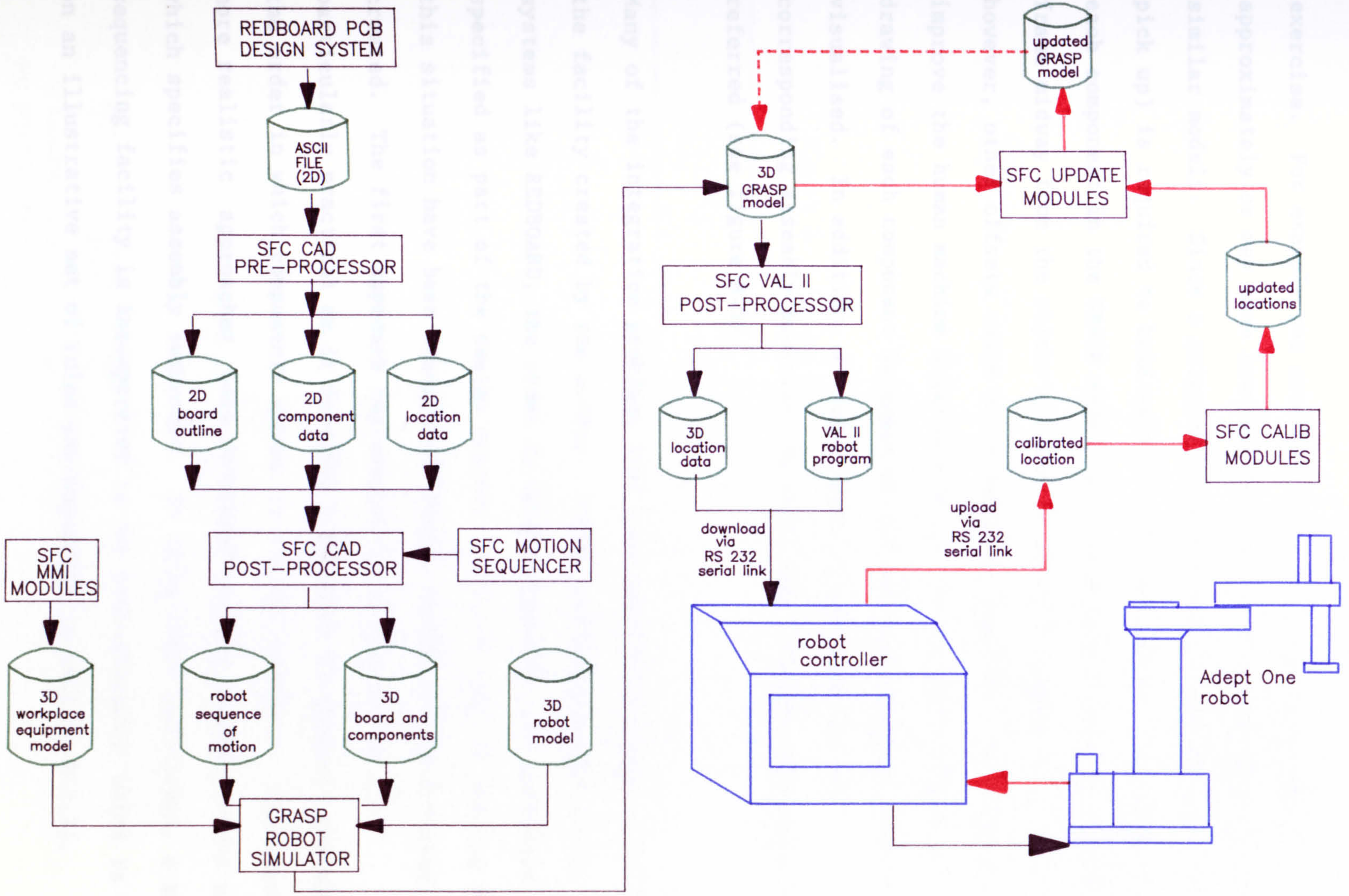



Figure 8–3 Overall integration of product design data with off–line robot programming system

exercise. For example the arbitrary positions can be calculated approximately or may have been established in the generation of other similar models. Since a gripping mechanism (which may involve vacuum pick up) is required to hold each IC chip (normally at its centroid), each component in the GRASP model is affixed with a pick up reference frame midway from the object's frame as shown in figure 8-4. Clearly, however, other offsets could be included as required. In order to improve the human machine interface to the 3D simulation facility, a drawing of each component is shown on the pcb such that any error can be visualised. In addition, a target frame is assigned to each corresponding assembly position, to which the insertion locations are referred (see figure 8-5).

Many of the integration problems that have been overcome are specific to the facility created by the author. Using contemporary pcb design systems like REDBOARD, the order in which components are assembled is not specified as part of the design process. Several ways of dealing with this situation have been considered whereby sequential information can be created. The first approach implemented is simple but is not particularly practical as it involves a sequence assignment equivalent to the order in which components appear in the CAD database. Subsequently more realistic approaches were developed using a pseudo process plan which specifies assembly sequences. In using these approaches, a motion sequencing facility is incorporated in the post-processor which is based on an illustrative set of rules (as explained in section 8.2.3).

The whole approach is depicted by figure 8-6a and b which illustrates the role of the required pre- and post-processors.

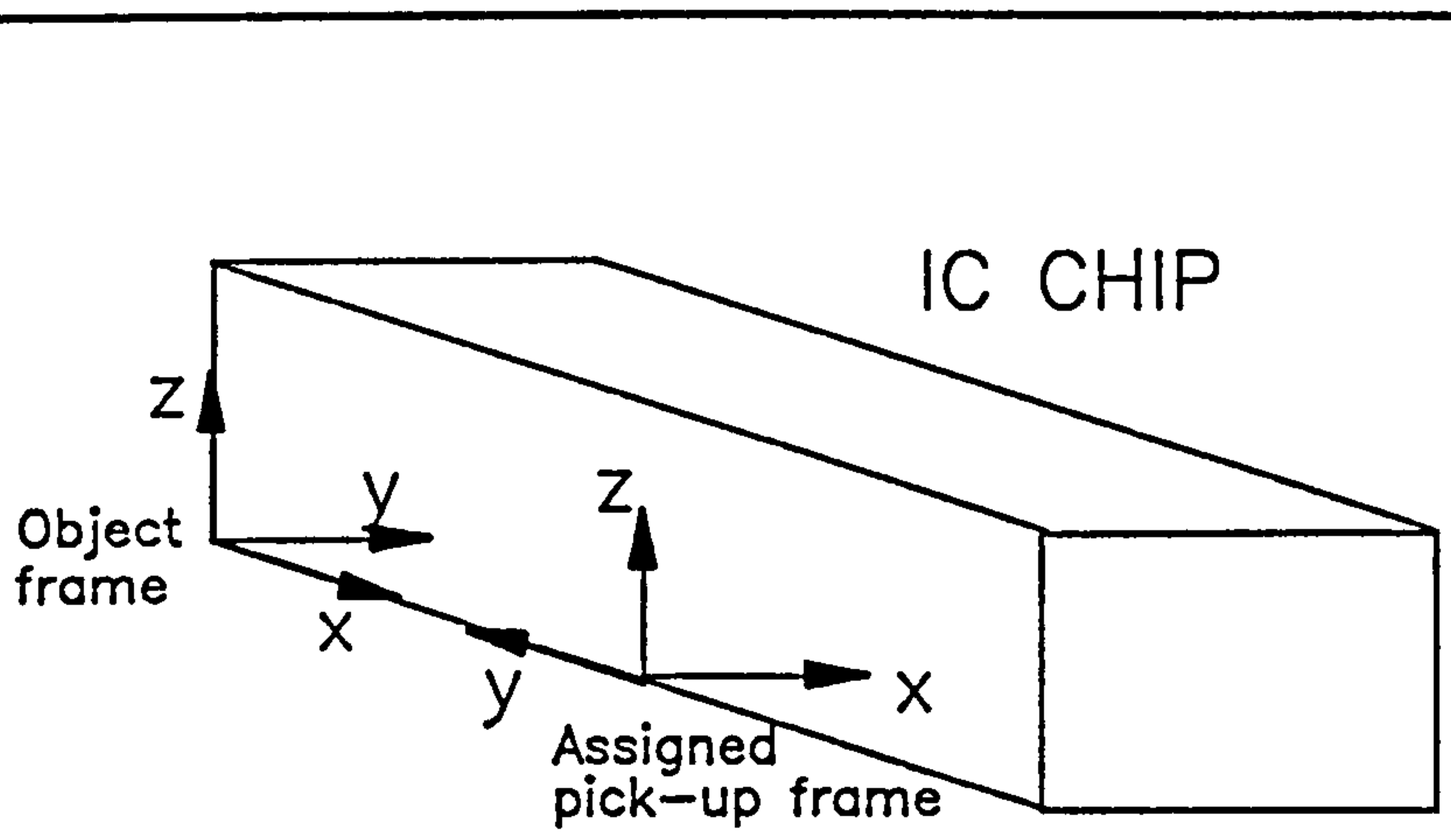



Figure 8-4 Frame assigned for pick up position

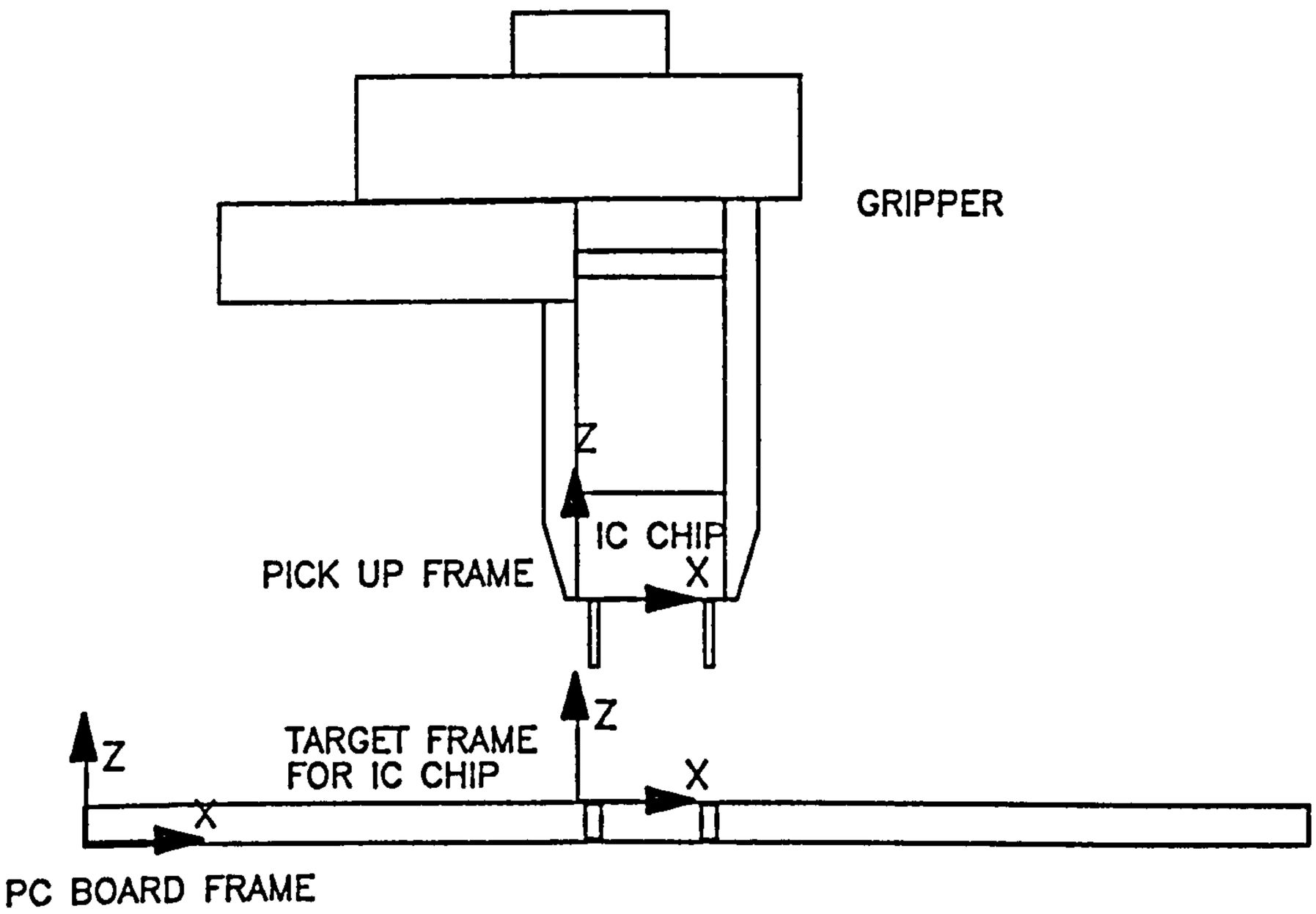



Figure 8-5 Electronic component insertion

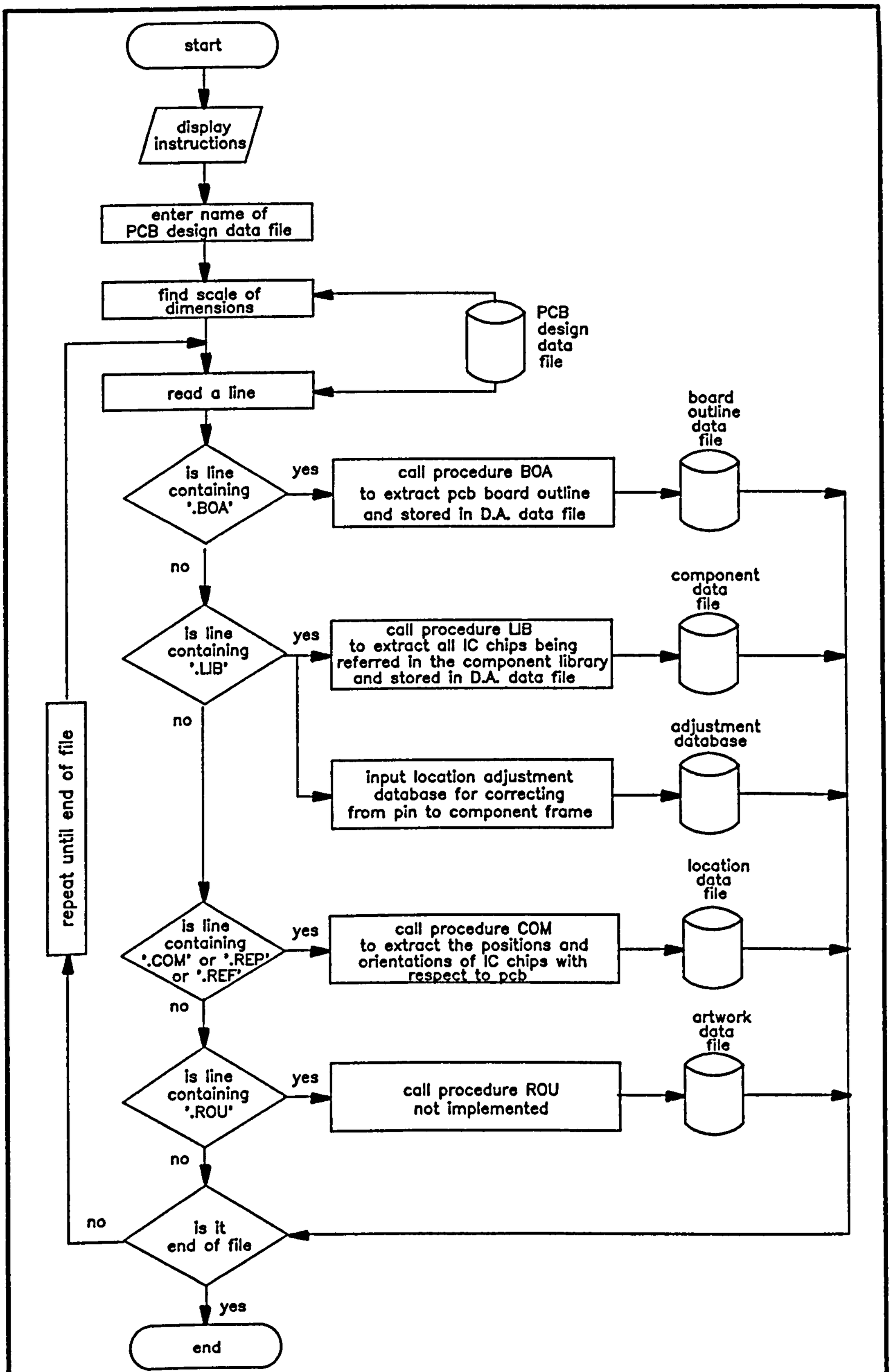



Figure 8–6a Flow chart of product design data pre–processor

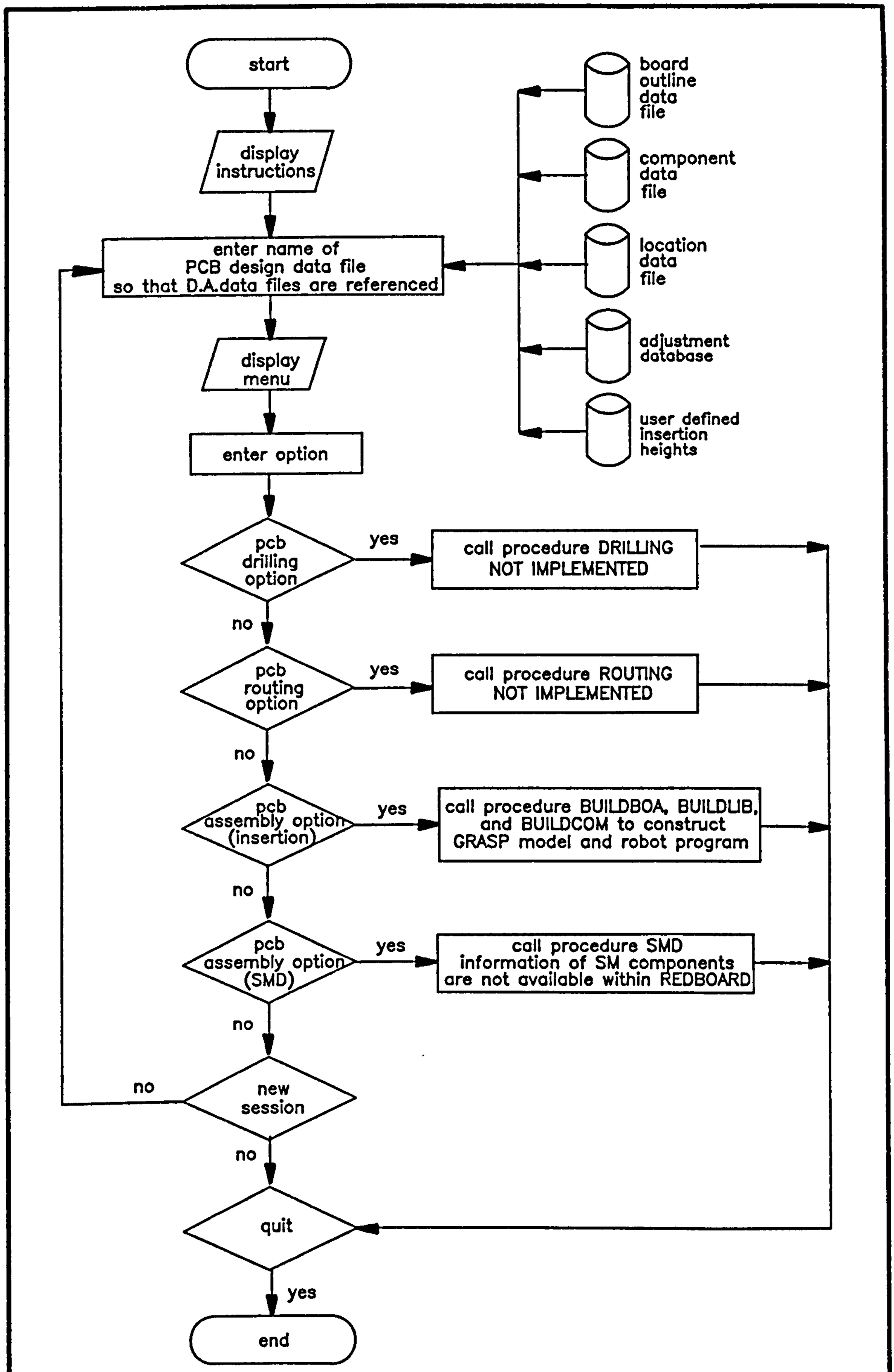



Figure 8–6b Flow chart of product design data post–processor

### 8.2.1 SFC Pre-processor

Since there was no neutral data format in existence, the integration of the pcb design system with the robot simulator can be achieved through using pre- and post-processors. The implementation of this method is specific to the design system and robot simulator involved. The purpose of a neutral data format is to provide a level of standardisation, defining common data structures for pre- and post-processing. Further discussion of pre- and post-processing methods has been presented in chapter five.

The SFC pre-processor reads in the ASCII output file of pcb layout design generated by the REDBOARD system, and transforms this information into a description of the 2D board outline, component definitions, and locations relative to the coordinate frame of the pcb. The transformed information is stored in three direct access data files. In addition a position adjustment database is automatically generated for adjusting the discrepancies between the reference frames as shown in figure 8-1. The offsets are calculated based on the type (number of pins, etc) and orientation of the components. The database functions as a look up table where the offsets are used to achieve correction in frame locations. The algorithms used for correcting these offsets are characterised by table 8-1. The abbreviations of dXmin., dXmax., dYmin., and dYmax. are explained in figure 8-7.

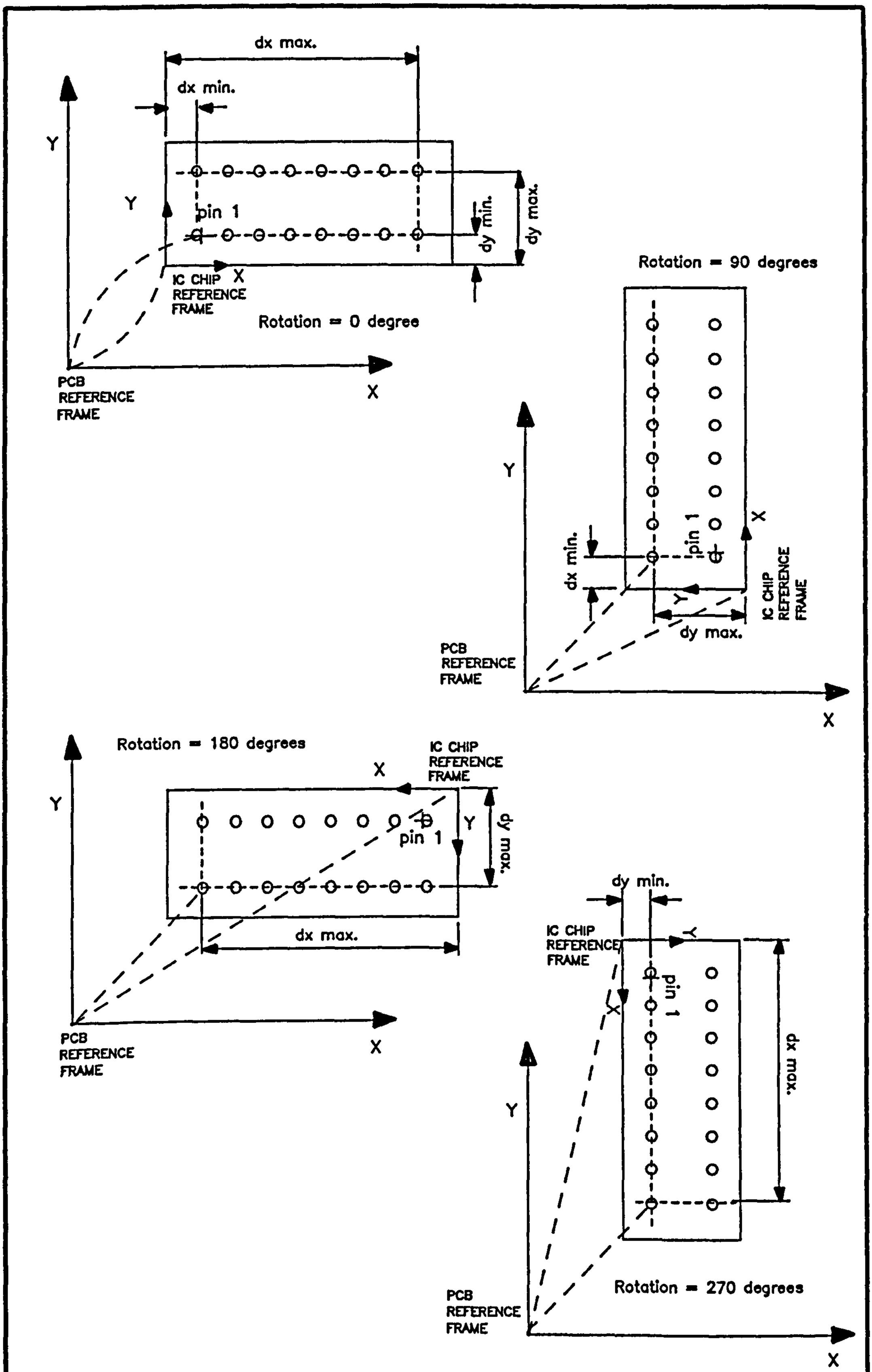



Figure 8–7 Correcting component location to frame location

| ROTATION | NEW X COORDINATE | NEW Y COORDINATE |
|---|---|---|
| 0 | X - dXmin. | Y - dYmin. |
| 90 | X + dYmax. | Y - dXmin. |
| 180 | X + dXmax. | Y + dYmax. |
| 270 | X - dYmin. | Y + dXmax. |

Table 8-1 Offsets Correction Algorithms

The formats adopted for these direct access data files were conceived by the author and will be referred to as the Pre-Processor Neutral Data Format or PPNDF. This information is thus readily available for further processing by a variety of post-processors each of which could transform the neutral data format into the customised syntax used by a proprietary robot simulator. The PPNDF adopted is illustrated by figure 8-8.

The SFC pre-processor is specifically design for processing data stored in REDBOARD format into the neutral data format. If a different pcb design system is used, a different pre-processor is required to create the same PPNDF neutral data format, thus maintaining the purpose of neutral data format.

### 8.2.2 SFC Post-processor for GRASP

The GRASP SFC post-processor reads PPNDF information from the appropriate direct access files, and converts the 2D model data into a corresponding 3D model in the GRASP syntax. This post-processor also generates a file which describes the relationships between the

NEUTRAL DATA FORMAT FOR PCB DIMENSIONS

RECORD BOARDNAME CORNERS POINTS

BOARDNAME = name given to a pcb
CORNERS = number of corners on pcb
POINTS = an array which stores coordinates of corners relative to the reference frame

NEUTRAL DATA FORMAT FOR COMPONENT DETAILS

RECORD IDNUM NUMPIN XDIM YDIM PINLOC PADTYPE

IDNUM = component reference number
NUMPIN = number of pins available on component
XDIM = X dimension of component body
YDIM = Y dimension of component body
PINLOC = an array which stores locations of pins on component
PADTYPE = type of pad used on IC

NEUTRAL DATA FORMAT FOR COMPONENT LOCATIONS

RECORD NAME IDNUM NUMPIN ORIENTATION XSHIFT
YSHIFT DIST REPORTED

NAME = name given to a component
IDNUM = component reference number
NUMPIN = number of pins available on component
ORIENTATION = orientation of component on pcb
XSHIFT = X location of component on pcb
YSHIFT = Y location of component on pcb
DIST = absolute distance of component from pcb frame
REPORTED = status of data usage

NEUTRAL DATA FORMAT FOR LOCATION ADJUSTMENTS

RECORD IDNUM ORIENTATION XADJUST YADJUST

IDNUM = component reference number
ORIENTATION = orientation of component on pcb
XADJUST = adjustment for offset in X location from pcb frame
YADJUST = adjustment for offset in y location from pcb frame

Figure 8-8 Pre-processor neutral data format for pcb design data

required pick up positions of the electronic components and their corresponding reference teach points. With the SFC insertion sequencer (see section 8.2.3) and user pre-defined insertion height for each component type, the sequence of electronic component insertions is determined by the decision rule selected, thereby illustrating the use of pseudo-process planning operations. The simulation task program is generated in two files with one describing the location of key entities in relation to their owner entities and the other describing the sequence of operations to be performed in the assembly task.

### 8.2.3 SFC Insertion Sequencer

At the present time there are no generally accepted rules or methodologies relating to the order in which electronic components should be inserted into or onserted onto pcb's. Investigation has revealed (through discussions with leading pcb manufacturing companies) that commonly each company has derived its own set of rules which depend on the type of insertion or onsertion machines used (e.g. axial, radial, IC, SMD, robot) and the auxilliary devices being used, such as the gripper, and feeding device.

In the absence of a generic methodology the following rules have been implemented and they should be treated as examples which demonstrate the principles involved rather than exhaustive or particularly realistic. Five simple rules which have been used fairly widely [Marconi, 1987; ICL, 1988] are described below.

(i) In the electronic industry, components are normally assembled in a pre-defined order at different assembly heights, e.g. the component with the most pins would be assembled first at the lowest level. Components with the largest number of pins and which appear first on the database should have top priority. This rule is based on the reasoning that in some circumstances, the assembly of smaller components before larger ones can be impractical and result in collision. This problem is usually referred to as the 'foot print' problem.

(ii) Components with the largest number of pins should be inserted first. If more than one component has that number of pins then the one with its insertion/onsertion position at the closest distance to the frame of the pcb should have top priority. This rule is based on the same reason as (i) above but with a different sequence of assembly.

(iii) For certain types of assembly machine, 'foot print' problems can occur where the gripper may collide with the components already assembled. The foot print problem occurs with certain types of gripping and feeding mechanism but not for all. For example, the use of a vacuum pick up mechanism to hold the electronic components will not have attendant foot print difficulties. The gripping mechanisms available for this research study exhibited foot print problems and hence there was a need for a more realistic rule to produce successful assemblies and avoid collisions. Assuming the gripper has two locating faces arranged to pick up components along the Y axis of component, the following rule can be used to define the order in which

the components should be assembled such that components requiring the least displacement of the robot's end effector along the Y axis are assembled first. If more than one component satisfies this requirement then preference is given to the component which requires the least displacement along the X axis from the taught frame (normally the pcb frame). Subsequent assembly is then continued along the X axis. Although this rule does not minimise the total distance travelled by the robot's end effector (insertion head), it provides the optimal solution (avoiding interference with previously inserted components during the opening of the gripper) for the demonstrator system constructed by the author.

(iv) Assuming there is no need to change the gripper or insertion head during insertion (onsertion) of IC's, a rule based on minimum distance of travel from the current position of the insertion head can be implemented.

(v) Rules based on a minimum distance of travel from one assembly position to the next (optimal cycle time) may be best if the components are automatically fed into the gripper (insertion head). There is no need for the gripper to pick up components from peripheral feeding devices. Based on this reasoning, components are sequenced according to component type (same definition in component library i.e. number of pins) and minimum distance of travel. A component of each type (with maximum number of pins first) which is to be inserted at the minimum distance from the frame of the pcb is chosen to be inserted first, and subsequently, any component of that type which is to be inserted at the nearest position to the current position of the

insertion head should be inserted next and so on. This method reduces the total movement of the insertion head (or end effector) and hence the cycle time. Clearly, however, 'minimisation' of cycle time may require complex rules particularly where multiple choices are involved. If a different gripper or insertion head is required for each component type, this rule also minimise the frequency at which gripper changing is required. Most important of all, this rule can provide a solution to foot print problems discussed earlier. Among the specimen rules implemented this rule probably represents the most commonly used in pcb assembly. However, this rule is not the most appropriate for the demonstration system where components are fed by peripheral feeders (see figure 8-9).

### 8.2.4 Integrating the Simulation Model with a Task Program to Generate an Off-line Robot Program

The 3D product (pcb and components) model and the task sequence together with the workplace equipment and robot model (selected from the robotic device library) form the input information to the GRASP robot simulator. Subsequent simulation of assembly operations can be used to investigate the operational behaviour of the proposed system. If no errors (including joint violation and object collision) are detected, then the task can be post-processed to generate a VAL II robot program for that assembly task along with information regarding important locations and orientations. The robot program is then downloaded to the Adept One robot controller via an RS 232 serial link. This program is then used to drive the Adept One robot for the intended sequence of motion.

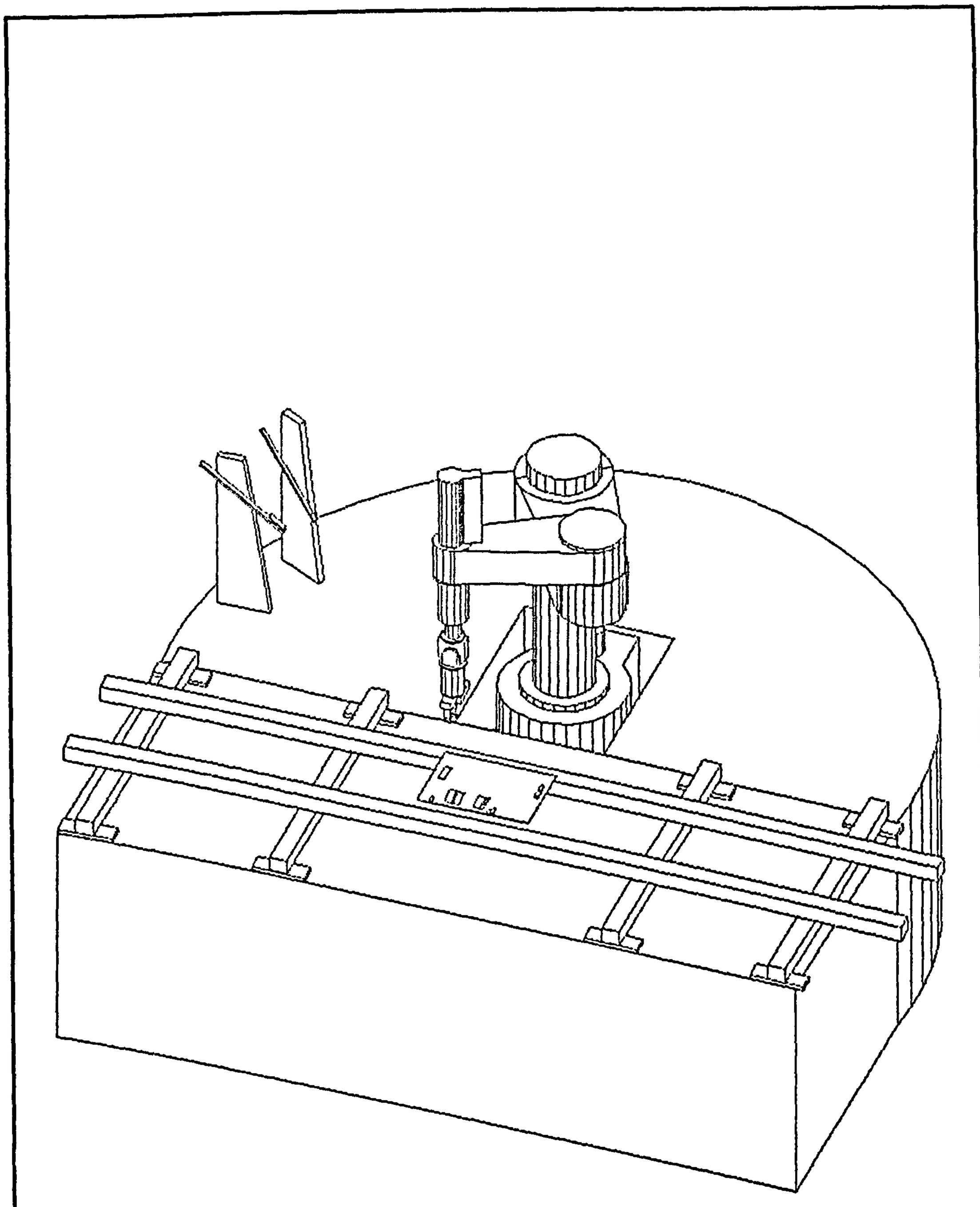


Figure 8–9 Demonstration system for pcb assembly

## 8.3 Difficencies of the Approach Implemented

Since the REDBOARD design system like many other contemporary PCB CAD systems can only produce meaningful information concerning the locations and orientations of components, it is impossible for any integration software to extract other types of information relating to assembly actions. The REDBOARD design system regards every item as a 'standard' electronic component (including edge connectors and switches etc), hence the need for complex integration software which recognises special features of the components in regard to their assembly needs. The need to provide more meaningful and comprehensive information could be addressed by adopting a new philosophy when implementing pcb design systems. For example the CAD system could assign a fixed prefix to components. In this way components requiring different assembly actions such as edge connectors (EC) and switches (SW) can be distinguished thereby providing the information with a contextual or semantic meaning. Alternatively, incorrect assembly motion would need to be deleted interactively, thus involving time consuming manual editing of the generated task sequences.

## 8.4 Experiment and Analysis

### 8.4.1 Experimental Set Up

The integration approach was verified through the construction of a demonstrator application and a series of experiments (see figure 8-9). A currently manufactured pcb was obtained from ICL and is considered to be representative of contemporary pcb's. The use of such a board

thus represents a realistic situation so that the practical problems encountered are likely to occur in conventional manufacture. The only reservation is that the only insertion hardware available for testing comprised a state of the art robot tooled-up for pcb assembly rather than conventional equipment.

The workcell is comprised of a worksurface, on which two sets of feeders are mounted, together with a clincher for the cutting and bending of pins of the electronic components to ensure correct lead lengths and fixing prior to soldering. Peripheral to the worksurface, a small conveyor system is linked with a main conveyor system for board transportation. A fixture holds the pcb while the assembly robot carries out the assembly. An Adept One SCARA is used as the assembly robot, and is located appropriately relative to the centre of the worksurface with the robot manipulator mounted on a stand to ensure that its height is suitable with respect to its working environment. A special pcb assembly gripper supplied by Meta Machines Ltd, was slightly modified for use in the assembly of a range of electronic components. The assembly robot and its working environment was modelled using the GRASP robot simulator. A camera was located on the fourth axis of the Adept One (see figure 8-10), so as to identify the board and find the location and orientation of the frame of the board with respect to the robot base.

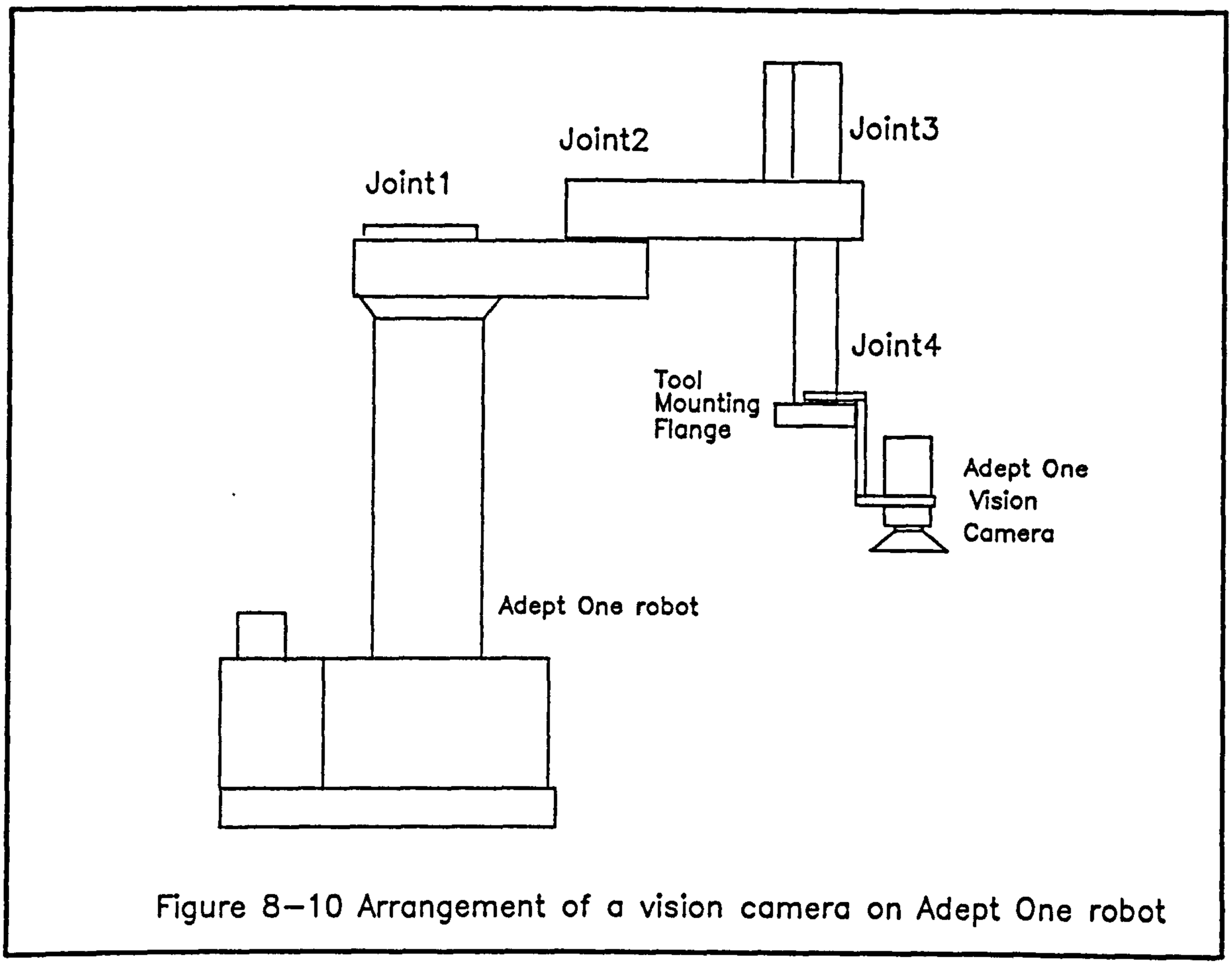



Figure 8–10 Arrangement of a vision camera on Adept One robot

## 8.4.2 Error Analysis for the Demonstrator System

The general sources of spatial inaccuracies which occur during off-line robot programming are described in chapter seven. Here a more detailed analysis of those errors is presented based on a specific study of the demonstrator system. This analysis is limited in its horizons but is used to partially verify the previous generic categorisation.

Experiments were conceived to measure the accuracy of the robot over a range of commanded translations with respect to a reference location (position and orientation) and to measure the variation in diameter of (drilled and plated) holes and pins (documentation of the results

obtained during these experiments are shown in appendix D.2). The emphasis of this experimentation is:

(i) to provide an indication of how the robot accuracy may vary,

(ii) to determine the approximate critical point beyond which the robot accuracy is worse than the part tolerance i.e. assembly becomes unreliable or impossible without implementing other enhancements.

The problem of hysteresis becomes apparent when the robot is commanded to move a long distance from its current position. In the electronic insertion example previously discussed, a pseudo process plan (insertion sequence) is chosen dependant upon features of the tooling and feeding equipment used. Consider the case where the robot is commanded to pick up an electronic component from a feeding device and insert it at the required location on the pcb. After each insertion, the robot is commanded to the pick up reference point, ready for the next insertion. Generally the distance between the pick up reference positions and the insertion locations will be large (up to 1 metre) so that phenomena which result in hysteresis which can have a significant effect on the robot accuracy. In theory, the robot accuracy is dependant upon the position of the robot end effector within its working envelope. The robot accuracy is also dependant upon the the direction of approach [Morgan, 1980] and its actual approach speed. The robot accuracy is expected to become worse during high speed and acceleration conditions [Tanguy, 1982]. The type of robot motion employed (such as straight line or circular path) can seriously affect robot positioning accuracy. These path following errors are dependent

upon the controller resolution and control laws. Another factor is the complexity of robot configuration, particularly when it is commanded to attain awkward positions, that is, arm positions too close to the joint restrictions. However, the Adept One robot of the SCARA type does not exhibit such complexity, and the effect of problem can be considered as negligible. The positioning inaccuracy due to accelerations and decelerations is affected by the moment of inertia due to the weight of the robot arm and the load carried. The dynamic performance is complicated by unknown and changing mass of the payload being manipulated [Mon and Broome, 1987]. However, in the electronic insertion application the payload can be considered to be negligible.

The results obtained from the experiments and the analysis presented in Appendix D.2 show that in general a small commanded distance from a taught reference point will result in smaller static position errors than those for large displacements. In fact the experimental results also indicated that this improved accuracy is maintained for small moves even when the robot approaches the required position at different approach speeds. Although the results do not formally quantify the approach for all robot types it is evident that the effects of unknown hysteresis, transmission characteristics, bending and controller characteristics should be smaller for small deviations [Jeyachandra et al, 1986; Benhabib et al, 1987]. The technique can thus be equated to a compromise between teach and off-line programming. The results also show a dependance on the approach speed and the commanded distance from the reference point. It is found that the position errors accumulate (i.e. the accuracy becomes worse) with in this case an increase of approximately 0.05mm for every 25mm

distance translated from a reference point. Hence assuming a knowledge of the way in which inaccuracies accumulate, the robot programmer must restrict the off-line programmed assembly positions so that they lie within the maximum allowable translation (200 mm in the case of the demonstration system: the derivation of this range will be described later) from a taught reference position.

the general equation of position error is given as
position error = 0.2% of the distance travelled (translated)

Clearly the accuracy of the system is not the same as the accuracy of the robot. Position errors also occur where component, fixture, and tooling tolerances lead to errors in the robot/workplace model. However, the use only of small movements from known taught reference points should also minimise these errors. However, this conclusion will only be true where errors can be predicted in probablistic terms, and unpredictable variations will not taken into consideration. Let us consider the nature of some of these probablistic occurences,

(a) the manufacturing tolerances in the drilling and plating process can be accommodated within one experiment which determines the variation in hole sizes resulting from the manufacturing process, provided that the process produces repeatable errors. If drift occurs in the drilling and plating process, then additional methodologies should be considered dependant on the nature of that drift.

(b) the manufacturing tolerance of the components can also be determined, provided that they can be predicted. If not, it may be necessary to restrict the type or suppliers of such components. Commonly the variation in pin size and hole size may be assumed

normally distributed with mean $\mu$ and standard deviation $\sigma$. However, the robot positioning accuracy should not be assumed to be normally distributed.

Since the standard deviation $\sigma$ of the whole population is unknown, the programmer of the assembly machine will have to estimate it from a small sample. It can be shown that if a sample of size n has a mean of X and standard deviation S then

(i) the best estimate of $\mu$ is X

(ii) the best estimate of $\sigma$ is $\sqrt{\frac{n}{(n-1)}}\ S$ [White et al, 1985]

where
n represents the size of the sample
X represents the sample mean
S represents the sample standard deviation
$\mu$ represents the mean value of the population
$\sigma$ represents the standard deviation of the population

It can also be shown that the statistic $t = \frac{(X-\mu)}{(\sigma/\sqrt{n})}$ follows the t-distribution with (n-1) degrees of freedom rather than a normal distribution. As the sample size n is large the t-distribution curve is close to the normal distribution curve. A sample size of 20 is large enough to give a close approximation of the normal distribution curve and therefore this sampling size is used in this error analysis.

From the readings of pin and hole diameters, a confidence interval analysis of where the average size value of holes and pins should lie can be carried out as shown in appendix D.2. The tolerance at a certain confidence level can then be determined e.g. at 99 percent confidence interval, the tolerance between the pin and hole is found

to be

0.36mm (lower tolerance) < tolerance < 0.40mm (upper tolerance).

In general, the robot accuracy tests show a mean accuracy of approximately 0.05mm over a translated distance of 25mm from a taught reference position (repeatability of 0.05mm was found as quoted by the robot manufacturer). With specific reference to the commanded translation from the reference location and the tolerances between pins and holes determined, the estimated robot accuracy can be used to determine the critical assembly range for achieving a successful insertion of the electronic components. For example, by experiment the critical range has been found to be bounded by a 200mm translation from the reference location. In these experiments involving the use of a specific type of component and the specific demonstration pcb, the robot accuracy is found to be about 0.40mm at approximately 200mm translation and about 0.36mm at approximately 175.0mm translation (see appendix D.2).

At 99% confidence interval, if the robot accuracy at a commanded translation from a taught reference frame is greater than 0.36mm (the lower assembly tolerance limit) then the robot accuracy is at its critical condition beyond which unsuccessful insertions are likely with the probability of unsuccessful insertions increasing as the translational distance from the taught location is increased. This means that any commanded translations within 175mm range (i.e. robot accuracy is smaller than the lower assembly tolerance limit) very high success rate for insertion would be expected. Any translation between 175mm and 200mm will correspond as grey area and beyond 200mm (upper

assembly tolerance limit) failure is likely. Clearly, real problems can be more complex than those analysed, but the results indicate a methodology for a quick assessment of possible success when robot assembly operations are programmed using off-line methods. From the experimental experience gained, with the particular programming and assembly arrangement created, Table 8-2 has been constructed to quantify the relative importance of system components in regard to their error contribution.

The accuracy of the robot, and gripper misalignment are considered to be the major sources of error which warrent improvement in the demonstrator system produced. The mounting of gripper on the robot flange could contribute between 0.1 to 1 degree orientational misalignment, the actual value being dependant on the skill of the programmer. This orientational misalignment can be magnified as the robot translates a long distance. Whereas the simulation system and the feeder contribute errors of medium magnitude. Other items listed in table 8-2 are considered to introduce relatively minor errors. However it may be that the accumulation of these error sources would cause failure in off-line programmed operations. Any accuracy improvements should be considered in relation to the cost involved, so that economic assessments can be made for a given application area.

### 8.4.3 Practical Problems and Considerations

It is important to highlight some of the limitations of the approach adopted by the author so that the above methodology can be placed in context.

| CATEGORIES | SYSTEM ELEMENTS | ESTIMATED ERROR RANGE |
|---|---|---|
| FORWARD PATH ERROR | REDBOARD pcb design system (depends on resolution chosen) | minor ~ 0.001 < E < 0.01mm |
| | NC drilling machine accuracy | minor ~ positioning accuracy 0.01mm<br>minor ~ drilling accuracy 0.05mm |
| | Robot simulator (GRASP) (resolution) | minor ~ 0.001 < E < 0.01mm (T)<br>medium ~ 0.02 < E < 0.06 degrees (R) |
| | Language post-processors (resolution) | minor ~ E < 0.0001mm (T)<br>minor ~ E < 0.0001 degree (R) |
| | Robot controller and arm accuracy | major ~ 0.2% of distance moved (T) |
| | Gripper misalignment | depends on skill and method (R) |
| | Feeder position error | medium ~ 0.05 < E < 0.1mm (T) |
| FEEDBACK PATH ERROR | Vision system | medium ~ 0.05 < E < 0.1mm (T) |
| PROCESS ERROR | Component size variation from nominal dimension | medium ~ 0.05 < E < 0.1mm (T) |
| | Leads location variations | major ~ 0.05 < E < 0.2mm (T) |

where

E = Error range

(T) = Translation

(R) = Rotation

Table 8-2 Relative significance of error sources

## (a) Restrictions Imposed by Available Feeding Mechanisms and Gripping Devices

Although the pcb used in the demonstrator system was supplied by a manufacturer, and thus represents a realistic assembly task, suitable gripping devices and feeding mechanisms for a wide range of electronic assembly tasks were not available. Therefore the experimentation was directed towards the assembly of electronic components for which feeding mechanisms were available and which do not present gripping problems with the available gripper and its attachments. Although experiments were directed towards the assembly of IC's of various types and sizes with the objective of illustrating the principles involved, other component types, or indeed similar components from alternative suppliers could present problems not encountered in this study.

## (b) Inconsistency of Component Body Dimension and Leads Locations on Component

One of the significant problem areas which must be faced when applying robots to pcb assembly is concerned with the inconsistencies in the component itself. Variation in component body dimension and locations of leads on component, can be quite substantial especially in odd form components [Cowan and Davies, 1986]. Ideally, components supplied by any manufacturer should be standardised regarding their size and shape, locations of leads and packaging. Suppliers are taking steps towards satifying this user standardisation requirement [Kochan, 1986].

## (c) An 'Inverted' Manufacture to Design Approach

In a typical electronic manufacturing environment the CAD model of the pcb design would be created before manufacturing. However, the experimental procedure followed in this project utilised an existing industrial pcb and therefore the pcb CAD model was created through measurement of parameters of that board. A conventional pcb NC drilling machine was used for this work and this will have resulted in small measurement errors (see appendix D.3). However, it is likely that tolerances caused by manufacturing process variations could be significantly more troublesome in accomplishing successful assembly.

## (d) Difficulty Introduced by the pcb Design System

The IC package sizes stored in the database of the REDBOARD system are not the exact sizes of the IC chips (details as shown in figure 8-2). This contributes modelling errors and subsequent gripping errors in the assembly, and leads to more difficulties than would be experienced in a factory situation, if methods of avoiding such errors were employed.

## (e) Numerical Accuracy of the Design and Simulation Systems

In general, if an item is required to perform several rotations in the simulation process, the algorithms used (for forward and inverse transformations) in the robot simulator will introduce numerical inaccuracy. The magnitude of this inaccuracy will vary from system to system. However, in this specific example, GRASP will introduce

cumulative orientation inaccuracy of between 0.02 and 0.06 degrees. The inaccuracy becomes worse if the number of rotations increases or if it involves more than one axis. This numerical inaccuracy is due to the inaccuracy accumulated over the forward and inverse kinematic transformation. Thus it would be advantageous to keep the number of rotations to a minimum (sufficient to perform the required simulation) so as to keep the error to a minimum.

## (f) Gripping Consideration

Another practical problem involved was to ensure that the IC chips were gripped at the midpoint (pick up points) along their Y axis. The solution adopted was to place the IC's at the lowest position on the feeder (i.e gravity fed for the particular feeders used) and then to teach the manipulator this position by moving the gripper to the position. The pick up positions are obtainable through a translation (half of the Y dimension of the IC chips concerned) along the Y axis of the IC's. A similar procedure could be followed for other feeder types. This location and orientation information associated with feeder positions must be feed back to the robot simulator so that IC's can be placed at their pick up points (in the model). The robot simulator uses information concerning these pick up points so that the gripper tool centre point can coincide with them. This will only be done once if the feeder position is fixed relative to the worksurface, and hence maintains a constant spatial relationship with the manipulator.

In a practical situation, there is uncertainty with regard to the

position and orientations of parts delivered by the feeding device and this can result in problems when the robot picks up the parts at the location specified. With the demonstrator workcell, occasional problems of this type have proved to be unavoidable. If parts are to be delivered at more precise positions and orientations, then alternative specially designed feeding devices would be required. This raises issues of cost against precision, and flexibility against precision.

## (g) pcb location consideration

Additional position errors can occur with regard to the feeding of bare or partially populated pcbs into the assembly system. In the demonstrator system, the pcb is located on a pallet which is stationed at some point on the conveyor. In such situations position errors can occur between the pallet and the conveyor. This implies that a positioning mechanism is required to ensure that the coordinate reference frame of any pallet or pcb will be located, within predictable location errors, at some conveyor position and orientation each time an assembly is required i.e. it is necessary to overcome unpredictable errors. Again issues such as cost and flexibility are raised when specifying 'dedicated' positioning mechanisms.

Alternatively, this could be achieved by utilising a vision system to determine the position and orientation of the pcb's coordinate reference frame so that appropriate compensation for errors can be made. Although the capital cost would be increased, the vision system will provide a more flexible approach than that of using specialised fixtures and in certain situations economic benefit could accrue.

## (h) calibration consideration when using vision

In situations where a vision system can be used economically, vision prototype training errors can occur where the centroid of the prototype object does not correspond to that determined by the vision system. When the vision system is used to identify the location of a part in the working environment, factors such as variations in the ambient lighting conditions can lead to measurement errors. The accuracy of vision systems (including illumination equipment) will vary from system to system (cost being a key factor).

Assuming the use of a vision system (a situation investigated through use of the demonstrator system) the location and orientation of each pcb board frame will usually be measured, prior to component assembly, to overcome both predictable and unpredictable position errors. The practice of using fiducial marks on pcb's is a common one, being used in pcb manufacture at various automatic machines including insertion, onsertion and testing equipment. Fiducial marks are created on the pcb (see figure 8-11 for detail) which are used by the vision system to determine the coordinate reference frame of the board. The required position of components on the pcb relative to the coordinate reference frame maintain fixed spatial relationships. However, due to the robot accuracy alone, these component positions may not be attainable if they are beyond the critical position.

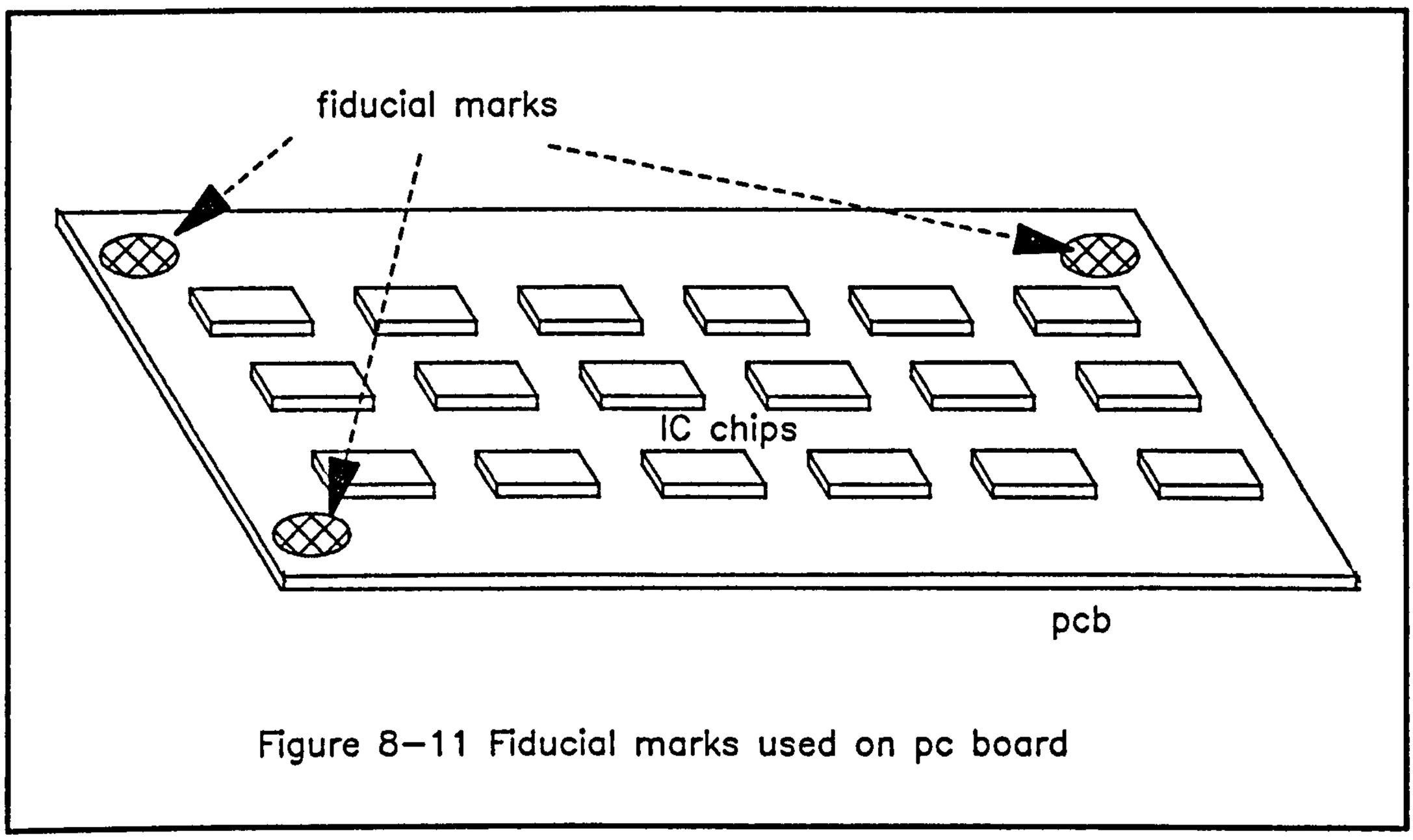



Figure 8–11 Fiducial marks used on pc board

These locations and orientations can be used to update the GRASP model if the set up is not going to be changed in the short term and the product belongs to a family series. Otherwise they may be simply stored at the robot controller. This implies that only positions and orientations of those entities which are common to the series of family products are required to be updated (in the simulation model). Other entities whose positions are subject to variations need not be updated. In the demonstrator system, only the pcb frame is variable and so it need not be updated in the simulation model.

The GRASP model update can be accomplished through the calibration software modules described in chapter seven. As the position and orientation of the pcb varies, it is reasonable to store the frame at the robot controller and it is not necessary to update the CAD model.

## 8.5 General Conclusions

The experimental observations, and specific conclusions made with regard to error sources, have illustrated that for the demonstrator system constructed the major error source is the robot manipulator itself. The remaining error sources identified can individually be considered to be relatively insignificant although they can cumulate to result in unsuccessful assembly operations.

In the experiments carried out, the robot positioning error was found to cumulate with an increase of approximately 0.05mm for every 25mm (i.e. 0.2%) distance travelled from a reference position. It should be stated that the experiments performed, in establishing the robot accuracy, cannot be considered to be precise or comprehensive but serve to illustrate the scale of the errors involved. Clearly more comprehensive testing could have been carried out (e.g. through using laser tracking) but the difficulties involved in carrying out such testing did not permit such a study.

One way of improving the robot accuracy and hence facilitating the use of off-line programming in a wider range of application areas is to manufacture and assemble the robot arm in such a way that very tight tolerances are met. However, in general this will be too expensive and even then certain error sources will remain. Clearly also the use of high "stiffness" cartesian manipulator systems will result in improved robot accuracy but may lead to to lack of dexterity and poor cycle times. Arguably therefore the most effective and possibly least costly method is to employ software based calibration procedures to assign a "signature"

to each robot.

Despite errors, the use of off-line robot programming in pcb assembly is theoretically possible and has been practically demonstrated for a SCARA arm provided that "a limited range of translated movement from the pcb reference frame" is applied as an operating constraint. It is apparent that further fiducial marks should be placed on the pcb so that the effective maximum translated distance range for pcb assembly can be increased. This would imply that a different set of fiducial marks should be used to avoid confusion in any visual recognition process used to establish different reference frame. This implies additional complication in the simulation and modelling procedure and could involve automatic updating of a stored model or could require a second stage, more complex, manual teaching phase. Furthermore, the gripper used was not ideal as it does not incorporate sensory feed back so that any variation in the off-line programmed pick up positions (due to robot position errors and discrepancy in the position of the electronic components delivered) may cause difficulty or failure in certain assembly tasks. This situation could have been improved by using sensory feed back (such as force sensing) and hence achieving self adjustment to reduce the magnitude of the errors involved.

The successful application of any off-line robot programming system will be dependant upon the relative spatial arrangements of workplace devices. Hence rigidly fixed arrangements should be employed where possible and any variations should at least be predictable. In such a situation any errors in the nominal positions of the workplace entities can be accounted for by updating the 'model' through following calibration

procedures. After this initialisation process, if the physical arrangement is rigidly fixed, then accurate spatial relationships can be maintained. If entities are moved to alternative locations which are outside acceptable tolerance limits, the robot programs generated off-line will not result in successful assembly processes and calibration procedures must be re-initiated.

Through providing integration software, shared electronic access of information can be achieved, such as access to information stored in CAD product design and robot simulator databases. This study has shown that product design data, created using a conventional pcb CAD system can be used to assist in model building. This can significantly simplify the off-line programming of robots so that lead time benefits can accrue, a fact of potentially great importance in certain industrial sectors where reduced "time to market" is vital. The information stored in a centralised or common database could utimately be distributed through the use of appropriate database management tools including data modelling and query language facilities. Information characterising the geometry of the pcb, geometry of components and their locations and orientations with respect to the frame of the board, plus the generated sequential and spatial relationships defining robot motions can be downloaded to the robot controller. In such a situation, the only unknown variable is the location and orientation of the frame of the pcb relative to the robot, for which teach methods can be employed. The teaching of the required pcb location and orientation unknowns would be a relatively simple task. In certain situations it may be possible to bypass the need for using a robot simulator, but the robot simulator can provide program verification before downloading to the robot controller. This is a rational,

practical, and implementable approach with current technology. Clearly as the need for such integrated systems is realised, future approaches may involve the redesign of the CAD, simulator and robot controller tools.

To enable the widespread use and acceptance of off-line programming methodologies, it is important that a 'neutral' format be agreed to enable information transfer between the components of the generic off-line programming system. Using contemporary practice this may comprise a number of sub-system elements (e.g. CAD, simulator/modeller, robot) which will be implemented by different vendors, using different computer hardware, operating systems and development languages. In such a situation the need for agreed neutral formats between CAD and simulator/modeller subsystems is apparent whereby the subsystem suppliers can write to an agreed information exchange specification. Similarly, an agreed information exchange specification is required between the modeller simulator and the robot controller.

# CHAPTER NINE

# CONCLUSIONS AND RECOMMENDATIONS

## 9.0 Introduction

In this chapter, the contribution to knowledge is first discussed, followed by the general implications and recommendations for future work.

## 9.1 Contribution to Knowledge

This thesis has illustrated how a robot can be programmed off-line based on a robot simulator. This research study has investigated the possibility of enhancing a robot simulator through 'user friendliness', calibration and integration. These form a package which should be considered for enhancement so as to achieve advantages of overall system improvement rather than detailed improvement in a single aspect. Therefore the theme of this research study has been the investigation of the advancement of this package.

Aspects of integration have been implemented in two phases. The first phase integrated a CAD pcb layout design system with a robot simulator through the capture of product assembly data. The integration of the robot simulator with the actual robot system has been implemented through a post-processing facility. In each phase of integration, neutral data formats were conceived by the author. Furthermore, an insertion sequencer was conceived to implement five commonly used rules when generating pseudo process plans. Pseudo process plan generation, based on the component type and minimum distance of travel from the current position, is the most commonly used in pcb assembly.

CAD/CAM facilities should be integrated with robot modelling and programming systems so that existing product design data captured in the CAD database can be used to speed up the modelling and programming process, improve productivity, reduce the chance of error etc. The practical experiment described in chapter eight illustrates that product design data can be used efficiently for modelling and robot task programming with the task sequence being determined by the sequence planner. Although the study findings could necessarily not be exhaustive, they represent an important contribution to the literature, pointing to areas of future study which ultimately could significantly reduce the "time to market" of new products.

Thus integration of a CAD pcb layout design system with a robot simulator has been demonstrated based on knowledge of parameterised model building, parameterised task programming and post-processing. In the integration investigation, the demonstrator system included a robot without on-line sensing capability (sensing only being used during initialisation procedures which could have been implemented using other methods). Such a flexible pcb assembly system, programmed off-line through the integration of a CAD pcb design system with a robot simulator, could be cost effectively employed in certain manufacturing situations. Integration problems were identified and classified. It is essential to understand that contemporary CAD pcb design tools do not provide information in an appropriate form for robot simulators, particularly as they lack 3D information for modelling and insertion sequence planning. This is understandable as design systems were intended for pcb NC drilling, ATE and ACI but not for integration with other systems. The

lack of a standard data format presents further problems such as post-processing efforts for different systems (as discussed in chapter five), and the author has derived a neutral data format approach as a solution.

In addition to integration problems, there are sources of errors which effect positional accuracy of off-line generated programs. These sources of error have been analysed through the demonstrator system. Understanding of the problems encountered enabled solutions to be generated. There are two major sources of error which require attention:

(i) the robot system accuracy (including robot controller and arm)

and

(ii) process error including accuracy of components delivered at the reference pick up positions and component size variations from nominal sizes (including body size variations of components and variations of pin positions on the component body) leading to additional variations in delivered positions.

Error compensation is therefore required in off-line programming systems. Consideration of the robot arm inaccuracy shows that there are two possible ways of improving the demonstrator system. One solution is to design and manufacture a robot to tight tolerances whilst another is to use software calibration for each individual robot arm. This raises issues of cost against precision and flexibility and the former approach

is considered too costly whilst the latter can provide the flexibility required. The variations of component size from nominal size should be tightened up so that the process error can be minimised to reduce uncertainty.

In addition to process error, different electronic component manufacturers supply functionally identical components in different sizes, shapes and packaging. User driven initatives may be beneficial in forcing suppliers to standardise on component size, shape and packaging. The electronic assembly industry has initated steps towards the standardisation in these aspects. The alternative approach of using sensory feedback elements such as vision and force sensing for correcting parameters from the nominal values is costly and inconvenient.

The post-processing module for integrating the robot simulator and the actual robot system illustrates the practical problems together with the author derived neutral data format. Through this post-processing study, it is concluded that whenever possible post-processing facilities for off-line programming should be accomplished in two stages, one responsible for generating neutral or standard data format with a second for post-processing this format into specific target robot languages. Furthermore, post-processing facilities should preferably be included in the robot simulation system as an "off-the-shelf" optional function. Since suppliers of robot simulation systems have the appropriate access to the system data structures, the incorporation of a post-processor in a simulator represents a more efficient use of data.

The lack of user friendliness of robot simulators has led to the investigation of enhancement through parameterised model building and parameterised simulation task generation (off-line robot programs). In demonstrating possible solutions, software modules have been implemented for certain classes of robotic entities and robot task applications. This study has concluded that parameterised model building and task generation would be more appropriately implemented as "macros" within the robot simulator. This is mainly because limitations on access to the data structures of the CAD model restricts the scope for further enhancement.

It is also concluded that modelling of common robotic workplace entities can be accomplished through parameterised solid modelling facilities. Similarly, it is also indicated that robot operations can be programmed through parameterised task languages. A parameterised task language has been proposed and discussed in chapter six. A full scale parameterised task language could be used to program the robot without reference to the robot simulator, but verification and modification would still require the simulator. This implies that it is more appropriate for parameterised task languages to be implemented as application macros within the robot simulation system. This study also demonstartes further possibilities for integration of these parameterised facilities with expert systems which have access to comprehensive knowledge bases.

It has been shown that off-line programming is not a viable proposition without the use of calibration techniques. Calibration can be carried out in three phases. Firstly, calibration of the robot arm is carried

out to eliminate or reduce discrepancies between the actual robot arm kinematics and the model derived kinematic equations based on nominal robot parameters. This robot arm calibration can be further classified into robot joint and entire arm calibration. Secondly, it is necessary to calibrate the simulation model against the real robotic workplace. Solutions have been derived and illustrated for calibrating and updating the simulation model which encompass the discrepancies between an idealised simulation workplace model and the actual robot workplace. To deal with calibration of a simulation model it is necessary to consider the appropriate options implied by the batch size. It is reasonable to conclude that if a product batch belongs to a product family then updating the simulation model is essential so that other off-line programs can be generated based upon the updated model. In other situations there is no need for long term retention of the calibrated information and it should therefore be kept locally at the robot controller.

With large batches, conditions might change during manufacture or assembly and in this case calibration is used to update all teach reference points and store them at the robot controller until further changes in the workshop arrangement. The need to update the original simulation model becomes less important, especially where the batch size is very small. However, any changes in the workplace spatial relationships should be updated in the simulation model so as to maintain a representative model.

The final phase of calibration concerns on-line dynamic calibration which has not formed part of this research study.

If off-line rather than on-line methods are used for programming a robot, a simulation model of the robotic workplace is required before robot tasks can be simulated, verified and evaluated. This means that for a simple task the time taken to programme a robot through off-line method; could be longer than that needed for on-line programming methods. In addition, off-line robot programming methods are associated with accuracy problems and therefore fine tuning of the robot program is required. However, in the batch manufacturing of large product families, off-line programming methods are becoming more efficient as the size of the product family increases. This is because the simulation model of the robotic workplace is virtually the same for most members of the family and therefore only a rearrangement of workplace elements and the new product design are required. Even when batch manufacturing a product which does not belong to a family, off-line programming methods still reap the advantage of not disrupting actual production systems and therefore higher utilisation of expensive equipments can be achieved. Furthermore, integration with product design data improves lead times. In some circumstances, robot applications which include synchronisation of robotic devices are difficult to programme using on-line methods. Due to the difficulties and cost implications of using robot simulation tools, off-line robot programming will be more easily justified in circumstances where complex robot tasks are required and/or where batch manufacturing of large product families are involved. Confidence in the use of off-line robot programming techniques can be much improved when

satisfactory solutions are provided to overcome accuracy problems.

## 9.2 General Implications and Recommendations

It is essential that facilities for processing sensory information and dealing with synchronisation and error conditions should be sufficiently sophisticated to perform the required task in an efficient manner. The allowable sophistication of any off-line programs generated will be limited by the capabilities of the robot simulator, and thus simulation of sensor conditions and capabilities for synchronisation and exception handling should be provided. Currently available simulators can only model very simple sensory conditions. It can be seen that off-line programming by graphical computer methods has considerable potential advantage, which must be tempered by a consideration of some fundamental limitations of existing systems. In certain circumstances, the use of off-line programming can be difficult due to the lack of good sensory input and output. A compromise hybrid approach should be adopted where off-line programming is used in conjunction with on-line programming such that robot task programs can be prepared with higher efficiency and/or functionality level.

A low cost flexible robot assembly system without sensing has major disadvantages in coping with problems of inaccuracy. Furthermore, the assembly process concerns not only the robot movement, but also the identification and verification of components and sub-assemblies. However, if the robot system involves the use of a gripper with sensing capability then any variations in component delivered by the feeding

mechanisms can be catered for. Furthermore, the use of sensing devices (such as bar codes and vision systems for identification) can ensure that the correct components are assembled satisfactorily (exception handling and problem tracing in the workshop). Laser stripers and vision systems for checking variations in component size and variations of locations of pins on components can be used so that corrective action can be invoked to ensure an improved success rate in assembly. The use of extra sensing and correction means improved flexibility and accuracy can be achieved at the expense of longer cycle time and high capital cost.

There are suggestions for further considerations and these are separated into functional areas.

(a) Standards

There is a need for a standard robot data format, with each simulator system having its own pair of processors to transfer data to and from this neutral format. In such a way any one robot simulator can be used to program multiple robots.

The development of product design data exchange based on the principle of a standard neutral data format (such as IGES, PDES/STEP etc.) can be applied in the field of 3D robot simulators. Data originally restricted to 2D drawing data information has more recently been enhanced to deal with more comprehensive product descriptions including 3D solid modelling. Within the foreseeable future, it is possible to envisage a standard neutral data format for 3D robot simulators. This would include

robot kinematic/dynamic modelling descriptions and the program format for robot task descriptions.

## (b) Robot Simulation Systems

A robot simulation software designer should not be bound to one single geometric modelling technique. A solid modelling system should be capable of utilising more than one modelling technique and thus facilitate wider scope for modelling and off-line programming applications. For example, surface modelling techniques allow spherical or continuous curved paths to be programmed by reference to the surface normal. Applications include arc welding and cutting, surface grinding, glass cutting, mould making, and drilling and riveting for aircraft panels. Robot simulation involving improved geometric modelling techniques could enable better accuracy in the simulation model and hence improved off-line robot programming.

Parametric design and task programming languages should be a feature of robot simulation systems so that generic robot simulators can be used conveniently for specific applications. Further advantages can be gained from the use of expert systems to access robot databases, tooling and other robotic peripheries databases. This would be useful for selecting the appropriate robot, tooling and attendant robotic equipment for the right application and at the right price etc. In the computer assisted task generation, it is preferred that the expert system be used for selection, computer assisted model building and task program generation. With the use of artificial intelligence or geometric reasoning, collision

free robot paths could be planned. The popularity of the parametric design and programming language is based upon the success of standardisation achieved in design data exchange between different robot simulators.

## (c) Calibration and Integration

Generic robot calibration procedures have yet to evolve in an internationally accepted sense, resulting from the complexity involved in defining a comprehensive scheme meaningful to the divergent areas of application of robot systems.

Although various attempts (including dynamic robot simulation and dynamic error correction at the robot controller) have been made to overcome or at least to reduce dynamic errors, the result of the dynamic simulation cannot closely predict the real dynamic error and hence correction. Unfortunately, the collection of dynamic data is a difficult task. However, integration of the robot controller with the simulation sub-system (part of a robot simulator), provides a channel towards incorporating robot modelling and error correction at the robot controller. The installation of a robot simulation package into a robot controller can potentially offer a more sophisticated method of predicting and correcting dynamic errors with flexibility. Theoretically, the robot dynamic behaviour can be monitored and models updated during each robot task performance. Based upon this updated dynamic robot model, new dynamic errors can be predicted and corrected algorithmically. The discrepancies between the simulation model and the

real world can easily be assessed within the same robot controller and the updating procedure may be simplified. This implies that further research is required before this facility can become a reality. In the foreseeable future this type of robot simulation/controller is likely to be commercially available.

The integration of robot simulators (off-line robot programming systems) with expert systems will facilitate better manufacturing procedure and parameter selection for various operations (e.g. welding voltage, current and time will be provided for welding; power required, cutting speed, feed rate and depth of cut for a certain material for turning and milling etc). In addition, process scheduling can be achieved so that the monitoring of the manufacturing or assembly process can be accomplished for each work cell. With the advancement and application of artificial intelligence in robotics, further improvements in CIM is desirable and achievable.

# REFERENCES

Adler, A. "TDL A Task Description Language for Programming Automated Robotic Workcells", IEEE International Conference on Systems, Man and Cybernetics, Atlanta, GA, USA. 14-17 Oct 1986, Vol. 1, pp65-68

Albus, J.S. "Robotics", Robotics and Artificial Intelligence (edited by Brady, M. et al), NATO ASI series Vol. F11, Springer-Verlag, 1984, pp65-93

Allen, D.K. "Architecture for Computer-Integrated Manufacturing", ANNALS of the CIRP, 1987, Vol. 36, No. 1, pp351-354

Ambler, A.P. "RAPT : An Object Level Robot Programming Language", IEEE Colloq. on languages for industrial robots, 1982, pp4/1-4/5

Ambler, A.P.; Popplestone, R.J. and Kempf, K.G. "An Experiment in the Off-line Programming of Robots", Proceedings of the 12th International Symposium on Industrial Robots, 1982, pp491-504

Ambler, A.P. "Languages for Programming Robots", Robotics and Artificial Intelligence (edited by Brady, M. et al), NATO ASI series Vol. F11, Springer-Verlag, 1984, pp219-227

Andrews, S. and Cliffe, R.W. "The Development of a Robot Post-processor: A Tool for Off-line Robot Programming", Proceedings of the 9th Annual British Robot Association Conference, Stratforn-Upon-Avon, 13-14 May, 1986, pp193-204

Appleton, E.; Fallside, F. and Richards, R.J. "Progress on an Automatic Assembly System Linked to a CAD Database", ACME Research Conference Proceedings, Nottingham University, 1988, Science and Engineering Research Council

Arai, T. and Matsumoto, A. "Intermediate Language for Robot Controller", Proceedings of the 4th International Conference on Assembly Automation (edited by Makino, H.), Tokyo, Japan, 11-13 Oct 1983, pp55-66

Arai, T.; Takashima, S.; Hirai, S. and Sata, T. "Standardization of Robot Software in Japan", Proceedings of the 15th International Symposium on Industrial Robots, 1985, pp995-1001

ASEA Limited, 48 Leicester Square, London, England "ASEA Off-line Programming System" CK09-1222E, 1986

Astrom, K.J. and Eykhoff, P. "System Identification - A Survey", Automatica, 1971, Vol.7, pp123-162

Ayres, R.U.; Miller, S.M.; Just, J.; King, K.; Osheroff, M.; Berke, G.; Spidaliere, P. and Ngoc, T. "Recent Developments in Robotics and Flexible Manufacturing Systems", Robotics and Flexible Manufacturing Technologies, Assessment, Impacts and Forecast, Noyes Publications, USA, 1985

Azadivar, F. "The Effect of Joint Position Errors of Industrial Robots on Their Performance in Manufacturing Operations", IEEE Journal of Robotics and Automation, April 1987, Vol. RA-3, No. 2, pp109-114

Baird, H.S. and Lurie, M. "Precise Robotic Assembly Using Vision in Hand", Rovisec 83, Cambridge, Mass. 1983, pp660-666

Benhabib, B.; Fenton, R.G.; and Goldenberg, A.A. "Computer-Aided Joint Error Analysis of Robots", IEEE Journal of Robotics and Automation, Aug 1987, Vol. RA-3, No. 4, pp317-322

Birk, J.R. "A Comparison for Robots to Orient and Position Hand-held Workpieces", IEEE Transactions on Systems, Man and Cybernetics, Oct 1976, Vol. SMC-6, No. 10, pp665-671

Bonner, S. and Shin, K. "A Comparative Study of Robot Languages", IEEE Computer, Dec 1982, pp87-97

Bonney, M.C. "Off-line Programming - GRASP Robot Simulation System", Off-line Robot Programming (edited by Storr, A. and McWaters, J.F.), North-Holland, 1987, pp171-179

Bonney, M.C.; Edwards, P.J.; Gleave, J.A.; Green, J.L.; Marshall, R.J. and Yong, Y.F. "The Simulation of Industrial Robot Systems", OMEGA, Int. J. of Management Science, 1984, Vol. 12, No. 3, pp273-281

Boren, R.B. "Graphics Simulation and Programming for Robotic Workcell Design", Robotics Age, Aug 1985, pp30-33

Braid, I.C. "The Synthesis of Solids Bounded by Many Faces", Communications, April 1975, pp209-216

BYG System, Highfield Science Park, Nottingham "Robot Simulation Using GRASP", 1987

BYG System, Highfield Science Park, Nottingham "VAL II Postprocessors Manual", 1988

Carter, S. "Off-line Robot Programming: the State of the Art", The Industrial Robot, 1987, Vol. 14, No. 4, pp213-215

Case, K.; Porter, J.M. and Bonney, M.C. "SAMMIE: A Computer Aided Design Tool for Ergonomics", Paper presented to Human Factors Society, Dayton, Ohio, 1986

Chan, S.F.; Weston, R.H. and Case, K. "Robot Simulation and Off-line Programming", Computer-Aided Engineering Journal, Aug 1988, Vol. 5, No. 4, pp157-162

Chawla, S.D.; and Gruver, W.A. "Off-line Robot Programming with an Integrated Graphics Subsystem", International Conference Proceedings of ASME Computers in Engineering, Aug 1984, pp111-114

Chen, J. and Chao, L.M. "Positioning Error Analysis for Robot Manipulators with All Rotary Joints", IEEE Journal of Robotics and Automation, Sept-Dec 1987, Vol. RA-3, No. 6, pp539-545

Cincinnati Milacron (UK) Co., Kingsbury Road, Birmingham, England. Cincinnati Milacron Documentation, "ROPS" A362, 1985

Computing Equipment, "Seeing is Believing", October 1985, p39

Cook, C.D. and Vu-Dinh, T. "A New General Algorithm for Describing Manipulator Kinematics", Mechanical Engineering Transactions, 1985, pp169-174

Cowan, D. and Davies, D. "Automatic System Copes with Variable pcb Production", Assembly Automation, Feb. 1986, Vol. 6, No. 1, pp19-22

Craig, J.J. "Anatomy of an Off-line Programming System", Robotics Today, Feb 1985, pp45-47

Cronk, R.N.; Callahan, P.H. and Bernstein, L. "Rule-Based Expert Systems for Network Management and Operations : An Introduction", IEEE Network, September 1988, Vol. 2, No. 5, pp7-21

Crookall, J.R. "Education for CIM", ANNALS of the CIRP, 1987, Vol. 36, No. 2, pp479-494

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, Tokyo, Japan, 1987, pp316-323

Denavit, J. and Hartenberg, R.S. "A Kinematic Notation for Lower-Pair Mechanisms based on Matrices", Journal of Applied Mechanics, June 1955, pp215-221

Derby, S. "Off-line Programming of Two Industrial Robots", Robots 8 Conference Proceedings, Detroit, MI, USA 4-7 June 1984a, Vol. 2, pp20/65-20/76

Derby, S. "GRASP from Computer Aided Robot Design to Off-line Programming", Robotics Age, Feb 1984b, pp11-12

Dooner, M. "Robotics Software and CADCAM", Computer-Aided Engineering Journal, Dec 1984, pp217-220

Dooner, M. "Techniques for Designing Production Systems", Computer-Aided Engineering Journal, Aug 1987, pp157-159

Driels, M.R. and Pathre, U.S. "Generalized Joint Model for Robot Manipulator Kinematic Calibration and Compensation", Journal of Robotic Systems, Feb 1987, Vol. 4, No. 1, pp77-114,

Dudko, E.A.; Naidek, A.V.; and Yampolskij, L.S. "A CAD/CAM Sub-system for Selecting a Suitable Industrial Robot", Soviet Engineering Research, 1984, Vol. 4, No. 12, pp45-48

Duelen, G. and Bernhardt, R. "Demands on Off-line Programming Systems for Industrial Robots", Software for Discrete Manufacturing, (edited by Crestin, J.P. and McWaters, J.F.) North-Holland, 1986, pp467-480

Durham, T. "Off-line Programming : Key to CAD/CAM Methods", Computing, 23 June 1985, pp26-27

ECSL menu 1986, CLE.COM Limited, King's Heath, Birmingham.

Edwards, P.R. and Howarth, M. "Computer Integrated Crystal Glass Pattern Cutting", ACME Research Conference Proceedings, Nottingham University, 1988, Science and Engineering Research Council

El-Zorkany, H.I. "Automatic location correction in off-line programming of industrial robots", Proceedings of the 14th International Symposium on Industrial Robots and the 7th International Conference on the Industrial Robot Technology, Gothernburg, Sweden, 2-4 Oct 1984, pp335-346

El-Zorkany, H.I. "Robot Programming", INFOR, Nov 1985, Vol. 23, No. 4, pp430-446

Eversheim, W.; Weck, M.; Scholing, H. and Zuhlke, D. "Off-line Programming of Numerically Controlled Industrial Robots Using the ROBEX-Programming Systems", ANNALS of the CIRP, 1981, Vol. 30, No. 1, pp419-422

Eykhoff, P. "System Identification; Parameter and Estimation", John Wiley & sons, 1974

Featherstone, R. "Robot Dynamics Algorithms", Kluwer Academic Publishers, 1987

Fernandez, K. "The Use of Computer Graphics Simulation in the Development of Robotic Systems", ACTA Astronaut (UK), Jan 1988, Vol. 17, No. 1, pp115-122

Forestier, P. "The CATIA Integrated Geometry Modeller", Computer Graphics, Proceedings of the International Conference, London. 1985, pp381-391

Frederikson, L.B. "Robotics in a Spray-up Process", Proceedings of the 14th International Symposium on Industrial Robots, Gothenburg, 1984, pp297-303

Gettelman, K. "Off-line Programming Comes to Robots", Modern Machine Shop, Nov 1985, pp60-67

George, L. and Mital, A. "Consequences od Robotization", Proceedings of the IXth International Conference of Production Research, Cincinnati, Ohio, USA., Aug 1987,

Gini, M. "The Future of Robot Programming", Robotica, 1987, Vol. 5, pp235-246

Gini, G. and Gini, M. "Dealing with World Model Based Languages", ACM Transactions on Programming Languages, April 1985, Vol. 7, pp334-347

Gini, G.; Gin, M.; Cividini, M. and Villa, G. "Programming of Robot Systems", Computer-Aided Design and Manufacturing Methods and Tools Second, Revised and Enlarged Edition, (Edited by Rembold, U. and Dillmann, R.) Springer-Verlag, 1987, pp235-278

GMF Robotics Corp., Troy, MI, USA., "KAREL Operating Manual", 1986

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 Sept 1985

Goldenberg, A.A. and Lawrence, D.L. "A Generalized Solution to the Inverse Kinematics of Robotic Manipulators", Journal of Dynamic Systems, Measurement, and Control March 1985, Vol. 107, pp103-106

Gondert, S. "Off-line Programming Increases Robot Productivity", Design News, 26 Mar 1984, Vol. 40, pp60-66

Groover, M.P. "Automation, Production Systems, and Computer-Aided Manufacturing", Prentice-Hall, 1980, pp261-280

Groover, M.P "Automation, Production Systems, and Computer-Integrated Manufacturing", Prentice-Hall, 1987

Groover, M.P.; Weiss, M.; Nagel, R.N. and Odrey, N.G. "Industrial Robotics, Technology, Programming, and Applications", McGraw-Hill Book Company, 1986

Grossman, D. "AML as a Plant Floor Language", Robotics and Computer-Integrated Manufacturing 1985, Vol. 2, No. 3/4, pp215-217

Grunewald, P. "Car Body Painting With the Spine Spray System", Proceedings of the 14th International Symposium on Industrial Robots, Gothenburg, 1984, pp663-641

Gruver, W.A.; Soroka, B.I.; Craig, J.J. and Turner, T.L. "Evaluation of Commercially Available Robot Programming Languages", Proceedings of the 13th International Symposium on Industrial Robots, 1983, pp12/58-12/68

Gupta, K.C. "Kinematic Analysis of Manipulators Using the Zero Reference Position Description", International Journal of Robotics Research, Summer 1986, Vol. 5, No. 2, pp5-13

Haffenden, A. "Off-line Robot Programming", System International (G.B.), Dec 1984, Vol. 12, No. 12, pp21-23

Halme, A., Heikkila, T. and Torvikoski, T. "A Task Level Control and Simulation System for Interactive Robotics", Theory of Robots, 1987, pp289-296

Harrison, J.P. and Mahajan, R. "The IGRIP Approach to Off-line Programming and Workcell Design", Robotics Today, Aug 1986, pp25-26

Hauke, W. "Stromungstechnische Untersuchungen an Dusen fur das Druckluftstrahlen", Machinenbautechnik 31, 1982, No. 2, pp87-90

Hayes-Roth, F.; Waterman, D.; and Lenat, D. "Building Expert Systems", Addison-Wesley, Reading, Mass., 1983

Hemami, H., Jaswa, V.C. and 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 if Illinois, Oct 1975, pp124-140

Ho, C.Y. "Study of Precision and Calibration for IBM RS-1 Robot System", Assembly Automation, Nov. 1982, pp198-201

Hocken, R. and Morris, G. "An Overview of Off-line Robot Programming Systems", ANNALS of the CIRP, 1986, Vol. 35, No. 2, pp495-503

Hoermann, K. "GRIPS - A Robot Action Planner for Automatic Part Assembly", SIRI Symposium on Industrial Robots, Milan, Italy, March 1988

Hollingham, J. "The MAP Report", IFS publications Ltd 1986

Hornick, M.L. and Ravani, B. "Computer-aided Off-line Planning and Programming of Robot Motion", The International Journal of Robotics Research, Winter 1986, Vol. 4, No. 4, pp18-31

Howe, J.A.M. and Fothergill, A.P. "Solid Modelling, Sensors, and the Effects of Uncertainty on the Programming of Robots", ACME Research Conference Proceedings, Nottingham University, 1988, Science and Engineering Research Council

Howie, P. "Graphic Simulation for Off-line Robot Programming", Robotics Today, 1984, Vol. 6, Part 1, pp63-66

Howie, P.V. and Williams, K.A. "Off-line Programming to the Factory Floor", Conference Proceedings of Synergy 84, Chicago, IL, USA, 13-15 NOV 1984, pp206-211

ICAM Project, "Computer Program Development Specification (DS) for ICAM Integrated Support System (IISS) Configuration Item : Precomputer", Prepared by Control Data Corporation and D. Appleton Company, USA, Dec 1983

ICL, Kidsgrove, England personal communication with Mr I. Hunt, 1988

Imam, I.; Sweet, L.M.; Davis, J.E.; Good, M. and Strobel, K. "Simulation and Display of Dynamic Path Errors for Robot Motion Off-line Programming", Robots 8 Conference Proceedings, Detroit, MI, USA, 4-7 June 1984, Vol. 1, pp4-28/4-44

Industrial Robot, "Robotic Programming", Sept 1982, Vol. 9, No. 9, pp182

Industrial robot, "Cambridge Control Packages its Dynamic Approach", Feb. 1987, Vol 14, No. 2, pp93-94

Intergraph (Great Britain) Limited, Norman House, Heritage Gate, Derby, England. "Robot Programming - Preliminary", DMD 020B0, 1985.

ISO/DP 8373, "Manipulating Industrial Robots - Vocabulary", ISO Draft Proposal ISO/DP 8373, ISO/TC 184/SC 2, April, 1986

ISO/DP 9283, "Pose Accuracy and Repeatability Characteristics", UK's Recommendation for Revision of Sections 6.1 and 6.2 of ISO/DP 9283, 1986

Jacobs, M.P. "Off-line Robot Programming: A Current Practical Approach", Robots 8 Conference Proceedings, Detroit, MI, USA, 4-7 JUNE 1984, Vol. 1, pp4/1-4/11

Jeyachandra, M.; Rosenshine, M. and Soyster, A.L. "Analysis of Robot Positioning Error", International Journal of Production Research, 1986, Vol. 25, No. 5, pp1159-1169

Jones, A.T. and McLean, C.R. "A Proposed Hierarchical Control Model for Automating Systems", Journal of Manufacturing Systems, 1986, Vol. 19, pp15-25,

Kacala, J. "Robot Programming Goes Off-line", Machine Design, Nov. 1985, pp89-92

Khosla, P.K. and Neuman, C.P. "Computational Requirements of Customerized Newton-Euler Algorithm", Journal of Robotic System (USA), 1985, Vol. 2, No. 3, pp309-327

Klein, A. "Off-line Programming of Painting Robots Using Colour Graphics Technique", Off-line Programming of Industrial Robots, IFIP Conference, Stuttgart, 1986, pp139-151

Klein, A. "CAD-based Off-line Proogramming of Painting Robots", Robotica, 1987, Vol. 5, pp267-271

Knight, J.A.G.; Edwards, P.R.; and Taylor, J. "To Develop a Manufacturing System for the Production of the Decorative Patterns on Crystal Glassware", ACME Research Conference Proceedings, Salford, 1986, Science and Engineering Research Council

Kochan, A. "Telecom Technology Brings pcb Problems for Plessey", Assembly Automation, Nov. 1986, Vol. 6, No. 4, pp182-185

Kochan, A. "Advanced Robotics : Towards a Third Generation", The Industrial Robot, 1987, Vol. 14, No. 4, pp229-230

Kretch, S.J. "Robotic Animation", Mechanical Engineering, Aug 1982, pp32-35

Kusiak, A. and Heragu, S.S "Computer Integrated Manufacturing: A Structural Perspective", IEEE Network, May 1988, Vol. 2, No. 3, pp14-21

Kuvin, B.F. "Off-line Programming Keeps Robots Working", Welding Design and Fabrication, Nov 1985, Vol. 58, pp34-39

Lambourne, E.B. "Towards Integration of Computer-Aided Design, Manufacture and Production Management", Computer-Aided Engineering Journal, Dec 1986, pp240-244

Lau, K. and Hocken, R.J. "A Survey of Current Robot Metrology Methods", ANNALS of the CIRP, 1984, Vol. 33, No. 2, pp485-487

Lee, S.H.; Eman, K.F. and Wu, S.M. "Trajectory Control in the World Coordinate System by an Adaptive Forecasting Algorithm", Special Issue on Robotics of the International Journal of Production Research, March 1989

Leu, M.C. "Robotics Software Systems", Robotics & Computer-Integrated Manufacturing, 1985, Vol. 2, No. 1, pp1-12

Leu, M.C. and Mahajan, R. "Computer Graphic Simulation of Robot Kinematics and Dynamics", Conference Proceedi ngs of Robots 8, Detroit, Michigan, USA 4-7 June 1984, Vol.1, pp4/80-4/101

Loucopoulos, P. and Champion, R. "Knowledge-based Approach to Requirements Engineering Using Method and Domain Knowledge", Knowledge-Based Systems, June 1988, Vol. 1, No. 3, pp179-187

Lozano-Perez, T. "Robot Programming", Proceedings of the IEEE, July 1983, Vol. 71, No. 7, pp821-841

Marconi Communication Branch, Charmsford, England. personal communication with Mr J. Kyrtsoudis, 1987.

Marklund, P. "Using CAD/CAM for Robotic Systems", The Industrial Robot, Oct 1986, pp30-32

Mattis, A. and Gill, K.D. "The Best Robot for the Job : Simulation can Help Decide", The Industrial Robot, 1988, Vol. 15, No. 1, pp32-34

Mauceri, J.G. "Robot Selection Expert 'Rose'", Journal of Automated Reasoning, 1985, Vol. 1, pp357-390

McCarthy, J. M. "Dual Orthogonal Matrices in Manipulator Kinematics", International Journal of Robotics Research, Summer 1986, Vol. 5, No. 2, pp45-51

McDonnell Douglas publication BW 1504065-150, 1986

McGuffin, L.J.; Reid, L.O.; and Sparks, R.S. "MAP/TOP in CIM Distributed Computing", IEEE Network, 1988, Vol. 2, No. 3, pp23-31

Middle, J.E. and Goh, K. "WRAPS - Welding Robot Adaptive Off-line Programming and Expert Control Systems", Second International Conference Developments in Automated and Robot Welding, London, 17-19 Nov 1987

Miller, R. "Manufacturing Simulation: A New Tool for Robotics, FMS and Industrial Process Design", Published SEAI Technical Publications, Sept 1985

Milner, D. A. and Brindley, J.D. "Hardware and Software Developments for a DNC Manufacturing Cell", International Journal of Production Research, 1978, Vol. 16, No. 6, pp441-452

Milovanovic, R. "Towards Sensor-based General Purpose Robot Programming Language", Robotica, 1987, Vol. 5, pp309-316

MMS Draft 6 Document, "EIA project 1393A Draft 6, 1987, Manufacturing Message Specification, Part 1: Service Specification, Appendix B: Guidelines for writing Companion Standards",

Mon, D.L. and Broome, D.R. "Positioning Accuracy and Adaptive Control of a Robot Manipulator", International Conference on Software Engineering for Realtime Systems, Cirencester, U.K., 28-30 Sept. 1987, pp97-103

Morgan, C. "The Rationalisation of Robot Testing", Proceedings of the 10th International Symposium on Industrial Robots, Milan, Italy, March 5-7 1980, pp399-406

Murray, J.J. and Neuman, C.P. "Computational Dynamic Robot Modelling", Proceedings of the 27th MIDWEST Symposium on Circuits and Systems, Morgantown, USA, 11-12 June 1984, Vol. 2, pp479-481

Nissley, L. "Understanding Positioning Errors in Your Robotic Arc Welding System", Welding Journal, Nov 1983, pp30-37

Novak, B. "Robotic Simulation Facilitates Assembly Line Design", Simulation, Dec 1984, pp298-299

O'Grady, P.J. "Controlling Automated Manufacturing Systems", Kogan page, 1986

Offodile, O.F.; Lambert, B.K. and Dudek, R.A. "Development of a computer Aided Robot Selection Procedure (CARSP)", International Journal of Production Research, 1987, Vol. 25, No. 8, pp1109-1121

Okada, T. and Mohri, S. "A Method to Correct Structural Errors in Articulated Robots", Bulletin of JSME, Oct 1985, Vol. 28, No. 244, pp2400-2406

Okino, N. and Shono, M. "A Method to Correct Structural Errors in Articulated Robots", Robotics & Computer-Integrated Manufacturing, 1987, Vol. 3, No. 4, pp429-437

Owen, J. and Bloor, M.S. "Neutral Formats for Product Data Exchange: The Current Situation", Computer Aided Design, 1987, Vol. 19, No. 8, pp436-443

Paul, R.P. "Robot Manipulators: Mathematics, Programming, and Control", The MIT Press, 1981

Paul, R.P. "Sensors and the Off-line Programming of Robots", Proceedings of the 1983 International conference on Automated Manufacturing, Birmingham, England, 16-19 May 1983, pp55-58

Paul, R.P. and Zhang, H. "Computationally Efficient Kinematics for Manipulators With Spherical Wrists Based on the Homogeneous Transformation Representation", International Journal of Robotics Research, Summer 1986, Vol. 5, No. 2, pp32-34

P-E information systems Ltd, Park House, Wick Road, Egham, Surrey. "HOCUS simulation with colour graphics", 1986

Pegden, D. and Ham, I. "Simulation of Manufacturing Systems Using SIMAN", ANNALS of the CIRP, 1982, Vol. 31, No. 1, pp365-369

Pickett, M.S. "Graphical Applications in Robotics", GM Research Publication GMR-4760, 23 July 1984

Porter, J.M.; Case, K. and Bonney, M.C. "SAMMIE : An Ergonomics CAD System for Vehicle Design and Evaluation", BODITEK 1986, Institute of British Carriage and Automobile Manufacturers, University of Keele.

Price, R.B. "Off-line Programming With a Microcomputer", Robots 8 Conference Proceedings, Detroit, MI, USA, 4-7 June 1984, Vol. 2, pp20/95-20/102

Putman, B.W. "RS-232 Simplified : Everything You Need to Know About Connecting, Interfacing and Trouble Shooting Peripheral Devices", Prentice-Hall, 1987

Ramaswamy, V.; Homsup, W. and Anderson, J.N. "Dynamic Computer Simulation and Performance Evaluation of Robotic Mechanisms", System Theory Symposium - 17th Southeastern Symposium, March 1985, pp178-182

Ranky, P.G. "Programming Industrial Robots in FMS", Robotica, 1984, Vol. 2, pp87-92

Ranky, P.G. and Ho, C.Y. "Robot Modelling : Control and applications with software", IFS publications, 1985

Redboard menu, 1986 RACAL-REDAC limited, Tewkesbury, Gloucestershire, England.

Professor Rembold, U., Personal communication 1988, Faculty for Informatics, Institute for Real-Time and Computer Control and Robotics, University of Karlsruhe, Federal Republic of Germany.

Rembold, U.; Blume, C. and Frommherz, B.J. "The Proposed Robot Software Interfaces SRL and IRDATA", Robotics and Computer Integrated Manufacturing, 1985, Vol. 2, No. 3/4, pp219-225

Rembold, U.; Dillman, R. and Negretto, U. Annual report 1987, Faculty for Informatics, Institute for Real-Time and Computer Control and Robotics, University of Karlsruhe (TH), Federal Republic of Germany.

Rembold, U.; Dillman, R. and Huck, M. "A Software System for the Simulation of Robot Based Manufacturing Process", Faculty for Informatics, Institute for Real-Time and Computer Control and Robotics, Publications 1988 of the University of Karlsruhe (TH), Federal Republic of Germany.

Renault, Personal Communication with an engineer at AUTOMAN, 1987

Requicha, A.A.G. "Representations for rigid solids: Theory, methods and systems", Computing Surveys, Dec 1980, Vol. 12, No. 4, pp437-464

Robotics World, "Intelligent Workcells Expand With Software Advances", March 1986a, Vol. 13, NO. 3, pp30-33

Robotics World, "AutoSimulation", March 1986b, Vol. 13, NO. 3, pp3

Robotics World, "Workcells/Robots Programmed Off-line", July 1986c, Vol. 13, NO. 6, pp36-37

Rock, S.T. "Developing Robot Programming Languages Using Existing Language as a Base - A Viewpoint", Robotica, 1989, Vol. 7, pp71-77

Rodighiero, F. and Canciani, A. "An Experience in Task Level Robot Programming", 1987 IEEE Workshop on Languages for Automation, Vienna, Austria, 26-27 Aug. 1987, pp86-89

Roth, Z.S.; Mooring, B.W. and Ravani, B. "An Overview of Robot Calibration", IEEE Journal of Robotics and Automation, October 1987, Vol. RA-3, No. 5, pp377-385

Rueb, K.D. and Wong, A.K.C. "Knowledge-Based Visual Part Identification and Location in a Robot Workcell", International Journal of Machine Tools and Manufacture, 1988, Vol. 28, No. 3, pp235-249

Rui, A.; Weston, R.H.; Gascoign, J.D.; Hodgson, A. and Sumpter, C.M. "Automating Information Transfer in Manufacturing Systems", Computer-Aided Engineering Journal, June 1988, Vol. 5, No. 3, pp113-121

Ruokangas, C.C.; Guthmiller, W.A.; Pierson, B.L.; Sliwinski, K.E. and Lee, J.M.F. "Off-line Programming Motion and Process Commands for Robotic Welding of Space Shuttle Main Engines", Journal of Robotic Systems, 1987, Vol. $, No. 3, pp355-375

Russell, G.T.; Herd, J.T.; Duffy, N.D.; Davies, W.J.; and Finlay, W.J. "An Integrated Control and Error Interpretation System for Collaborating Robots", Special Issue on Robotics of the International Journal of Production Research, March 1989.

Sata, T.; Kimura, F.; Hiraoka, H.; Suzuki, H.; and Fujita, T. "Comprehensive Modelling of a Machine Assembly for Off-line Programming of Industrial Robots", Off-line Robot Programming (edited by Storr, A. and McWaters, J.F.), North-Holland, 1987, pp19-33

Schreiber, R.R. "How to Teach a Robot", Robotics Today, June 1984a, pp51-56

Schreiber, R.R. "Robot System Simulation", Robotics Today, June 1984b, pp81-90

Shimano, B.E.; Geschke, C.C.; and Spalding, C.H. "VAL II : A New Robot Control System for Automatic Manufacturing", IEEE International Conference on Robotics, Atlanta, Georgia, 1984, pp278-292

Sjolund, P. and Donath, M. "Robot Task Planning: Programming Using Interactive Computer Graphics", SME Technical Paper MS 83-344, 1983

Smith, B.M. "A Reporting of the PDES Initation Activities", Room A353, Bldg 220, National Bureau of Standards, Gaithersburg, MD, USA, 1987

Snyder, W.E. "Industrial Robots: Computer Interfacing and Control", Prentice-Hall, 1985

Soroka, B.I. "What Can't Robot Languages Do?", Proceedings of the 13th International Symposium on Industrial Robots, 1983, pp12/1-12/8

Stauffer, R.N. "Robot System Simulation", Robotics Today, June 1984, pp81-90

Stobart, R.K. "Geometric Tools for the Off-line Programming of Robots", Robotica, 1987, Vol. 5, pp273-280

Stone, H.W.; Sanderson, A.C.; and Neuman, C.P. "Arm Signature Identification", Proceedings 1986 IEEE International Conference Robotics and Automation, San Francisco, CA, April 1986, pp41-48

Storr, A. and Schumacher, H. "Programming Methods for Industrial Robots", Off-line Robot Programming (edited by Storr, A. and McWaters, J.F.), North-Holland, 1987, pp1-4.

Strejc, V. "Least Squares and Regression Methods, Trends and Progress in System Identification", Edited by Eykhoff, P. Pergammon Press, 1980

Su, S.Y.W.; Lam, H.; Khatib, M.; Krishnamurthy, V.; Kumar, A.; Malik, S.; Mitchell, M. and Barkmeyer, E. "The Architecture and Prototype Implementation of an Integrated Manufacturing Database Administration System", Sprong COMPCON, 1986

Szeto, V.K.W. and Lichten, L. "Simulation of Parameterized Robots with Solid Modelling", Proceedings of the IASTED International Symposium Robotics and Automation, Lugano, Switzerland, 24-26 June 1985, pp258-262

Tanguy, F. "Assessment of the Mechanical Performance of Industrial Robots", Proceedings of the 12th International Symposium on Industrial Robots Paris, France, June 9-11, 1982, pp349-358

Tarvin, R.L. "Considerations for Off-line Programming a Heavy Duty Industrial Robot", Proceedings of the 10th International Symposium of Industrial Robots, Milan, Italy, March 1980, pp109-117

Thomson, C.C. "Robot Modelling - The Tools Needed for Optimal Design and Utilization", Computer Aided Design, Nov. 1984, Vol. 16, No. 6, pp335-375

Toppen, D.L. "FORTH : An Application Approach", McGraw-Hill, 1985

Van Aken, L. and Van Brussel, H. "A Structured Geometric Database in an Off-line Robot Programming System", Robotica, 1987, Vol. 5, pp333-339

Van Aken, L. and Van Brussel, H. "Robot Programming Languages : A Statement of Problem", Robotica, 1988, Vol. 6, pp141-148

Van Assche, F.; Layzell, P.; Loucopoulos, P. and Speltincx, G. "Information Systems Development: A Rule-based Approach", Knowledge-Based Systems, Sept 1988, Vol. 1, No. 4, pp227-234

Veitschegger, W.K. and Wu, C. "Robot Accuracy Analysis Based on Kinematics", IEEE Journal of Robotics and Automation, Sept 1986, Vol. RA-2, No. 3, pp171-179

Volz, R.D. "Report of the Robot Programming Language Working Group : NATO Workshop on Robot Programming Languages", IEEE Journal of Robotics and Automation, Feb 1988, Vol. 4, No. 1, pp86-90

Vukobratovic, M. and Stokic, D. "Control of Manipulation Robots, Theory and Applications", Scientific Fundamentals of Robots 2, Springer-Verlag, 1982

Wang, T. and Kohli, D. "Closed and Expanded Form of manipulator Dynamics Using Lagrangian Approach", Transactions of ASME Journal of Mechanical, Transmission and Automation Design, (USA), June 1985, Vol. 107, No. 2, pp223-225

Warnecke, W.J.; Weck, M.; Brodbeck, B. and Engel, E. "Assessment of Industrial Robots", Annals of the CIRP, 1982, Vol. 29, no. 1, pp391-396

Weck, M.; Eversheim, W. and Zuhlke, D. "ROBEX - An Off-line Programming System for Industrial Robots", Proceedings of the 11th International Symposium on Industrial Robots, 1981, pp655-662

Weck, M.; Eversheim, W.; Zuhlke, D. and Scholing, H. "Requirements for Robot Off-line Programming Shown at the Example ROBEX", Advanced Software in Robotics, North-Holland, 1984, pp321-330

Weck, M. and Niehaus, T. "Off-line Robot Programming via Standardized Interfaces", Industrial Robot (G.B.), September 1984, Vol. 11, No. 3, pp177-179

Weck, M.; Niehaus, T. and Osterwinter, M. "An Interactive Model Based Robot Programming and Simulation Workstation", Off-line Programming of Industrial Robots (edited by Storr, A. and McWaters, J. F.), North-Holland, 1987

Welch, T.L. "Evaluating Robotic Systems Through Simulation", SME Technical paper MS 83-340, 1983

Wesley, M.A.; Lozano-Perez, T.; Lieberman, L.I.; Lavin, M.A. and Grossman, D.D. "A Geometric Modeling System for Automated Mechanical Assembly", IBM Journal of Research and Development, Jan 1980, Vol. 24, No. 1, pp64-74

Weston, R .H.; Sumpter, C.M. and Gascoigne, J.D. "Industrial Computer Networks and the Role of MAP, Part 1", Microprocessors and microsystems, Sept 1986, Vol. 10, No. 7, pp363-370

Weston, R .H.; Sumpter, C.M. and Gascoigne, J.D. "Industrial Computer Networks and the Role of MAP, Part 2", Microprocessors and microsystems, Jan 1987a, Vol. 11, No. 1, pp21-34

Weston, R .H.; Gascoigne, J.D.; Sumpter, C.M. and Hodgson, A. "Methods of Integrating the Elements of Flexible Assembly Systems", ACME Research Conference Proceedings, Cambridge University, 1987b, Science and Engineering Research Council

Weston, R.H.; Gascoigne, J.D.; Rui, A.; Hodgson, A.; Sumpter, C.M. and Coutts, I. "Steps Towards Information Integration in Manufacture", International Journal of Computer Integrated Manufacturing, 1988, Vol. 1, No. 3, pp140-153

White, J.; Yeates, A. and Skipworth, G. "Tables for Statisticians", 3rd Edition, Stanly Thorne Publications, 1985

Whitney, D.E.; Lozinski, C.A. and Rourke, J.M. "Industrial Robot Calibration Method and Results", Proceedings of ASME Conference Computers and Engineering, Las Vegas, 1984, Vol. 1, pp92-100

Wilhelm, W.E. and Sarin, S.C. "A Structure for Sequencing Robot Activities in Machine Tending Applications", International Journal of Production Research, 1985, Vol. 23, No. 1, pp47-64

Wilkinson, D. and Hallam, R. "A Study of Product Data Transfer Using IGES", Computer-Aided Engineering Journal, June 1987, pp131-136

Wolovich, W.A. "Robotics: Basic Analysis and Design", CBS College Publishing, 1987

Wood, B.O. and Fugelso, M.A. "MCL, The Manufacturing Control Language", 13th International Symposium on Industrial Robots, Chicago, Il, 1983, pp12/84-13/0

Woodwark, J.R. "Generating Wireframes From Set-theoretic Solid Models by Spatial Division", Computer Aided Design, Jul/Aug 1986, Vol. 18, No. 6, pp307-315

Dr Wright, E., Personal communication 1987, Department of Mechanical Engineering, Queen's University, Belfast, Northern Ireland.

Yoffa, N.A. "Off-line Programming for Automotive Spot Welding", Robotics World, April 1988, pp24-25

Yong, Y.F.; Gleave, J.A.; Green, J.L. and Bonney, M.C. "Off-line Programming of Robots", Industrial Handbook on Robotics, Wiley, 1985

# APPENDICES

## APPENDIX A.1 : EXAMPLE OUTPUT OF WORLD-STATE POST-PROCESSING METHODOLOGY

To illustrate the results obtained through the use of world-state post-processing methodology, an original robot simulation program, equivalent robot move positions (in absolute form) and VAL II robot program are shown respectively. The immediately followed example is the original track used for a pick and place operation. The robot move locations were described in absolute form relative to the robot base frame.

```
TRACK JOB10


LOC_OBJ1 : LOCATE OBJECT1  OWNER PLATFORM  AT PLATFORM (SHIFT X  12.500 Y
 12.500 Z  50.000),

LOC_OBJ2 : LOCATE OBJECT2  OWNER PLATFORM  AT PLATFORM (SHIFT X 112.500 Y
 112.500 Z 50.000),

PARKSTEP10 : PARK ,

APP_OBJ1 (MEDSPEED ): POSITION ADEPTONE (SHIFT X  46.860 Y  403.130 Z
 739.750  ROTATE Y  180.000 Z  -90.000),

ONTO_OBJ1 : POSITION ADEPTONE (SHIFT X  46.860 Y  403.130 Z  689.750 ROTATE
 Y  180.000 Z  -90.000),

GRIP_OBJ1 : GRIP OBJECT1 ,

LIFT_OBJ1 (LOWSPEED ): POSITION ADEPTONE (SHIFT X  46.860 Y  403.130 Z
 739.750 ROTATE Y  180.000 Z  -90.000),

APP_OBJ2 : POSITION ADEPTONE (SHIFT X  -53.140 Y  503.130 Z  749.750 ROTATE
 Y  180.000 Z  -90.000),

PILL10 : POSITION ADEPTONE (SHIFT X  -53.140 Y  503.130 Z  739.730 ROTATE
 Y  180.000  Z  -90.000),

RELEASE10 : RELEASE OBJECT1  TO OBJECT2 ,

COMPLETE10 : POSITION ADEPTONE (SHIFT X  -53.140 Y  503.130 Z  752.250 ROTATE
 Y  180.000 Z  -90.000),

PARKSTEP20 : PARK ,

;
```

The following shows the 3D location information extracted from the above track using the world state post-processing methodology. The extracted location information is presented as cartesian coordinates and eulerian angle set.

```
POSITION    46.860  403.130  739.750    0.000  180.000  -90.000
POSITION    46.860  403.130  689.750    0.000  180.000  -90.000
GRIP
POSITION    46.860  403.130  739.750    0.000  180.000  -90.000
POSITION   -53.140  503.130  749.750    0.000  180.000  -90.000
POSITION   -53.140  503.130  739.730    0.000  180.000  -90.000
RELEASE
POSITION   -53.140  503.130  752.250    0.000  180.000  -90.000
```

The equivalent VAL II robot program is obtained through the valformatting module. At the top of this robot program, operating speed and 3D location information are defined. For safety reason, the robot gripper is opened before any robot movement.

```
.PROGRAM JOB10
;PERFORM PICK AND PLACE
; FOR JOB 10
SPEED 50 ALWAYS
SET POINT1= TRANS(46.860 403.130 739.750 0.000 180.000 -90.000)
SET POINT2= TRANS(46.860 403.130 689.750 0.000 180.000 -90.000)
SET POINT3= TRANS(46.860 403.130 739.750 0.000 180.000 -90.000)
SET POINT4= TRANS(-53.140 503.130 749.750 0.000 180.000 -90.000)
SET POINT5= TRANS(-53.140 503.130 739.730 0.000 180.000 -90.000)
SET POINT6= TRANS(-53.140 503.130 752.250 0.000 180.000 -90.000)
OPENI
MOVE POINT1
MOVE POINT2
CLOSEI
MOVE POINT3
MOVE POINT4
MOVE POINT5
OPENI
MOVE POINT6
STOP
.END
```

## APPENDIX A.2 : EXAMPLE OUTPUT OF HIERARCHICAL - TOP DOWN POST-PROCESSING METHODOLOGY

The following example shows the original track JOB1 which is equivalent to track JOB10 as shown in appendix A.1. All move positions are stored relative to any reference object in the workplace (but not necessarily referencing the robot base or world frame). This method of programming is flexible when there are changes in the locations of reference objects.

```
TRACK JOB1

LOC_OBJECT1 : LOCATE OBJECT1  OWNER PLATFORM  AT PLATFORM (SHIFT X  12.500 Y
 12.500 Z  50.000),

LOC_OBJECT2 : LOCATE OBJECT2  OWNER PLATFORM  AT PLATFORM (SHIFT X 112.500 Y
 112.500 Z 50.000),

PARKSTEP1 : PARK ,

APP_OBJECT1 (MEDSPEED ): POSITION OBJECT1 (SHIFT Z  50.000 ROTATE Y  180.000),

ONTO_OBJECT1 : POSITION OBJECT1 ( ROTATE Y  180.000),

GRIP_OBJECT1 : GRIP OBJECT1 ,

LIFT_OBJECT1 (LOWSPEED ): POSITION PLATFORM (SHIFT X  12.500 Y  12.500 Z
 100.000 ROTATE Y  180.000),

APP_OBJECT2 : POSITION OBJECT2 (SHIFT Z  60.000 ROTATE Y  180.000),

PILL : POSITION OBJECT2 (SHIFT Z  50.000 ROTATE Y  180.000),

RELEASING : RELEASE OBJECT1  TO OBJECT2 ,

COMPLETE : POSITION OBJECT1 (SHIFT Z  12.500 ROTATE Y  180.000),

PARKSTEP2 : PARK ,


;
```

The second part of this appendix shows the results of 3D location information extracted from the simulation model and concatenated with move positions and orientations specified in the track using the hierarchical - top down approach. Although these robot move locations were stored relative to objects in the workplace (rather than the robot base or world frame), the intermediate format for location information is similar to that of appendix A.1. However, the location information are described as matrix elements.

```
G_TCP        1.000    0.000    0.000    0.000    1.000    0.000
     0.000    0.000    1.000    0.000    0.000  114.500
POSITION     0.174    0.985    0.000    0.985   -0.174    0.000
     0.000    0.000   -1.000  208.067 -694.872  755.500
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  739.750
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  689.750
GRIP
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  739.750
POSITION     0.000   -1.000    0.000    1.000    0.000    0.000
     0.000    0.000    1.000   46.860  403.130  679.750
POSITION     0.000   -1.000    0.000    1.000    0.000    0.000
     0.000    0.000    1.000   46.860  403.130  689.750
RELEASE
POSITION     0.000   -1.000    0.000    1.000    0.000    0.000
     0.000    0.000    1.000   46.860  403.130  677.250
POSITION     0.174    0.985    0.000    0.985   -0.174    0.000
     0.000    0.000   -1.000  208.067 -694.872  755.500
POSITION     0.174    0.985    0.000    0.985   -0.174    0.000
     0.000    0.000   -1.000  208.067 -694.872  755.500
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  739.750
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  689.750
GRIP
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000   46.860  403.130  739.750
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000  -53.140  503.130  749.750
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000  -53.140  503.130  739.730
RELEASE
POSITION     0.000   -1.000    0.000   -1.000    0.000    0.000
     0.000    0.000   -1.000  -53.140  503.130  752.250
POSITION     0.174    0.985    0.000    0.985   -0.174    0.000
     0.000    0.000   -1.000  208.067 -694.872  755.500
```

The final part of this appendix shows the equivalent VAL II robot program. At the top of this robot program, a TOOL command statement is used to correct the tool frame (from centre frame to robot mounting flange frame). Robot operating speed and robot command statements are defined with the robot move location information described in matrix form in one single file.

```
.PROGRAM JOB1
 ; PROGRAM POST-PROCESSED BY TOP-DOWN APPROACH
 ; FOR JOB 1
 SPEED 50 ALWAYS
 OPENI
 TOOL G_TCP
 MOVES PT1
 MOVES PT2
 MOVES PT3
 CLOSEI
 MOVES PT4
 MOVES PT5
 MOVES PT6
 OPENI
 MOVES PT7
 MOVES PT8
 MOVES PT9
 MOVES PT10
 MOVES PT11
 CLOSEI
 MOVES PT12
 MOVES PT13
 MOVES PT14
 OPENI
 MOVES PT15
 MOVES PT16
 STOP
.END
```

```
.LOCATIONS
 G_TCP    1.000      0.000      0.000     0.000     1.000     0.000
     0.000      0.000      1.000     0.000     0.000   114.500
 PT1      0.174      0.985      0.000     0.985    -0.174     0.000
     0.000      0.000     -1.000   208.067  -694.872   755.500
 PT2      0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   739.750
 PT3      0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   689.750
 PT4      0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   739.750
 PT5      0.000     -1.000      0.000     1.000     0.000     0.000
     0.000      0.000      1.000    46.860   403.130   679.750
 PT6      0.000     -1.000      0.000     1.000     0.000     0.000
     0.000      0.000      1.000    46.860   403.130   689.750
 PT7      0.000     -1.000      0.000     1.000     0.000     0.000
     0.000      0.000      1.000    46.860   403.130   677.250
 PT8      0.174      0.985      0.000     0.985    -0.174     0.000
     0.000      0.000     -1.000   208.067  -694.872   755.500
 PT9      0.174      0.985      0.000     0.985    -0.174     0.000
     0.000      0.000     -1.000   208.067  -694.872   755.500
 PT10     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   739.750
 PT11     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   689.750
 PT12     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000    46.860   403.130   739.750
 PT13     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000   -53.140   503.130   749.750
 PT14     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000   -53.140   503.130   739.730
 PT15     0.000     -1.000      0.000    -1.000     0.000     0.000
     0.000      0.000     -1.000   -53.140   503.130   752.250
 PT16     0.174      0.985      0.000     0.985    -0.174     0.000
     0.000      0.000     -1.000   208.067  -694.872   755.500
.END
```

## APPENDIX A.3 : EXAMPLE OUTPUT OF HIERARCHICAL - APPEARANCE POST-PROCESSING METHODOLOGY

To illustrate the differences between the "TOP DOWN" and "APPEARANCE" post-processing methodologies, the same GRASP track (as shown in appendix A.2) is used. This part of the appendix shows the equivalent VAL II robot program with location information presented in relation to reference objects. The location information is stored in a separate file.

At the top of this robot program, a TOOL statement is used to correct the tool frame (from centre frame to robot mounting flange frame). SET statements are used to define the absolute positions of all move positions. Robot operating speeds are defined in a similar way as used in the track. Robot move location information is presented as matrix elements in a separate file. This VAL II robot program can be re-converted into a track through the VALTOTRACK module which is explained in appendix A.4.

```
.PROGRAM JOB1
 TOOL G_TCP
 SET OBJECT1=PLATFORM:TF1
 SET OBJECT2=PLATFORM:TF2
 OPENI
 MOVES PARK
 SPEED  100.000 MMPS ALWAYS
 MOVES OBJECT1:TF3
 MOVES OBJECT1:TF4
 CLOSEI
 SPEED   50.000 MMPS ALWAYS
 MOVES PLATFORM:TF5
 MOVES OBJECT2:TF6
 MOVES OBJECT2:TF7
 OPENI
 MOVES OBJECT1:TF8
 MOVES PARK
.END
```

```
.LOCATIONS
 G_TCP      1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000    1.000    0.000    0.000  114.500
 PLATFORM   0.000    1.000    0.000   -1.000    0.000    0.000
   0.000    0.000    1.000   59.360  390.630  639.750
 TF1      1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000    1.000   12.500   12.500   50.000
 TF2      1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000    1.000  112.500  112.500   50.000
 PARK     0.174    0.985    0.000    0.985   -0.174    0.000
   0.000    0.000   -1.000  208.067 -694.872  755.500
 TF3     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000    0.000    0.000   50.000
 TF4     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000    0.000    0.000    0.000
 TF5     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000   12.500   12.500  100.000
 TF6     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000    0.000    0.000   60.000
 TF7     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000    0.000    0.000   50.000
 TF8     -1.000    0.000    0.000    0.000    1.000    0.000
   0.000    0.000   -1.000    0.000    0.000   12.500
.END
```

## APPENDIX A.4 : EXAMPLE OUTPUT OF VALTOTRACK MODULE

Through the use of VALTOTRACK module, a VAL II robot program can be re-converted into a GRASP track. Thus the robot simulator can be used in a different way. The original input VAL II robot program and location information were presented in appendix A.3. Since the robot program is not associated with any path and step names, the module automatically use PATH1, PATH2, STEP1, STEP2 etc for re-converting robot program into a GRASP track.

```
PATH  PATH1 STRAIGHT SPEED  100.000 ;
PATH  PATH2 STRAIGHT SPEED   50.000 ;
TRACK JOB1
STEP1 : LOCATE OBJECT1  OWNER PLATFORM AT PLATFORM (SHIFT X 12.500 Y 12.500 Z
 50.000),
STEP2 : LOCATE OBJECT2  OWNER PLATFORM AT PLATFORM (SHIFT X 112.500 Y 112.500 Z
 50.000),
STEP3 : PARK ,
STEP4 (PATH1 ) : POSITION OBJECT1 (SHIFT Z 50.000 ROTATE Y 180.000),
STEP5 : POSITION OBJECT1 ( ROTATE Y 180.000),
STEP6 : GRIP OBJECT1 ,
STEP7 (PATH2 ) : POSITION PLATFORM (SHIFT X 12.500 Y 12.500 Z 100.000 ROTATE Y
 180.000),
STEP8 : POSITION OBJECT2 (SHIFT Z 60.000 ROTATE Y 180.000),
STEP9 : POSITION OBJECT2 (SHIFT Z 50.000 ROTATE Y 180.000),
STEP10 : RELEASE OBJECT1 TO OBJECT2 ,
STEP11 : POSITION OBJECT1 (SHIFT Z 12.500 ROTATE Y 180.000),
STEP12 : PARK ,
STOP ;
```

## APPENDIX B.1 GRASP SYNTAX GENERATED FOR SOLID MODELLING THROUGH COMPUTER ASSISTED SOLID MODELLING MODULE

This appendix shows GRASP syntax of a pallet generated through one of the software modules derived for parameterised solid modelling.

```
CUBOID PALLET_BASE  300.000  250.000   20.000;
CUBOID PALLET_DBAR1   20.000  250.000  100.000;
COPY PALLET_DBAR1 PALLET_DBAR2 ;
CUBOID PALLET_DBAR3   20.000  210.000  100.000;
CUBOID PALLET_DBAR4   20.000  210.000  100.000;
CUBOID PALLET_DBAR5   20.000  210.000  100.000;
CUBOID PALLET_CBAR1  260.000   20.000  100.000;
COPY PALLET_CBAR1 PALLET_CBAR2;
CUBOID PALLET_CBAR3   50.000   20.000  100.000;
CUBOID PALLET_CBAR4   50.000   20.000  100.000;
CUBOID PALLET_CBAR5   50.000   20.000  100.000;
CUBOID PALLET_CBAR6   50.000   20.000  100.000;
CUBOID PALLET_CBAR7   50.000   20.000  100.000;
CUBOID PALLET_CBAR8   50.000   20.000  100.000;
CUBOID PALLET_CBAR9   50.000   20.000  100.000;
CUBOID PALLET_CBAR10   50.000   20.000  100.000;
CUBOID PALLET_CBAR11   50.000   20.000  100.000;
CUBOID PALLET_CBAR12   50.000   20.000  100.000;
CUBOID PALLET_CBAR13   50.000   20.000  100.000;
CUBOID PALLET_CBAR14   50.000   20.000  100.000;
SET PALLET = PALLET_BASE
  PALLET_DBAR1 (SHIFT Z   20.000)
  PALLET_DBAR2 (SHIFT X  280.000 Z   20.000)
  PALLET_DBAR3 (SHIFT X  70.000 Y   20.000 Z   20.000)
  PALLET_DBAR4 (SHIFT X  140.000 Y   20.000 Z   20.000)
  PALLET_DBAR5 (SHIFT X  210.000 Y   20.000 Z   20.000)
  PALLET_CBAR1 (SHIFT X   20.000 Z   20.000)
  PALLET_CBAR2 (SHIFT X   20.000 Y  230.000 Z   20.000)
  PALLET_CBAR3 (SHIFT X   20.000 Y   57.500 Z   20.000)
  PALLET_CBAR4 (SHIFT X   90.000 Y   57.500 Z   20.000)
  PALLET_CBAR5 (SHIFT X  160.000 Y   57.500 Z   20.000)
  PALLET_CBAR6 (SHIFT X  230.000 Y   57.500 Z   20.000)
  PALLET_CBAR7 (SHIFT X   20.000 Y  115.000 Z   20.000)
  PALLET_CBAR8 (SHIFT X   90.000 Y  115.000 Z   20.000)
  PALLET_CBAR9 (SHIFT X  160.000 Y  115.000 Z   20.000)
  PALLET_CBAR10 (SHIFT X  230.000 Y  115.000 Z   20.000)
  PALLET_CBAR11 (SHIFT X   20.000 Y  172.500 Z   20.000)
  PALLET_CBAR12 (SHIFT X   90.000 Y  172.500 Z   20.000)
  PALLET_CBAR13 (SHIFT X  160.000 Y  172.500 Z   20.000)
  PALLET_CBAR14 (SHIFT X  230.000 Y  172.500 Z   20.000)
;
STOP ;
```

## APPENDIX B.2 : GRASP SYNTAX GENERATED FOR TASK SIMULATION THROUGH COMPUTER ASSISTED TASK PROGRAM GENERATION MODULE

### (i) PALLETISING TASKS

In this appendix an example output of a robot palletising program generated by the PROG_PALLET software module. Local variables are declared at the top of this program. The program generated by the software module is created in the high level GRASP language where reference objects and workpieces can be defined as arrays. These arrays can be advanced as the programmer desires. After the local variable declarations, it is important to fill these arrays with the names of reference objects and workpeices. Locate statements are used to give an initial position for each objects that are to be manipulated during the palletising, such that every time the program is replayed the initial condition is maintained. The following program was created for option C with palletising pattern G to illustrate how parameterised tasks can be applied.

```
TRACK PCG
VARIABLES
REFOBJECT PICK.REF(1:12)
WORKPIECE WPE (1:12)
END
STEP1 : SET PICK.REF TO LIST (T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12),
STEP2 : SET WPE TO LIST (BOX1,BOX2,BOX3,BOX4,BOX5,BOX6,BOX7,BOX8,
 BOX9,BOX10,BOX11,BOX12),
STEP3 : ADVANCE PICK.REF TO 1  ,
STEP4 : ADVANCE WPE TO 1 ,
STEP5 : REPEAT 12 TIMES ,
STEP6 : LOCATE WPE OWNER PALLET1 AT PICK.REF,
STEP7 : ADVANCE PICK.REF BY 1 ,
STEP8 : ADVANCE WPE BY 1 ,
STEP9 : ENDREPEAT FROM STEP5 ,
STEP10 : ADVANCE PICK.REF TO 1  ,
STEP11 : ADVANCE WPE TO 1 ,
STEP12 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP13 : POSITION PICK.REF (ROTATE Y 180.00),
STEP14 : GRIP WPE ,
STEP15 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP16 : POSITION PALLET2 (SHIFT X 30.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP17 : POSITION PALLET2 (SHIFT X 30.00 Y 30.00 Z 50.00 ROTATE Y 180.00),
STEP18 : RELEASE WPE TO PALLET2 ,
STEP19 : POSITION PALLET2 (SHIFT X 30.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP20 : ADVANCE WPE BY 1 ,
STEP21 : ADVANCE PICK.REF BY 1 ,
STEP22 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP23 : POSITION PICK.REF (ROTATE Y 180.00),
STEP24 : GRIP WPE ,
STEP25 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP26 : POSITION PALLET2 (SHIFT X 90.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP27 : POSITION PALLET2 (SHIFT X 90.00 Y 30.00 Z 50.00 ROTATE Y 180.00),
STEP28 : RELEASE WPE TO PALLET2 ,
STEP29 : POSITION PALLET2 (SHIFT X 90.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
```

```
STEP30 : ADVANCE WPE BY 1 ,
STEP31 : ADVANCE PICK.REF BY 1 ,
STEP32 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP33 : POSITION PICK.REF (ROTATE Y 180.00),
STEP34 : GRIP WPE ,
STEP35 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP36 : POSITION PALLET2 (SHIFT X 150.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP37 : POSITION PALLET2 (SHIFT X 150.00 Y 30.00 Z 50.00 ROTATE Y 180.00),
STEP38 : RELEASE WPE TO PALLET2 ,
STEP39 : POSITION PALLET2 (SHIFT X 150.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP40 : ADVANCE WPE BY 1 ,
STEP41 : ADVANCE PICK.REF BY 1 ,
STEP42 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP43 : POSITION PICK.REF (ROTATE Y 180.00),
STEP44 : GRIP WPE ,
STEP45 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP46 : POSITION PALLET2 (SHIFT X 210.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP47 : POSITION PALLET2 (SHIFT X 210.00 Y 30.00 Z 50.00 ROTATE Y 180.00)
STEP48 : RELEASE WPE TO PALLET2 ,
STEP49 : POSITION PALLET2 (SHIFT X 210.00 Y 30.00 Z 70.00 ROTATE Y 180.00),
STEP50 : ADVANCE WPE BY 1 ,
STEP51 : ADVANCE PICK.REF BY 1 ,
STEP52 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP53 : POSITION PICK.REF (ROTATE Y 180.00),
STEP54 : GRIP WPE ,
STEP55 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP56 : POSITION PALLET2 (SHIFT X 30.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP57 : POSITION PALLET2 (SHIFT X 30.00 Y 90.00 Z 50.00 ROTATE Y 180.00),
STEP58 : RELEASE WPE TO PALLET2 ,
STEP59 : POSITION PALLET2 (SHIFT X 30.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP60 : ADVANCE WPE BY 1 ,
STEP61 : ADVANCE PICK.REF BY 1 ,
STEP62 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP63 : POSITION PICK.REF (ROTATE Y 180.00),
STEP64 : GRIP WPE ,
STEP65 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP66 : POSITION PALLET2 (SHIFT X 90.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP67 : POSITION PALLET2 (SHIFT X 90.00 Y 90.00 Z 50.00 ROTATE Y 180.00),
STEP68 : RELEASE WPE TO PALLET2 ,
STEP69 : POSITION PALLET2 (SHIFT X 90.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP70 : ADVANCE WPE BY 1 ,
STEP71 : ADVANCE PICK.REF BY 1 ,
STEP72 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP73 : POSITION PICK.REF (ROTATE Y 180.00),
STEP74 : GRIP WPE ,
STEP75 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP76 : POSITION PALLET2 (SHIFT X 150.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP77 : POSITION PALLET2 (SHIFT X 150.00 Y 90.00 Z 50.00 ROTATE Y 180.00),
STEP78 : RELEASE WPE TO PALLET2 ,
STEP79 : POSITION PALLET2 (SHIFT X 150.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
```

```
STEP80 : ADVANCE WPE BY 1 ,
STEP81 : ADVANCE PICK.REF BY 1 ,
STEP82 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP83 : POSITION PICK.REF (ROTATE Y 180.00),
STEP84 : GRIP WPE ,
STEP85 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP86 : POSITION PALLET2 (SHIFT X 210.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP87 : POSITION PALLET2 (SHIFT X 210.00 Y 90.00 Z 50.00 ROTATE Y 180.00),
STEP88 : RELEASE WPE TO PALLET2 ,
STEP89 : POSITION PALLET2 (SHIFT X 210.00 Y 90.00 Z 70.00 ROTATE Y 180.00),
STEP90 : ADVANCE WPE BY 1 ,
STEP91 : ADVANCE PICK.REF BY 1 ,
STEP92 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP93 : POSITION PICK.REF (ROTATE Y 180.00),
STEP94 : GRIP WPE ,
STEP95 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP96 : POSITION PALLET2 (SHIFT X 30.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP97 : POSITION PALLET2 (SHIFT X 30.00 Y 150.00 Z 50.00 ROTATE Y 180.00),
STEP98 : RELEASE WPE TO PALLET2 ,
STEP99 : POSITION PALLET2 (SHIFT X 30.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP100 : ADVANCE WPE BY 1 ,
STEP101 : ADVANCE PICK.REF BY 1 ,
STEP102 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP103 : POSITION PICK.REF (ROTATE Y 180.00),
STEP104 : GRIP WPE ,
STEP105 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP106 : POSITION PALLET2 (SHIFT X 90.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP107 : POSITION PALLET2 (SHIFT X 90.00 Y 150.00 Z 50.00 ROTATE Y 180.00),
STEP108 : RELEASE WPE TO PALLET2 ,
STEP109 : POSITION PALLET2 (SHIFT X 90.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP110 : ADVANCE WPE BY 1 ,
STEP111 : ADVANCE PICK.REF BY 1 ,
STEP112 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP113 : POSITION PICK.REF (ROTATE Y 180.00),
STEP114 : GRIP WPE ,
STEP115 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP116 : POSITION PALLET2 (SHIFT X 150.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP117 : POSITION PALLET2 (SHIFT X 150.00 Y 150.00 Z 50.00 ROTATE Y 180.00),
STEP118 : RELEASE WPE TO PALLET2 ,
STEP119 : POSITION PALLET2 (SHIFT X 150.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP120 : ADVANCE WPE BY 1 ,
STEP121 : ADVANCE PICK.REF BY 1 ,
STEP122 : POSITION PICK.REF (SHIFT Z 20.00 ROTATE Y 180.00),
STEP123 : POSITION PICK.REF (ROTATE Y 180.00),
STEP124 : GRIP WPE ,
STEP125 : POSITION PICK.REF (SHIFT Z 70.00 ROTATE Y 180.00),
STEP126 : POSITION PALLET2 (SHIFT X 210.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP127 : POSITION PALLET2 (SHIFT X 210.00 Y 150.00 Z 50.00 ROTATE Y 180.00),
STEP128 : RELEASE WPE TO PALLET2 ,
STEP129 : POSITION PALLET2 (SHIFT X 210.00 Y 150.00 Z 70.00 ROTATE Y 180.00),
STEP130 : ADVANCE WPE BY 1 ,
STEP131 : ADVANCE PICK.REF BY 1 ,
;
STOP ;
```

## (ii) MACHINE TENDING TASKS

This appendix shows an example output of a machine tending program generated by the PROG_TEND software module. The robot tending program is created for one of the shopfloor configurations as described, where any object can be assigned to any machine. A robot program is generated together with shell programs for the attendant peripheral robotic devices. The synchronisation of these robotic devices is through simple sensory input and output.

Global variables are declared before any robot or peripheral programs. In general three types of variable can be used, including signal, workpiece and reference objects variables. It is important that signal variables must be declared at the global level. Within one program a signal variable can only be used for input or output purpose. If any path control is to be used in controlling robot motion, the path must be defined before it is used in any program for robot or other peripheral peripheral devices. The author also illustrate how a path can be controlled through this example.

```
GLOBAL
VARIABLES
SIGNAL PART_READY, PART_CLEAR_IN,
MILL1_START, MILL1_STOP, MILL1_CLEAR,
DRILL1_START, DRILL1_STOP, DRILL1_CLEAR,
LATHE1_START, LATHE1_STOP, LATHE1_CLEAR
WORKPIECE PART (1:10)
END ;
PATH ST_CONV STRAIGHT SPEED 80 ;
PATH ST_MC PTPT ;
```

The following track describes the conveyor belt movement which drives raw material or partially completed parts into the robot workplace. Assuming a sensing device is installed at the pick up position on the conveyor system, and when a part is arrived at such a position, a signal output is generated from the sensor which stops the conveyor belt. The conveyor will not be re-started until after the part is picked up by the robot.

```
TRACK CONV_IN_IN
VARIABLES
REAL DIST(1:10)
END
CONV_IN_IN1 : SET PART TO LIST (OBJ1,OBJ10,OBJ2,OBJ3,OBJ9,OBJ4,
 OBJ5,OBJ6,OBJ7,OBJ8) ,
CONV_IN_IN2 : SET DIST TO LIST(350.0,600.0,850.0,1100.0,1350.0,
 1600.0,1850.0,2100.0,2350.0,2600.0) ,
CONV_IN_IN3 : ADVANCE PART TO 1 ,
CONV_IN_IN4 : ADVANCE DIST TO 1 ,
CONV_IN_IN5 : REPEAT 10 TIMES ,
CONV_IN_IN6 : LOCATE PART OWNER CONV_IN_BELT AT CONV_IN_BELT(NULL),
CONV_IN_IN7 : LOCATE PART BY (SHIFT X DIST) ,
CONV_IN_IN8 : ADVANCE PART BY 1 ,
```

```
CONV_IN_IN9 : ADVANCE DIST BY 1 ,
CONV_IN_IN10 : ENDREPEAT FROM CONV_IN_IN5 ,
CONV_IN_IN11 : ADVANCE DIST TO 1,
CONV_IN_IN12 : SET PART_READY TO FALSE ,
CONV_IN_IN13 : LOCATE CONV_IN_BELT  OWNER CONV_IN AT CONV_IN
 (SHIFT X 475 Y 11910 Z 750),
CONV_IN_IN14 (ST_CONV) : POSITION CONV_IN_TARG (SHIFT X -DIST) ,
CONV_IN_IN15 : SET PART_READY TO TRUE ,
CONV_IN_IN16 : WAIT UNTIL PART_CLEAR_IN ,
CONV_IN_IN17 : SET PART_READY TO FALSE ,
CONV_IN_IN18 : ADVANCE DIST BY 1,
CONV_IN_IN19 : GOTO CONV_IN_IN14 ,
 ;
```

This track describes a second conveyor belt movement which drives completed parts out of the robot workplace to other workshop in the manufacturing system. This conveyor belt is not controlled by any signal and moves continuously and stops when the robot task is completed. This also illustrates different ways of programming.

```
TRACK CONV_OUT_OUT
CONV_OUT_OUT1 : LOCATE CONV_OUT_BELT  OWNER CONV_OUT AT CONV_OUT
 (SHIFT X 475 Y 11910 Z 750),
CONV_OUT_OUT2 : POSITION CONV_OUT_TARG ,
CONV_OUT_OUT3 (ST_CONV) : POSITION CONV_OUT_TARG (SHIFT X -20000) ,
 ;
```

The following track describes the robot task and this robot track calls different macros (sub-routines) for different attendant machines. When the raw material is arrived at the pick up position, a signal is received by the robot controller that the part is ready to be pick up. Initially, every attendant machine is ready for processing raw material, the robot will load each machine according to the priority order specified by the programmer. Once the workpiece is loaded into the machine. A signal output from the attendant machine to the robot controller preventing re-loading before unloading process takes place and so avoid unnecessary collision and breakage. Once the manufacturing process is completed, the attendant machine outputs another signal to the robot controller signifies the process is completed and the finished part is ready to be unloaded. When the robot finishes unloading such machine, the machine is ready for the next workpiece and so on.

```
TRACK C
CSTEP1 : ADVANCE PART TO 1 ,
CSTEP2 : CALL MILL1_LOAD ,
CSTEP3 : ADVANCE PART BY 1 ,
CSTEP4 : CALL DRILL1_LOAD ,
CSTEP5 : ADVANCE PART BY 1 ,
CSTEP6 : CALL LATHE1_LOAD ,
CSTEP7 : ADVANCE PART BY 1 ,
CSTEP8 : ADVANCE PART TO 1 ,
```

```
CSTEP_REP : REPEAT 2 TIMES ,
CSTEP9 : CALL MILL1_UNLOAD ,
CSTEP10 : ADVANCE PART BY 3,
CSTEP11 : CALL MILL1_LOAD ,
CSTEP12 : ADVANCE PART BY 8,
CSTEP13 : CALL DRILL1_UNLOAD ,
CSTEP14 : ADVANCE PART BY 3,
CSTEP15 : CALL DRILL1_LOAD ,
CSTEP16 : ADVANCE PART BY 8,
CSTEP17 : CALL LATHE1_UNLOAD ,
CSTEP18 : ADVANCE PART BY 3,
CSTEP19 : CALL LATHE1_LOAD ,
CSTEP20 : ADVANCE PART BY 8,
CSTEP_END : ENDREPEAT FROM CSTEP_REP ,
CSTEP21 : CALL MILL1_UNLOAD ,
CSTEP22 : ADVANCE PART BY 1 ,
CSTEP23 : CALL DRILL1_UNLOAD ,
CSTEP24 : ADVANCE PART BY 1 ,
CSTEP25 : CALL LATHE1_UNLOAD ,
CSTEP26 : ADVANCE PART BY 1 ,
 ;

TRACK MILL1_LOAD
MILL1_LOAD1 : WAIT UNTIL PART_READY,
MILL1_LOAD2 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_LOAD3 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
MILL1_LOAD4 : POSITION CONV_IN_TARG (NULL) ,
MILL1_LOAD5 : GRIP PART ,
MILL1_LOAD6 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
MILL1_LOAD7 : SET PART_CLEAR_IN TO TRUE ,
MILL1_LOAD8 : SET MILL1_CLEAR TO FALSE,
MILL1_LOAD9 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_LOAD10 : SET PART_CLEAR_IN TO FALSE,
MILL1_LOAD11 : POSITION MILL1_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_LOAD12 : POSITION MILL1_TARG (SHIFT Z  200.000) ,
MILL1_LOAD13 : POSITION MILL1_TARG ,
MILL1_LOAD14 : RELEASE PART TO MILL1_TARG ,
MILL1_LOAD15 : POSITION MILL1_TARG (SHIFT Z  200.000) ,
MILL1_LOAD16 : SET MILL1_START TO TRUE ,
MILL1_LOAD17 : POSITION MILL1_TARG (SHIFT Y -400  Z  200.000) ,
MILL1_LOAD18 : SET MILL1_START TO FALSE ,
MILL1_LOAD19 : RETURN ,
 ;

TRACK MILL1_PROG
MILL1_PROG1 : WAIT UNTIL MILL1_START ,
MILL1_PROG2 : PAUSE 2.500 ,
MILL1_PROG3 : SET MILL1_STOP TO TRUE ,
MILL1_PROG4 : WAIT UNTIL MILL1_CLEAR ,
MILL1_PROG5 : SET MILL1_STOP TO FALSE ,
MILL1_PROG6 : GOTO MILL1_PROG1 ,
 ;
```

```
TRACK MILL1_UNLOAD
MILL1_UNLOAD1 : WAIT UNTIL MILL1_STOP,
MILL1_UNLOAD2 : POSITION MILL1_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_UNLOAD3 : POSITION MILL1_TARG (SHIFT Z  200.000) ,
MILL1_UNLOAD4 : POSITION MILL1_TARG ,
MILL1_UNLOAD5 : GRIP PART ,
MILL1_UNLOAD6 : POSITION MILL1_TARG (SHIFT Z  200.000) ,
MILL1_UNLOAD7: POSITION MILL1_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_UNLOAD8 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_UNLOAD9 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
MILL1_UNLOAD10 : POSITION CONV_OUT_TARG (NULL) ,
MILL1_UNLOAD11 : RELEASE PART TO CONV_OUT_BELT,
MILL1_UNLOAD12 : SET MILL1_CLEAR TO TRUE ,
MILL1_UNLOAD13 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
MILL1_UNLOAD14 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
MILL1_UNLOAD15 : RETURN ,
 ;

TRACK DRILL1_LOAD
DRILL1_LOAD1 : WAIT UNTIL PART_READY,
DRILL1_LOAD2 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_LOAD3 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
DRILL1_LOAD4 : POSITION CONV_IN_TARG (NULL) ,
DRILL1_LOAD5 : GRIP PART ,
DRILL1_LOAD6 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
DRILL1_LOAD7 : SET PART_CLEAR_IN TO TRUE ,
DRILL1_LOAD8 : SET DRILL1_CLEAR TO FALSE,
DRILL1_LOAD9 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_LOAD10 : SET PART_CLEAR_IN TO FALSE,
DRILL1_LOAD11 : POSITION DRILL1_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_LOAD12 : POSITION DRILL1_TARG (SHIFT Z  200.000) ,
DRILL1_LOAD13 : POSITION DRILL1_TARG ,
DRILL1_LOAD14 : RELEASE PART TO DRILL1_TARG ,
DRILL1_LOAD15 : POSITION DRILL1_TARG (SHIFT Z  200.000) ,
DRILL1_LOAD16 : SET DRILL1_START TO TRUE ,
DRILL1_LOAD17 : POSITION DRILL1_TARG (SHIFT Y -400  Z  200.000) ,
DRILL1_LOAD18 : SET DRILL1_START TO FALSE ,
DRILL1_LOAD19 : RETURN ,
 ;

TRACK DRILL1_PROG
DRILL1_PROG1 : WAIT UNTIL DRILL1_START ,
DRILL1_PROG2 : PAUSE 2.500 ,
DRILL1_PROG3 : SET DRILL1_STOP TO TRUE ,
DRILL1_PROG4 : WAIT UNTIL DRILL1_CLEAR ,
DRILL1_PROG5 : SET DRILL1_STOP TO FALSE ,
DRILL1_PROG6 : GOTO DRILL1_PROG1 ,
 ;
```

```
TRACK DRILL1_UNLOAD
DRILL1_UNLOAD1 : WAIT UNTIL DRILL1_STOP,
DRILL1_UNLOAD2 : POSITION DRILL1_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_UNLOAD3 : POSITION DRILL1_TARG (SHIFT Z  200.000) ,
DRILL1_UNLOAD4 : POSITION DRILL1_TARG ,
DRILL1_UNLOAD5 : GRIP PART ,
DRILL1_UNLOAD6 : POSITION DRILL1_TARG (SHIFT Z  200.000) ,
DRILL1_UNLOAD7: POSITION DRILL1_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_UNLOAD8 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_UNLOAD9 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
DRILL1_UNLOAD10 : POSITION CONV_OUT_TARG (NULL) ,
DRILL1_UNLOAD11 : RELEASE PART TO CONV_OUT_BELT,
DRILL1_UNLOAD12 : SET DRILL1_CLEAR TO TRUE ,
DRILL1_UNLOAD13 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
DRILL1_UNLOAD14 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
DRILL1_UNLOAD15 : RETURN ,
 ;

TRACK LATHE1_LOAD
LATHE1_LOAD1 : WAIT UNTIL PART_READY,
LATHE1_LOAD2 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_LOAD3 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
LATHE1_LOAD4 : POSITION CONV_IN_TARG (NULL) ,
LATHE1_LOAD5 : GRIP PART ,
LATHE1_LOAD6 : POSITION CONV_IN_TARG (SHIFT Z  200.000) ,
LATHE1_LOAD7 : SET PART_CLEAR_IN TO TRUE ,
LATHE1_LOAD8 : SET LATHE1_CLEAR TO FALSE,
LATHE1_LOAD9 : POSITION CONV_IN_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_LOAD10 : SET PART_CLEAR_IN TO FALSE,
LATHE1_LOAD11 : POSITION LATHE1_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_LOAD12 : POSITION LATHE1_TARG (SHIFT Z  200.000) ,
LATHE1_LOAD13 : POSITION LATHE1_TARG ,
LATHE1_LOAD14 : RELEASE PART TO LATHE1_TARG ,
LATHE1_LOAD15 : POSITION LATHE1_TARG (SHIFT Z  200.000) ,
LATHE1_LOAD16 : SET LATHE1_START TO TRUE ,
LATHE1_LOAD17 : POSITION LATHE1_TARG (SHIFT Y -400  Z  200.000) ,
LATHE1_LOAD18 : SET LATHE1_START TO FALSE ,
LATHE1_LOAD19 : RETURN ,
 ;

TRACK LATHE1_PROG
LATHE1_PROG1 : WAIT UNTIL LATHE1_START ,
LATHE1_PROG2 : PAUSE 2.500 ,
LATHE1_PROG3 : SET LATHE1_STOP TO TRUE ,
LATHE1_PROG4 : WAIT UNTIL LATHE1_CLEAR ,
LATHE1_PROG5 : SET LATHE1_STOP TO FALSE ,
LATHE1_PROG6 : GOTO LATHE1_PROG1 ,
 ;
```

```
TRACK LATHE1_UNLOAD
LATHE1_UNLOAD1 : WAIT UNTIL LATHE1_STOP,
LATHE1_UNLOAD2 : POSITION LATHE1_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_UNLOAD3 : POSITION LATHE1_TARG (SHIFT Z  200.000) ,
LATHE1_UNLOAD4 : POSITION LATHE1_TARG ,
LATHE1_UNLOAD5 : GRIP PART ,
LATHE1_UNLOAD6 : POSITION LATHE1_TARG (SHIFT Z  200.000) ,
LATHE1_UNLOAD7: POSITION LATHE1_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_UNLOAD8 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_UNLOAD9 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
LATHE1_UNLOAD10 : POSITION CONV_OUT_TARG (NULL) ,
LATHE1_UNLOAD11 : RELEASE PART TO CONV_OUT_BELT,
LATHE1_UNLOAD12 : SET LATHE1_CLEAR TO TRUE ,
LATHE1_UNLOAD13 : POSITION CONV_OUT_TARG (SHIFT Z  200.000) ,
LATHE1_UNLOAD14 : POSITION CONV_OUT_TARG (SHIFT Y -400 Z  200.000) ,
LATHE1_UNLOAD15 : RETURN ,
 ;
STOP
```

## APPENDIX C.1 : EXAMPLE DATA SHEET OF ROBOT KINEMATIC CHARARTERISTICS SUPPLIED BY A ROBOT MANUFACTURER

Acceleration Performance of Adept One Robot

Joint 1 and 2

| | |
|---|---|
| Terminal velocity: | 1.5 rps (rev. per second) (=10 rad/s) |
| Acceleration: | 0 - 0.5 rps: 80 ms (6.2 r/s2) (39 rad/s2) |
| | 0.5 - 1.0 rps: 120 ms (4.1 r/s2) (25 rad/s2) |
| | 1.0 - 1.5 rps: 200 ms (2.5 r/s2) (15 rad/s2) |

Joint 3

| | |
|---|---|
| Terminal velocity: | 20 ips (Inches per second) (=0.5 m/s) |
| Acceleration: | 0 - 5 ips: 26 ms (192 i/s2) (4.8 m/s2) |
| | 5 - 10 ips: 30 ms (165 i/s2) (4.2 m/s2) |
| | 10 - 15 ips: 48 ms (104 i/s2) (2.7 m/s2) |
| | 15 - 20 ips: 100 ms ( 50 i/s2) (1.3 m/s2) |

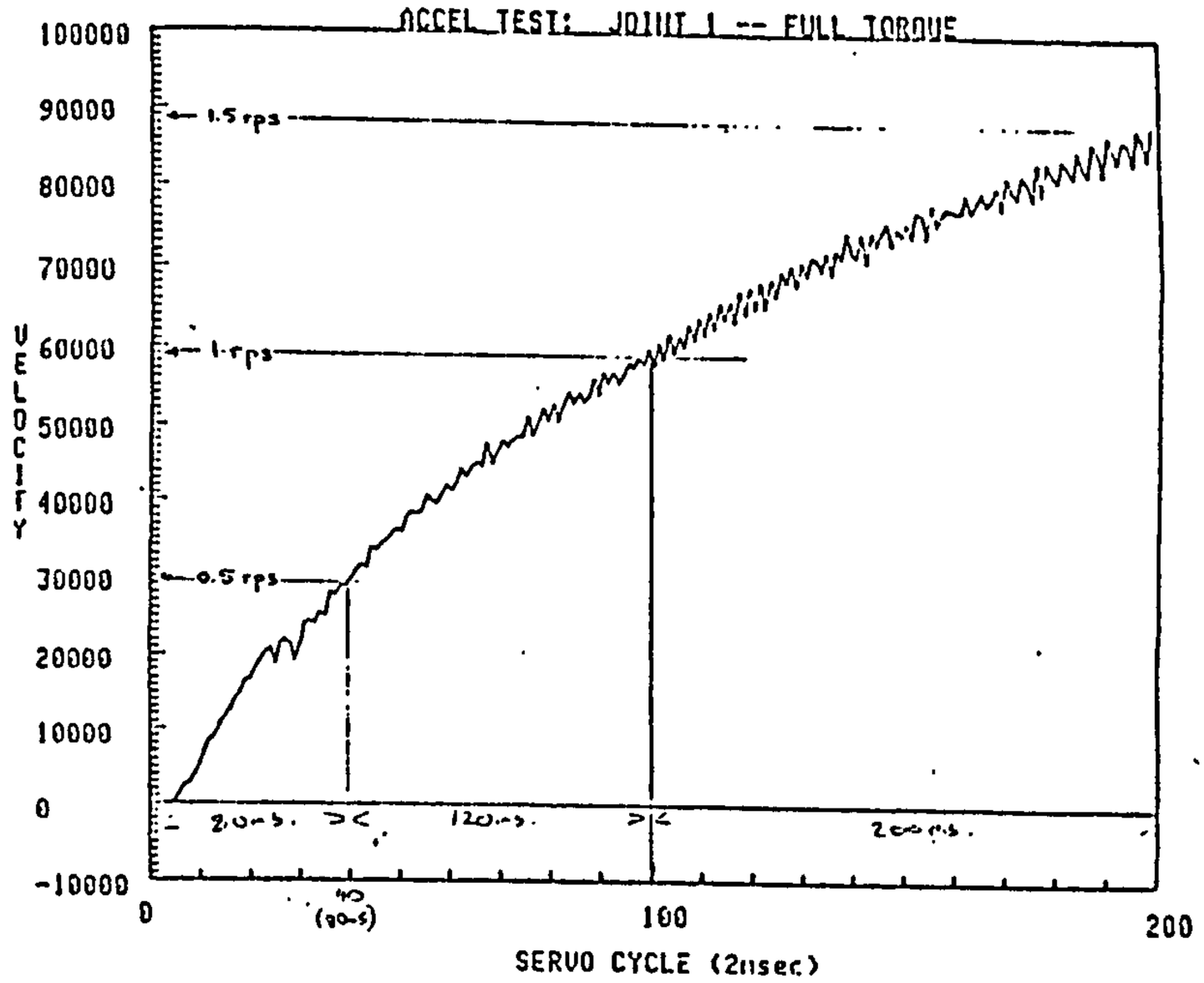
Joint 4

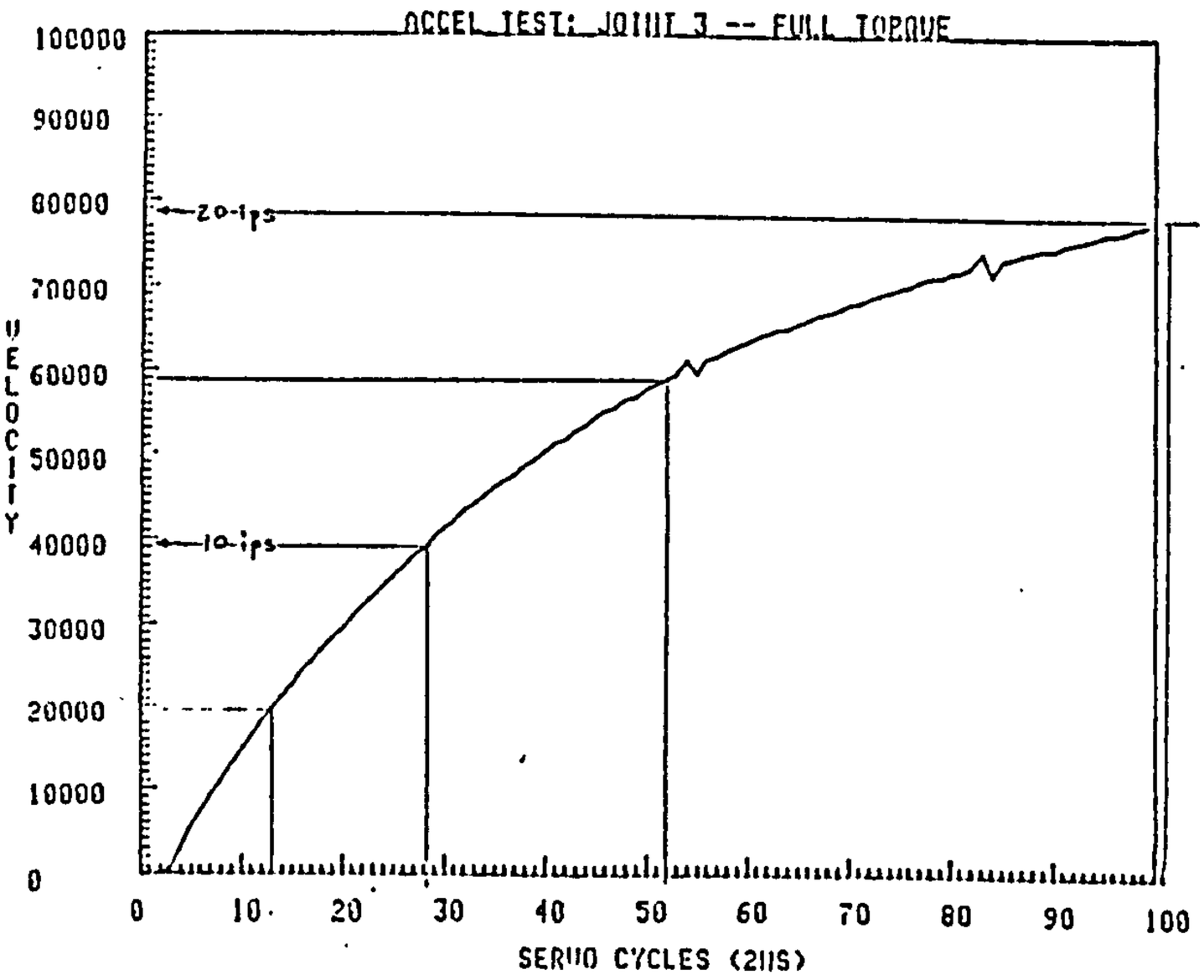
| | |
|---|---|
| Terminal velocity: | 10 rps (rev. per second) (=60 rad/s) |
| Acceleration: | 0 - 5 rps: 10 ms (500 r/s2) (3100 rad/s2) |
| | 5 - 7.5 rps: 5 ms (500 r/s2) (3100 rad/s2) |

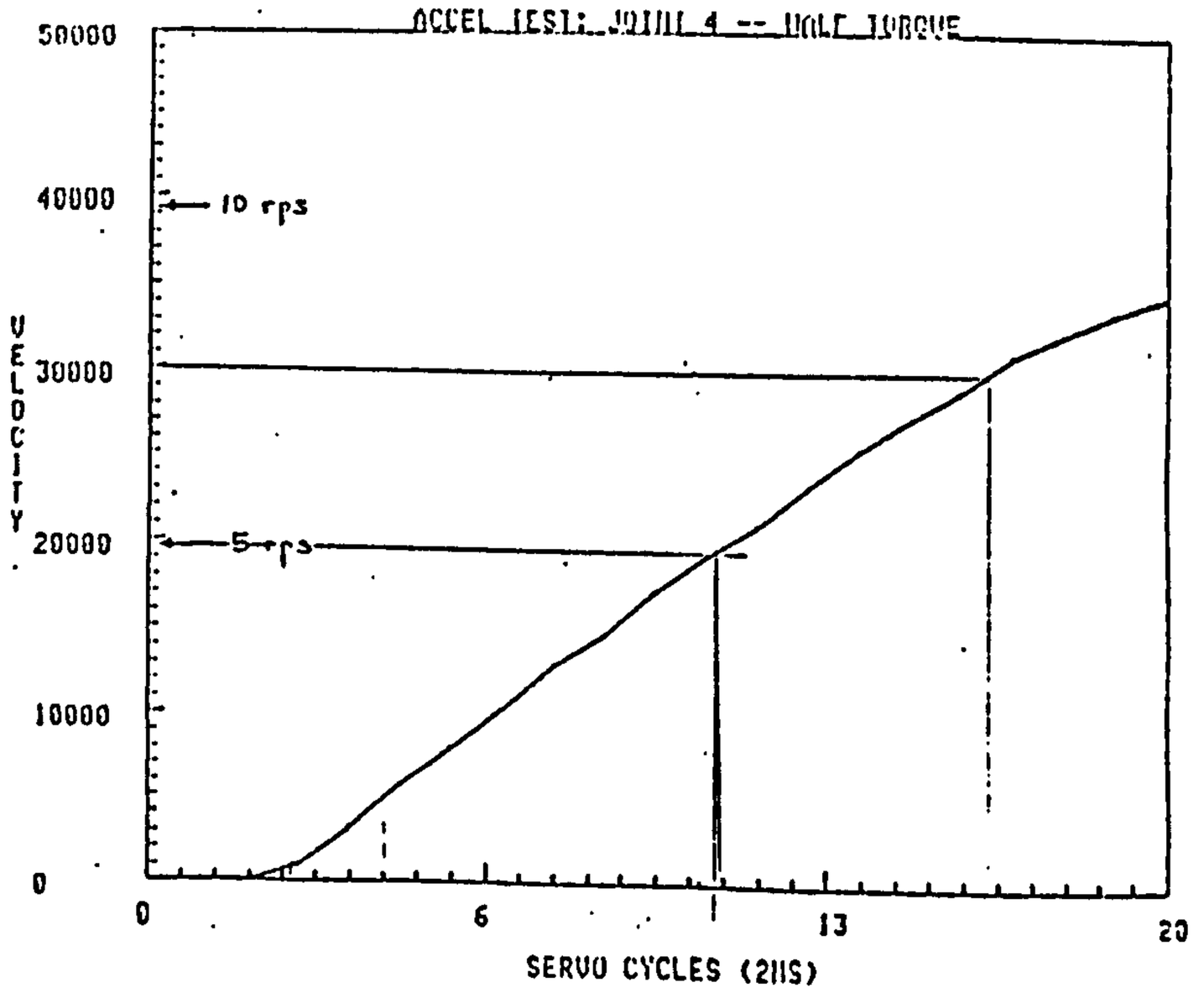
Graphs which depict the acceleration curves for joint 1, 3 and 4 are shown on the next page. Joint 1 and 2 perform in a similar manner. The joint 4 measurements were performed at half torque.

source of information: (Adept Technology Inc, June 1986)

Meta Machines Limited,
9 Blacklands Way,
Abingdon Business Park,
Abingdon, Oxford, England


ACCEL TEST: JOINT 1 -- FULL TORQUE
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
-10000
VELOCITY
1.5 rps
1 rps
0.5 rps
20 ms.
120 ms.
200 ms.
0
100
200
SERVO CYCLE (2msec)


ACCEL TEST: JOINT 3 -- FULL TORQUE
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
VELOCITY
20 ips
10 ips
0
10
20
30
40
50
60
70
80
90
100
SERVO CYCLES (2MS)


ACCEL TEST: JOINT 4 -- HALF TORQUE
50000
40000
30000
20000
10000
0
VELOCITY
10 rps
5 rps
0
6
13
20
SERVO CYCLES (2MS)

## APPENDIX C.2 ILLUSTRATIVE EXAMPLE OF VALTOGRASP CALIBRATION MODULE

The VALTOGRASP is a program used to convert any frame or teach positions (all positions are stored as a robot TOOL frame) into equivalent frame used in GRASP. Unaltered pcb frame is obtained from vision system and robot program (VAL II). These unaltered frames are stored in a location file called "filenameU.LC". This location file is transferred to PRIME via RS232 serial link. The TOOL frame is converted into GRASP frame through rotation of 180 degrees about the y axis and then 90 degrees about the rotated z axis. The final transformation matrix converted into eulerian angles and X, Y, Z translations.

The following shows the original calibrated location file called BOARDU.LC which contains the location information of the board frame obtained through the use of vision system and VAL II.

```
.LOCATIONS
 BOARD  -9.302992E-3 -0.9999567 1.46098E-5 -0.9999567 9.302992E-3 -2.161314E-
 2.147629E-5 -1.481025E-5 -1 623.2843 -87.916473 629.99499
.END
```

To run the VALTOGRASP calibration module type: SEG VALTOGRASP
the computer then prompts for the name of robot involved: ADEPTONE
the computer then prompts for the name of location file: BOARDU.LC

After locations are read from the file, the location information is then presented in a 4 x 4 homogeneous transformation matrix as below :

| | | | |
|---|---|---|---|
| -0.009 | -1.000 | 0.000 | 623.284 |
| -1.000 | 0.009 | -0.000 | -87.916 |
| 0.000 | -0.000 | -1.000 | 629.995 |
| 0.000 | 0.000 | 0.000 | 1.000 |

The rotation of 180 degrees about the y axis is transformed into a 4 x 4 homogeneous transformation matrix as:

| | | | |
|---|---|---|---|
| -1.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 1.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | -1.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 1.000 |

The robot tool frame is then multiplied by a rotation of 180 degrees about the y axis, the rotated frame becomes:

| | | | |
|---|---|---|---|
| 0.009 | -1.000 | -0.000 | 623.284 |
| 1.000 | 0.009 | 0.000 | -87.916 |
| -0.000 | -0.000 | 1.000 | 629.995 |
| 0.000 | 0.000 | 0.000 | 1.000 |

The rotation of 90 degrees about the z axis is transformed into a 4 x 4 homogeneous transformation matrix as:

| | | | |
|---|---|---|---|
| 0.000 | -1.000 | 0.000 | 0.000 |
| 1.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | 1.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 1.000 |

After rotation of 90 degrees about the rotated z axis,
the calibrated location for robot simulator becomes:

```
-1.000   -0.009   -0.000  623.284
 0.009   -1.000    0.000  -87.916
-0.000    0.000    1.000  629.995
 0.000    0.000    0.000    1.000
```

The calibrated and updated location file called BOARDU.UP which contains the location of calibrated and updated board frame through the VALTOGRASP module as shown below:

```
TO ADEPTONE ADD BOARD ( SHIFT X 623.284 Y -87.916 Z 629.995
 EULER 0.000 0.000 179.467);
```

The listing of the Prime Operating System commands generated for GRASP invokation and automatically loads in the updated locations or frames so that updating simulation model can be performed. To run this operating system command program type RUN BOARDU <name of original simulation model>, these operating command invokes GRASP and load in the original simulation model. The content of the operating system command file is explained below:

```
&args source   {enter name of simulation model                          }
GRASP          {invoke GRASP                                            }
               {two carriage returns are required                       }
               {by the GRASP system                                     }
4              {select type of graphics terminal or workstation         }
N              {not loading simulation model in binary form             }
I              {select input/output menu                                }
I              {select input data option                                }
%source%       {enter name of original simulation model to GRASP        }
               {a carriage return is required at the end of data file   }
I              {select input data option again                          }
boardu.UP      {enter name of calibrated and updated data file          }
               {a carriage return is required at the end of data file   }
WORKPLACE      {select workplace menu                                   }
ATTACH         {select attach command to change ownership of objects    }
BOARD          {object to be attached to a new owner                    }
WORKPLACE      {new owner                                               }
               {a carriage return exit the workplace menu               }
OUT            {select output option                                    }
WORKPLACE      {output model from the top hierarchy                     }
%source%.NEW   {enter name of new simulation model, use original name   }
               {with a suffix  .NEW                                     }
Y              {output tracks and paths                                 }
STOP           {finish GRASP session                                    }
N              {select no binary storage                                }
N              {do not want another session                             }
LD             {list the directory so that the programmer can check     }
               {the newly created GRASP simulation model                }
```

## APPENDIX D.1 : DIFFERENCES IN DATA FORMATS BETWEEN REDBOARD AND GRASP SYSTEMS

### (i) DATA FORMAT OF REDBOARD SYSTEM

This part of the appendix describes the data formats that are used in the Redboard system. The overall data formats are given, but only the relevant portions of the data formats are explained in full detail. Further information can be found in the Redboard menu, 1986.

```
.REM REDBOARD STANDARD DEFAULTS


.IFL                    { To start the part name data }
BOARD.IND
/.CPI
/IC1   SN7402
/IC2   SN7417
/IC3   SN7417
/IC4   SN7400
.EOD                    { To end the part name data   }


.PCB                    { To start pcb data           }


.ASS
{ Design Assignment - including pad code definitions,  }
{ track code definitions, text code definitions and    }
{ space limitation for checking.                       }

MAR 14 1                { These two lines are use by the REDAC systems other }
CMD 0                   { than ASCII input/output                            }

MAX  2                  { Defines the maximum number of layers on a board,   }
                        { number between 1 and 16                            }

UNI  40                 { Defines the structure data unit i.e. resolution,   }
                        { number in range of 32 to 127 structure data        }
                        { units (D.S.U.) per inch                            }

{ Definitions of pad code                                                    }
{ pad code, layer, pad size, pad shape, drill diameter, orientation,         }
{ finger, plated                                                             }
PAD  0 -   55 1   28 0
PAD  1 -    0 1    0 0
PAD  2 -   60 2   32 0
PAD  3 -   60 1   32 0
PAD  4 -   60 3   32 0
PAD  5 -   75 1   40 0
```

```
PAD  6 -   75 2   40 0
PAD  7 -   75 3   40 0
PAD  8 -  100 1   44 0
PAD  9 -  100 2   44 0
PAD 10 -  100 3   44 0
PAD 11 -  125 1   44 0
PAD 12 -   60 3  110 0
PAD 13 -   75 1    0 0
PAD 14 -   60 3  120 0
PAD 15 -  215 1    1 0
PAD 16 -  220 1    1 0
PAD 17 -  210 1    1 0
PAD 18 -   50 1   28 0
PAD 19 -   50 2   28 0
PAD 20 -   50 3   28 0
PAD 21 -   50 0    0 0  175 0
PAD 22 -   60 0    0 0  170 0
PAD 23 -   75 0    0 0  163 0
PAD 24 -  100 0    0 0  150 0
PAD 25 -   50 0    0 1  175 0
PAD 26 -   60 0    0 1  170 0
PAD 27 -   75 0    0 1  163 0
PAD 28 -  100 0    0 1  150 0
PAD 29 -   60 0   32 0    7 0
PAD 30 -   60 0   32 1    7 0
PAD 31 -   20 0    0 0  285 0


{ Definitions of track code                                  }
{ track width code number, layer,     track width           }
{        0-7             ,  1-16,    0-255 (0.001")        }
TRA 0 -   50
TRA 1 -   12
TRA 2 -   15
TRA 3 -   20
TRA 4 -   25
TRA 5 -   50
TRA 6 -   50
TRA 7 -   50

{ Definitions of text                                                  }
{ text size code, height of text characters, width of line             }
{      0-3      ,       0-127 D.S.U.      ,   0-255 (0.001")        }
TEX 0   2  12
TEX 1   3  15
TEX 2   5  20
TEX 3   8  25
TTS -   12          { Defines track to track spacing }
TPS -   12          { Defines track to pad spacing   }
PPS -   12          { Defines pad to pad spacing     }
```

```
.BOA
{ Definitions of board outline                                          }
{ Defines number of corners on the board outline with minimum           }
{ of 2 to maximum of 62, followed by x and y coordinates which          }
{ defines the outline of pcb. Coordinates must be positive integers     }
{ in the range of 0 to 1023 inclusive.                                  }

L  5

   0    0
 352    0
 352  500
   0  500
   0    0

.LIB
{ The component library definition defines the physical characteristics, }
{ i.e. overall dimensions and pad positions and size codes, of every     }
{ component type to be used on the circuit board.                        }

L  0    8    8   2
     2    4  3
     6    4  3
L  1   24    4   2
     2    2  3
    22    2  3
L  2   32   16  14
     4    2  3
     8    2  3
    12    2  3
    16    2  3
    20    2  3
    24    2  3
    28    2  3
    28   14  3
    24   14  3
    20   14  3
    16   14  3
    12   14  3
     8   14  3
     4   14  3
L  3   36   16  16
     4    2  3
     8    2  3
    12    2  3
    16    2  3
    20    2  3
    24    2  3
    28    2  3
    32    2  3
    32   14  3
    28   14  3
    24   14  3
    20   14  3
    16   14  3
    12   14  3
     8   14  3
     4   14  3
```

```
L  4    84     28   40
     4      2   3
     8      2   3
    12      2   3
    16      2   3
    20      2   3
    24      2   3
    28      2   3
    32      2   3
    36      2   3
    40      2   3
    44      2   3
    48      2   3
    52      2   3
    56      2   3
    60      2   3
    64      2   3
    68      2   3
    72      2   3
    76      2   3
    80      2   3
    80     26   3
    76     26   3
    72     26   3
    68     26   3
    64     26   3
    60     26   3
    56     26   3
    52     26   3
    48     26   3
    44     26   3
    40     26   3
    36     26   3
    32     26   3
    28     26   3
    24     26   3
    20     26   3
    16     26   3
    12     26   3
     8     26   3
     4     26   3
L  5    32     16    6
    12      4   5
    20      4   5
    28      4   5
    28     12   5
    20     12   5
    12     12   5
L  6    12      4    2
     2      2   3
    10      2   3
L  7   130     16   22
     2      8 26
     8      8 26
    14      8 26
    20      8 26
    26      8 26
```

```
  32     8 26
  38     8 26
  44     8 26
  50     8 26
  56     8 26
  62     8 26
  68     8 26
  74     8 26
  80     8 26
  86     8 26
  92     8 26
  98     8 26
 104     8 26
 110     8 26
 116     8 26
 122     8 26
 128     8 26


{ Index of library reference relates local library numbers to the     }
{ library numbers in the disc based library. This definition is not   }
{ reqiured for assembly.                                              }

.IDX
L  0  7102
L  1  5250
L  2  1022
L  3  1023
L  4  1136
L  5 13100
L  6  7200
L  7 12205

{ The components list preceded by .COM defines components with        }
{ reference markers. This list defines the name of component,         }
{ orientation of component name, x and Y coordinates of the position  }
{ of the component name relative to the centre of the component,      }
{ library reference number for the component type, type of pad,       }
{ orientatiion of the component layout as compared with the attitude  }
{ of the outline in the library definition (defined as 0, 1,, 2 and 3 }
{ quadrants of 90 degrees), and coordinates of the lower left hand pad}
{ after rotation (in D.S.U. measured relative to layout). Components  }
{ are grouped together according to its own heading and then the pin  }
{  number.                                                            }
{ This component list is required for pcb assembly example.           }
.COM
IC29 0   0   0 L  4 0 0   32  232
IC32 0   0   0 L  4 0 0   32  328
IC33 0   0   0 L  4 0 0   32  360
IC57 0   0   0 L  4 0 0  176  440
IC8  0   0   0 L  3 0 0    8  160
IC9  0   0   0 L  3 0 0    8  180
IC10 0   0   0 L  3 0 0   32  212
IC15 0   0   0 L  3 0 0   12  436
IC28 0   0   0 L  3 0 0   80  212
IC49 0   0   0 L  3 0 0  180   20
IC56 0   0   0 L  3 0 0  224   20
```

```
.CON
{ A connection is defined as a link between two pads.          }
{ A connection definition is not used in this assembly example }

.REM UNROUTED
.COD 2
.REM TREE    1
  IC5    25 IC3    14
.REM TREE    2
  IC5    18 IC3     6
.REM TREE    3
  IC2     6 IC5     6
.REM TREE    4
  IC3    12 IC5    23
.REM TREE    5
  IC3    10 IC5    21
.REM TREE    6
  IC5     5 IC4     3
  IC4     3 IC4     4
.REM TREE    7
  IC4     2 IC4     6
.REM TREE    8
  IC5     2 IC2     2
.REM TREE    9
  IC2    12 IC5    36
.REM TREE   10
  IC5    11 R5      1
  R5      1 C1      1
  C1      1 D1      2
.REM TREE   11
  IC4     1 R1      2
  R1      2 SW1     3
.REM TREE   12
  R3      2 IC5     1
  IC5     1 IC1     2
  IC1     2 IC1     3
  IC1     3 PL1    11
.REM TREE   13
  IC5     4 IC1     1
  IC1     1 D1      1
.REM TREE   14
  IC3    13 PL1     7
.REM TREE   15
  IC5    38 IC2    14
.REM TREE   16
  SW1     1 IC4     5
  IC4     5 R2      2
.REM TREE   17
  PL1    12 R5      2
.REM TREE   18
  PL1     4 IC3     7
.REM TREE   19
  IC2     3 PL1    18
.REM TREE   20
  PL1     6 IC3    11
.REM TREE   21
  IC3     9 PL1     5
```

```
.REM TREE  22
  PL1    16 IC2    11
.REM TREE  23
  IC5     9 PL1     8
.REM TREE  24
  PL1    15 IC2     7
.REM TREE  25
  IC2    13 PL1    17
.REM TREE  26
  PL1    14 IC5    17
.REM TREE  27
  IC5    19 PL1    13
.REM TREE  28
  PL1     9 IC5     8
.REM TREE  29
  IC5     7 PL1    10
.REM TREE  30
  IC5    26 R4      2

.COD 6
.REM TREE  31
  IC5    40 R2      1
  R2      1 R4      1
  R4      1 C2      1
  C2      1 PL1    22
  PL1    22 C3      1
  C3      1 R3      1
  R3      1 R1      1
  R1      1 IC1    14
  IC1    14 IC4    14
  IC4    14 IC2    16
.REM     10
  IC2    16 IC3    16

.COD 7
.REM TREE  32
  IC5    20 IC2     1
  IC2     1 IC3    15
  IC3    15 PL1     1
  PL1     1 C3      2
  C3      2 C2      2
  C2      2 C1      2
  C1      2 IC5    39
  IC5    39 IC5    10
  IC5    10 SW1     2
  SW1     2 IC1     7
.REM     10
  IC1     7 IC4     7
  IC4     7 IC2     8
  IC2     8 IC3     8


.REM NO DATA FOR .ROU
{ A route defines the path which must be taken by a connection  }
{ to avoid other items on the pcb.                              }
{ Route definitions are not used in this assembly example       }
```

```
.REM NO DATA FOR .COP
{ The copper list defines copper area on a layer.                }
{ The copper definitions are not used in this assembly example }


.REM NO DATA FOR .TEX
{ Text definiitions are used to define text, layer on    }
{ which it appears, orientation,, and position.          }
{ Text definitions are not used in this assembly example }

.COD 7
.EOD                    { To end pcb data }
```

(ii) DATA FORMAT OF GRASP SYSTEM

This appendix shows the GRASP syntax that describes the pcb and electronic components, assignment of pick up positions and robot program.

SYNTAX OF SOLID MODELLING

The following GRASP syntax is created for generating solid models of a pcb and electronic components.

```
POLYPRISM %BOA HEIGHT 2 AXIS Z
   0.000   0.000
   0.000 223.520
 317.500 223.520
 317.500   0.000
   0.000   0.000
;
CUBOID %4  53.340   17.780 5;
SET L4 = %4 ;
SET %IC29 = L4 ;
SET IC29 = %IC29 ( ROTATE Z -90 SHIFT Y 26.670) ;
CUBOID PTIC29 53.340 17.780 2;
SET TARG_IC29 =  ;
SET TIC29 = PTIC29
 TARG_IC29(SHIFT X 26.670 Z 2 ROTATE Z 90) ;
COPY _L4 %IC32 ;
SET IC32 = %IC32( ROTATE Z -90  SHIFT Y 26.670) ;
CUBOID PTIC32 53.340 17.780 2;
SET TARG_IC32 =  ;
SET TIC32 = PTIC32
 TARG_IC32(SHIFT X 26.670 Z 2 ROTATE Z 90) ;
COPY _L4 %IC33 ;
SET IC33 = %IC33( ROTATE Z -90  SHIFT Y 26.670) ;
CUBOID PTIC33 53.340 17.780 2;
SET TARG_IC33 =  ;
SET TIC33 = PTIC33
 TARG_IC33(SHIFT X 26.670 Z 2 ROTATE Z 90) ;
COPY _L4 %IC57 ;
SET IC57 = %IC57( ROTATE Z -90  SHIFT Y 26.670) ;
CUBOID PTIC57 53.340 17.780 2;
SET TARG_IC57 =  ;
SET TIC57 = PTIC57
 TARG_IC57(SHIFT X 26.670 Z 2 ROTATE Z 90) ;
CUBOID %3  22.860   10.160 5;
SET L3 = %3 ;
SET %IC8 = L3 ;
SET IC8 = %IC8 ( ROTATE Z -90 SHIFT Y 11.430) ;
CUBOID PTIC8 22.860 10.160 2;
SET TARG_IC8 =  ;
SET TIC8 = PTIC8
 TARG_IC8(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
```

```
COPY  L3 %IC9 ;
SET IC9 = %IC9( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC9 22.860 10.160 2;
SET TARG_IC9 =  ;
SET TIC9 = PTIC9
 TARG_IC9(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
COPY  L3 %IC10 ;
SET IC10 = %IC10( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC10 22.860 10.160 2;
SET TARG_IC10 =  ;
SET TIC10 = PTIC10
 TARG_IC10(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
COPY  L3 %IC15 ;
SET IC15 = %IC15( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC15 22.860 10.160 2;
SET TARG_IC15 =  ;
SET TIC15 = PTIC15
 TARG_IC15(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
COPY  L3 %IC28 ;
SET IC28 = %IC28( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC28 22.860 10.160 2;
SET TARG_IC28 =  ;
SET TIC28 = PTIC28
 TARG_IC28(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
COPY  L3 %IC49 ;
SET IC49 = %IC49( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC49 22.860 10.160 2;
SET TARG_IC49 =  ;
SET TIC49 = PTIC49
 TARG_IC49(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
COPY  L3 %IC56 ;
SET IC56 = %IC56( ROTATE Z -90  SHIFT Y 11.430) ;
CUBOID PTIC56 22.860 10.160 2;
SET TARG_IC56 =  ;
SET TIC56 = PTIC56
 TARG_IC56(SHIFT X 11.430 Z 2 ROTATE Z 90) ;
SET BOARD = %BOA
 TIC29 (ROTATE Z 0.000 SHIFT X 17.780 Y 146.050)
 TIC32 (ROTATE Z 0.000 SHIFT X 17.780 Y 207.010)
 TIC33 (ROTATE Z 0.000 SHIFT X 17.780 Y 227.330)
 TIC57 (ROTATE Z 0.000 SHIFT X 109.220 Y 278.130)
 TIC8 (ROTATE Z 0.000 SHIFT X 2.540 Y 100.330)
 TIC9 (ROTATE Z 0.000 SHIFT X 2.540 Y 113.030)
 TIC10 (ROTATE Z 0.000 SHIFT X 17.780 Y 133.350)
 TIC28 (ROTATE Z 0.000 SHIFT X 48.260 Y 133.350)
 TIC49 (ROTATE Z 0.000 SHIFT X 111.760 Y 11.430)
 TIC56 (ROTATE Z 0.000 SHIFT X 139.700 Y 11.430)
 TIC15 (ROTATE Z 0.000 SHIFT X 5.080 Y 275.590)
;
STOP
```

SYNTAX OF COMPONENT PICK UP REFERENCE ASSIGNMENT

The following GRASP syntax is used for assignment of electronic component to peripheral feeders (or teach positions) with component of the same type being allocated to the same feeder.

```
SET TPOINT4 =  ;
TO TPOINT4 ADD IC29 ;
TO TPOINT4 ADD IC32 ;
TO TPOINT4 ADD IC33 ;
TO TPOINT4 ADD IC57 ;
SET TPOINT3 =  ;
TO TPOINT3 ADD IC8 ;
TO TPOINT3 ADD IC9 ;
TO TPOINT3 ADD IC10 ;
TO TPOINT3 ADD IC15 ;
TO TPOINT3 ADD IC28 ;
TO TPOINT3 ADD IC49 ;
TO TPOINT3 ADD IC56 ;
STOP ;
```

SYNTAX OF ROBOT SIMULATION PROGRAM

This is a robot track generated for pcb assembly. The order of insertion sequence is based on the component type and minimum distance of travel from the current position. The initial insertion of any component type is chosen to start with a component that is to be assembled at a position nearest to the pcb frame, any subsequent insertion is based on the minimum distance of travel from the current position. This rule represents the most commonly used method. This is based on the reasoning that different componenttype can be assembled at different height and sequence, resolve foot print problems, minimises distance travelled and hence cycle time. If tooling or gripper is required to be changed for each type of component, this rule reduce the frequency at which tooling/gripper changing is necessary. This appendix shows an example result generated from the integration software modules.

```
PATH SLOW STRAIGHT  SPEED 25.00 ACCELERATION 0.00;
PATH MEDIUM STRAIGHT  SPEED 50.00 ACCELERATION 0.00;
PATH FAST STRAIGHT  SPEED 100.00 ACCELERATION 0.00;
TRACK JOBD
LOC1 : LOCATE IC29 OWNER  TPOINT4 AT TPOINT4 (SHIFT Y 18.67) ,
LOCST1 : LOCATE TIC29 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 17.78 Y 146.05) ,
LOCSTEP1 : LOCATE TARG_IC29 OWNER  TIC29 AT TIC29 (SHIFT X 26.67 Z 2.00
 ROTATE Z 90.00) ,
LOC13 : LOCATE IC32 OWNER  TPOINT4 AT TPOINT4 (SHIFT Y 18.67) ,
LOCST13 : LOCATE TIC32 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 17.78 Y 207.01) ,
LOCSTEP13 : LOCATE TARG_IC32 OWNER  TIC32 AT TIC32 (SHIFT X 26.67 Z 2.00
 ROTATE Z 90.00) ,
LOC25 : LOCATE IC33 OWNER  TPOINT4 AT TPOINT4 (SHIFT Y 18.67) ,
LOCST25 : LOCATE TIC33 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 17.78 Y 227.33) ,
LOCSTEP25 : LOCATE TARG_IC33 OWNER  TIC33 AT TIC33 (SHIFT X 26.67 Z 2.00
 ROTATE Z 90.00) ,
```

```
LOC37 : LOCATE IC57 OWNER  TPOINT4 AT TPOINT4 (SHIFT Y 18.67) ,
LOCST37 : LOCATE TIC57 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 109.22 Y 278.13) ,
LOCSTEP37 : LOCATE TARG_IC57 OWNER  TIC57 AT TIC57 (SHIFT X 26.67 Z 2.00
 ROTATE Z 90.00) ,
LOC49 : LOCATE IC8 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST49 : LOCATE TIC8 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 2.54 Y 100.33) ,
LOCSTEP49 : LOCATE TARG_IC8 OWNER  TIC8 AT TIC8 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC61 : LOCATE IC9 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST61 : LOCATE TIC9 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 2.54 Y 113.03) ,
LOCSTEP61 : LOCATE TARG_IC9 OWNER  TIC9 AT TIC9 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC73 : LOCATE IC10 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST73 : LOCATE TIC10 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 17.78 Y 133.35) ,
LOCSTEP73 : LOCATE TARG_IC10 OWNER  TIC10 AT TIC10 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC85 : LOCATE IC28 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST85 : LOCATE TIC28 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 48.26 Y 133.35) ,
LOCSTEP85 : LOCATE TARG_IC28 OWNER  TIC28 AT TIC28 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC97 : LOCATE IC49 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST97 : LOCATE TIC49 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 111.76 Y 11.43) ,
LOCSTEP97 : LOCATE TARG_IC49 OWNER  TIC49 AT TIC49 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC109 : LOCATE IC56 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST109 : LOCATE TIC56 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 139.70 Y 11.43) ,
LOCSTEP109 : LOCATE TARG_IC56 OWNER  TIC56 AT TIC56 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
LOC121 : LOCATE IC15 OWNER  TPOINT3 AT TPOINT3 (SHIFT Y 3.43) ,
LOCST121 : LOCATE TIC15 OWNER  BOARD AT BOARD ( ROTATE Z 0.00 SHIFT X
 5.08 Y 275.59) ,
LOCSTEP121 : LOCATE TARG_IC15 OWNER  TIC15 AT TIC15 (SHIFT X 11.43 Z 2.00
 ROTATE Z 90.00) ,
STEP1 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP2 (SLOW): POSITION IC29 (SHIFT Z 25 ROTATE Y 180) ,
STEP3 (SLOW): POSITION IC29 (ROTATE Y 180) ,
STEP4 : GRIP IC29 ,
STEP5 (SLOW): POSITION TPOINT4 (SHIFT Z 25 ROTATE Y 180) ,
STEP6 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP7 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP8 (SLOW): POSITION TARG_IC29 (SHIFT Z 25 ROTATE Y 180) ,
STEP9 (SLOW): POSITION TARG_IC29 (SHIFT Z 2 ROTATE Y 180) ,
STEP10 : RELEASE IC29 TO BOARD ,
STEP11 (SLOW): POSITION TARG_IC29 (SHIFT Z 50 ROTATE Y 180) ,
STEP12 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP13 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP14 (SLOW): POSITION IC32 (SHIFT Z 25 ROTATE Y 180) ,
STEP15 (SLOW): POSITION IC32 (ROTATE Y 180) ,
STEP16 : GRIP IC32 ,
STEP17 (SLOW): POSITION TPOINT4 (SHIFT Z 25 ROTATE Y 180) ,
STEP18 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
```

```
STEP19 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP20 (SLOW): POSITION TARG_IC32 (SHIFT Z 25 ROTATE Y 180) ,
STEP21 (SLOW): POSITION TARG_IC32 (SHIFT Z 2 ROTATE Y 180) ,
STEP22 : RELEASE IC32 TO BOARD ,
STEP23 (SLOW): POSITION TARG_IC32 (SHIFT Z 50 ROTATE Y 180) ,
STEP24 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP25 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP26 (SLOW): POSITION IC33 (SHIFT Z 25 ROTATE Y 180) ,
STEP27 (SLOW): POSITION IC33 (ROTATE Y 180) ,
STEP28 : GRIP IC33 ,
STEP29 (SLOW): POSITION TPOINT4 (SHIFT Z 25 ROTATE Y 180) ,
STEP30 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP31 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP32 (SLOW): POSITION TARG_IC33 (SHIFT Z 25 ROTATE Y 180) ,
STEP33 (SLOW): POSITION TARG_IC33 (SHIFT Z 2 ROTATE Y 180) ,
STEP34 : RELEASE IC33 TO BOARD ,
STEP35 (SLOW): POSITION TARG_IC33 (SHIFT Z 50 ROTATE Y 180) ,
STEP36 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP37 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP38 (SLOW): POSITION IC57 (SHIFT Z 25 ROTATE Y 180) ,
STEP39 (SLOW): POSITION IC57 (ROTATE Y 180) ,
STEP40 : GRIP IC57 ,
STEP41 (SLOW): POSITION TPOINT4 (SHIFT Z 25 ROTATE Y 180) ,
STEP42 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP43 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP44 (SLOW): POSITION TARG_IC57 (SHIFT Z 25 ROTATE Y 180) ,
STEP45 (SLOW): POSITION TARG_IC57 (SHIFT Z 2 ROTATE Y 180) ,
STEP46 : RELEASE IC57 TO BOARD ,
STEP47 (SLOW): POSITION TARG_IC57 (SHIFT Z 50 ROTATE Y 180) ,
STEP48 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP49 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP50 (SLOW): POSITION IC8 (SHIFT Z 25 ROTATE Y 180) ,
STEP51 (SLOW): POSITION IC8 (ROTATE Y 180) ,
STEP52 : GRIP IC8 ,
STEP53 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP54 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP55 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP56 (SLOW): POSITION TARG_IC8 (SHIFT Z 25 ROTATE Y 180) ,
STEP57 (SLOW): POSITION TARG_IC8 (SHIFT Z 2 ROTATE Y 180) ,
STEP58 : RELEASE IC8 TO BOARD ,
STEP59 (SLOW): POSITION TARG_IC8 (SHIFT Z 50 ROTATE Y 180) ,
STEP60 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP61 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP62 (SLOW): POSITION IC9 (SHIFT Z 25 ROTATE Y 180) ,
STEP63 (SLOW): POSITION IC9 (ROTATE Y 180) ,
STEP64 : GRIP IC9 ,
STEP65 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP66 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP67 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP68 (SLOW): POSITION TARG_IC9 (SHIFT Z 25 ROTATE Y 180) ,
STEP69 (SLOW): POSITION TARG_IC9 (SHIFT Z 2 ROTATE Y 180) ,
STEP70 : RELEASE IC9 TO BOARD ,
STEP71 (SLOW): POSITION TARG_IC9 (SHIFT Z 50 ROTATE Y 180) ,
STEP72 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP73 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP74 (SLOW): POSITION IC10 (SHIFT Z 25 ROTATE Y 180) ,
STEP75 (SLOW): POSITION IC10 (ROTATE Y 180) ,
```

```
STEP76 : GRIP IC10 ,
STEP77 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP78 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP79 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP80 (SLOW): POSITION TARG_IC10 (SHIFT Z 25 ROTATE Y 180) ,
STEP81 (SLOW): POSITION TARG_IC10 (SHIFT Z 2 ROTATE Y 180) ,
STEP82 : RELEASE IC10 TO BOARD ,
STEP83 (SLOW): POSITION TARG_IC10 (SHIFT Z 50 ROTATE Y 180) ,
STEP84 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP85 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP86 (SLOW): POSITION IC28 (SHIFT Z 25 ROTATE Y 180) ,
STEP87 (SLOW): POSITION IC28 (ROTATE Y 180) ,
STEP88 : GRIP IC28 ,
STEP89 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP90 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP91 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP92 (SLOW): POSITION TARG_IC28 (SHIFT Z 25 ROTATE Y 180) ,
STEP93 (SLOW): POSITION TARG_IC28 (SHIFT Z 2 ROTATE Y 180) ,
STEP94 : RELEASE IC28 TO BOARD ,
STEP95 (SLOW): POSITION TARG_IC28 (SHIFT Z 50 ROTATE Y 180) ,
STEP96 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP97 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP98 (SLOW): POSITION IC49 (SHIFT Z 25 ROTATE Y 180) ,
STEP99 (SLOW): POSITION IC49 (ROTATE Y 180) ,
STEP100 : GRIP IC49 ,
STEP101 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP102 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP103 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP104 (SLOW): POSITION TARG_IC49 (SHIFT Z 25 ROTATE Y 180) ,
STEP105 (SLOW): POSITION TARG_IC49 (SHIFT Z 2 ROTATE Y 180) ,
STEP106 : RELEASE IC49 TO BOARD ,
STEP107 (SLOW): POSITION TARG_IC49 (SHIFT Z 50 ROTATE Y 180) ,
STEP108 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP109 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP110 (SLOW): POSITION IC56 (SHIFT Z 25 ROTATE Y 180) ,
STEP111 (SLOW): POSITION IC56 (ROTATE Y 180) ,
STEP112 : GRIP IC56 ,
STEP113 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP114 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP115 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP116 (SLOW): POSITION TARG_IC56 (SHIFT Z 25 ROTATE Y 180) ,
STEP117 (SLOW): POSITION TARG_IC56 (SHIFT Z 2 ROTATE Y 180) ,
STEP118 : RELEASE IC56 TO BOARD ,
STEP119 (SLOW): POSITION TARG_IC56 (SHIFT Z 50 ROTATE Y 180) ,
STEP120 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP121 (FAST): POSITION PKPOS1 (ROTATE Y 180) ,
STEP122 (SLOW): POSITION IC15 (SHIFT Z 25 ROTATE Y 180) ,
STEP123 (SLOW): POSITION IC15 (ROTATE Y 180) ,
STEP124 : GRIP IC15 ,
STEP125 (SLOW): POSITION TPOINT3 (SHIFT Z 25 ROTATE Y 180) ,
STEP126 (MEDIUM): POSITION PKPOS1 (ROTATE Y 180) ,
STEP127 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
STEP128 (SLOW): POSITION TARG_IC15 (SHIFT Z 25 ROTATE Y 180) ,
STEP129 (SLOW): POSITION TARG_IC15 (SHIFT Z 2 ROTATE Y 180) ,
STEP130 : RELEASE IC15 TO BOARD ,
STEP131 (SLOW): POSITION TARG_IC15 (SHIFT Z 50 ROTATE Y 180) ,
STEP132 (FAST): POSITION PKPOS2 (ROTATE Y 180) ,
;
STOP
```

## APPENDIX D.2 : Analysis of Assembly Tolerance and Robot Accuracy

(i) Analysis of Assembly Tolerance: Pin Size Variations

| Reading no. | Measured size $x_i$ (mm) | Variations $x_i-\bar{x}$ (mm) | $(x_i-\bar{x})^2$ |
|---|---|---|---|
| 1 | 0.56 | -0.013 | 0.0002 |
| 2 | 0.56 | -0.013 | 0.0002 |
| 3 | 0.56 | -0.013 | 0.0002 |
| 4 | 0.54 | -0.033 | 0.0011 |
| 5 | 0.54 | -0.033 | 0.0011 |
| 6 | 0.56 | -0.013 | 0.0002 |
| 7 | 0.56 | -0.013 | 0.0002 |
| 8 | 0.56 | -0.013 | 0.0002 |
| 9 | 0.56 | -0.013 | 0.0002 |
| 10 | 0.54 | -0.033 | 0.0011 |
| 11 | 0.61 | 0.037 | 0.0014 |
| 12 | 0.57 | -0.003 | 0.000009 |
| 13 | 0.60 | 0.027 | 0.0007 |
| 14 | 0.60 | 0.027 | 0.007 |
| 15 | 0.62 | 0.047 | 0.0022 |
| 16 | 0.59 | 0.017 | 0.0002 |
| 17 | 0.60 | 0.027 | 0.0007 |
| 18 | 0.58 | 0.007 | 0.00005 |
| 19 | 0.57 | -0.003 | 0.000009 |
| 20 | 0.58 | 0.007 | 0.00005 |

$$\sum_{i=1}^{i=20} x_i = 11.46$$

$$\text{mean } \bar{x} = \frac{11.46}{20} = 0.573$$

$$\sum_{i=1}^{i=20} (x_i-\bar{x})^2 = 0.010818$$

$$s = \sqrt{\frac{0.010818}{20}} = 0.02326$$

$$\hat{\sigma} = \sqrt{\frac{20}{19}} * 0.02326 = 0.02386 mm$$

From the equation $\bar{x} - t^* \frac{\hat{\sigma}}{\sqrt{20}} \leq \mu \leq \bar{x} + t^* \frac{\hat{\sigma}}{\sqrt{20}}$

we can deduce the range at which the mean of the population lies at 95 % confidence interval,

t= 2.093

$$0.573 - \frac{2.093 * 0.02386}{\sqrt{20}} \leq \mu \leq 0.573 + \frac{2.093 * 0.02836}{\sqrt{20}}$$

$$0.5618 \leq \mu \leq 0.5842$$

we can also deduce the range at which the mean of the population lies at 99 % confidence interval,

t= 2.861

$$0.573 - \frac{2.861 * 0.02386}{\sqrt{20}} \leq \mu \leq 0.573 + \frac{2.861 * 0.02836}{\sqrt{20}}$$

$$0.5577 \leq \mu \leq 0.5883$$

(ii) Analysis of Assembly Tolerance: Hole Size Variations

| Reading no. | Measured size $x_i$ | Variations $x_i - \bar{x}$ | $(x_i - \bar{x})^2$ |
|---|---|---|---|
| 1 | 0.9375 | -0.01375 | 0.00019 |
| 2 | 0.9375 | -0.01375 | 0.00019 |
| 3 | 0.9375 | -0.01375 | 0.00019 |
| 4 | 0.9375 | -0.01375 | 0.00019 |
| 5 | 0.9375 | -0.01375 | 0.00019 |
| 6 | 0.9375 | -0.01375 | 0.00019 |
| 7 | 0.9375 | -0.01375 | 0.00019 |
| 8 | 0.9625 | 0.01125 | 0.0001265 |
| 9 | 0.9625 | 0.01125 | 0.0001265 |
| 10 | 0.9625 | 0.01125 | 0.0001265 |
| 11 | 0.9375 | -0.01375 | 0.00019 |
| 12 | 0.9625 | 0.01125 | 0.0001265 |
| 13 | 0.9625 | 0.01125 | 0.0001265 |
| 14 | 0.9625 | 0.01125 | 0.0001265 |
| 15 | 0.9625 | 0.01125 | 0.0001265 |
| 16 | 0.9625 | 0.01125 | 0.0001265 |
| 17 | 0.9625 | 0.01125 | 0.0001265 |
| 18 | 0.9625 | 0.01125 | 0.0001265 |
| 19 | 0.9625 | 0.01125 | 0.0001265 |
| 20 | 0.9375 | -0.01375 | 0.00019 |

$$\sum_{i=1}^{i=20} x_i = 19.025$$

$$\text{mean } \bar{x} = \frac{19.025}{20} = 0.95125$$

$$\sum_{i=1}^{i=20} (x_i - \bar{x})^2 = 0.0029$$

$$s = \sqrt{\frac{0.0029}{20}} = 0.01204$$

$$\hat{\sigma} = \sqrt{\frac{20}{19}} * 0.01204 = 0.0124 \ mm$$

From the equation $\bar{x} - t^* \frac{\hat{\sigma}}{\sqrt{20}} \leq \mu \leq \bar{x} + t^* \frac{\hat{\sigma}}{\sqrt{20}}$

we can deduce the range at which the mean of the population lies at 95 % confidence interval,

t= 2.093

$$0.95125 - \frac{2.093 * 0.0124}{\sqrt{20}} \leq \mu \leq 0.95125 + \frac{2.093 * 0.0124}{\sqrt{20}}$$

$$0.9455 \leq \mu \leq 0.9571$$

we can also deduce the range at which the mean of the population lies at 99 % confidence interval,

t= 2.861

$$0.95125 - \frac{2.861 * 0.0124}{\sqrt{20}} \leq \mu \leq 0.95125 + \frac{2.861 * 0.0124}{\sqrt{20}}$$

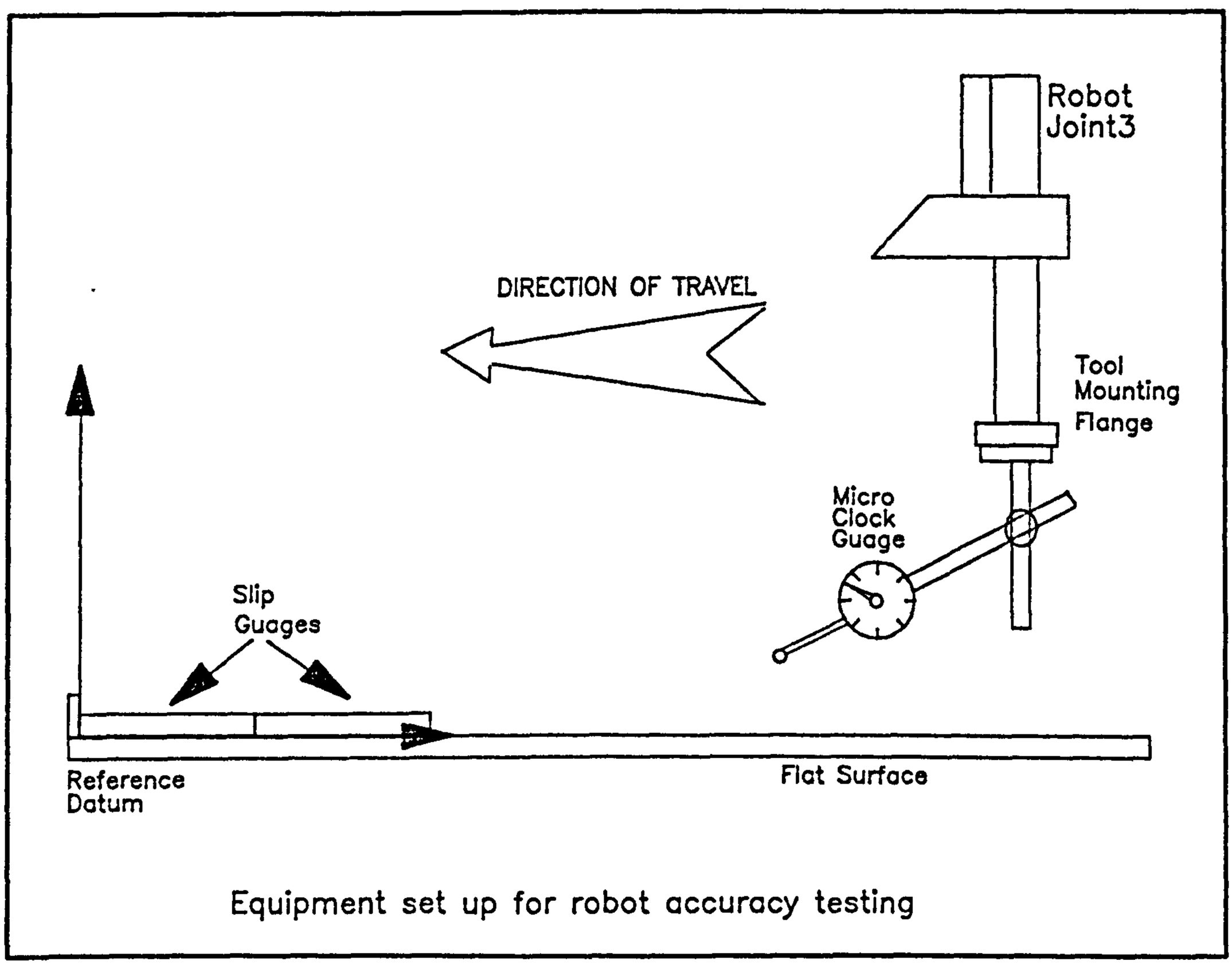
$$0.9433 \leq \mu \leq 0.9592$$

From the mean value of pin and hole size, we can determine the upper and lower assembly tolerance limits at 95% and 99% confidence intervals respectively. At 95% confidence interval, the mean value of pin size is found to lie between 0.5577mm and 0.5883mm, whilst the mean value of plated hole size is found to lie between 0.9433mm and 0.9592mm. Therefore the upper assembly tolerance limit is found to be 0.4015mm and the lower assembly tolerance is found to be 0.3550mm.

Similarly, at 99% confidence interval, the mean value of pin size is found to lie between 0.5618mm and 0.5842mm, whilst the mean value of plated hole size is lie between 0.9455mm and 0.9571mm. The upper and lower assembly tolerance limit are found to be 0.3953mm and 0.3613mm respectively.

(iii) Analysis of Robot Accuracy

Since the robot accuracy of each target position is known as normally distributed, but the robot accuracy varies over the working envelope. Robot accuracy tests were carried out at three different actual approach speeds and at different commanded translations. The experiment was aiming to provide an indication of how the robot accuracy varies at different commanded translations along one robot axis. Measured accuracy readings were obtained at corresponding commanded translations and presented in the following tables. The experimental set-up is as shown in the diagram attached.




Equipment set up for robot accuracy testing

Readings taken at actual speed of 6.25 mm per second

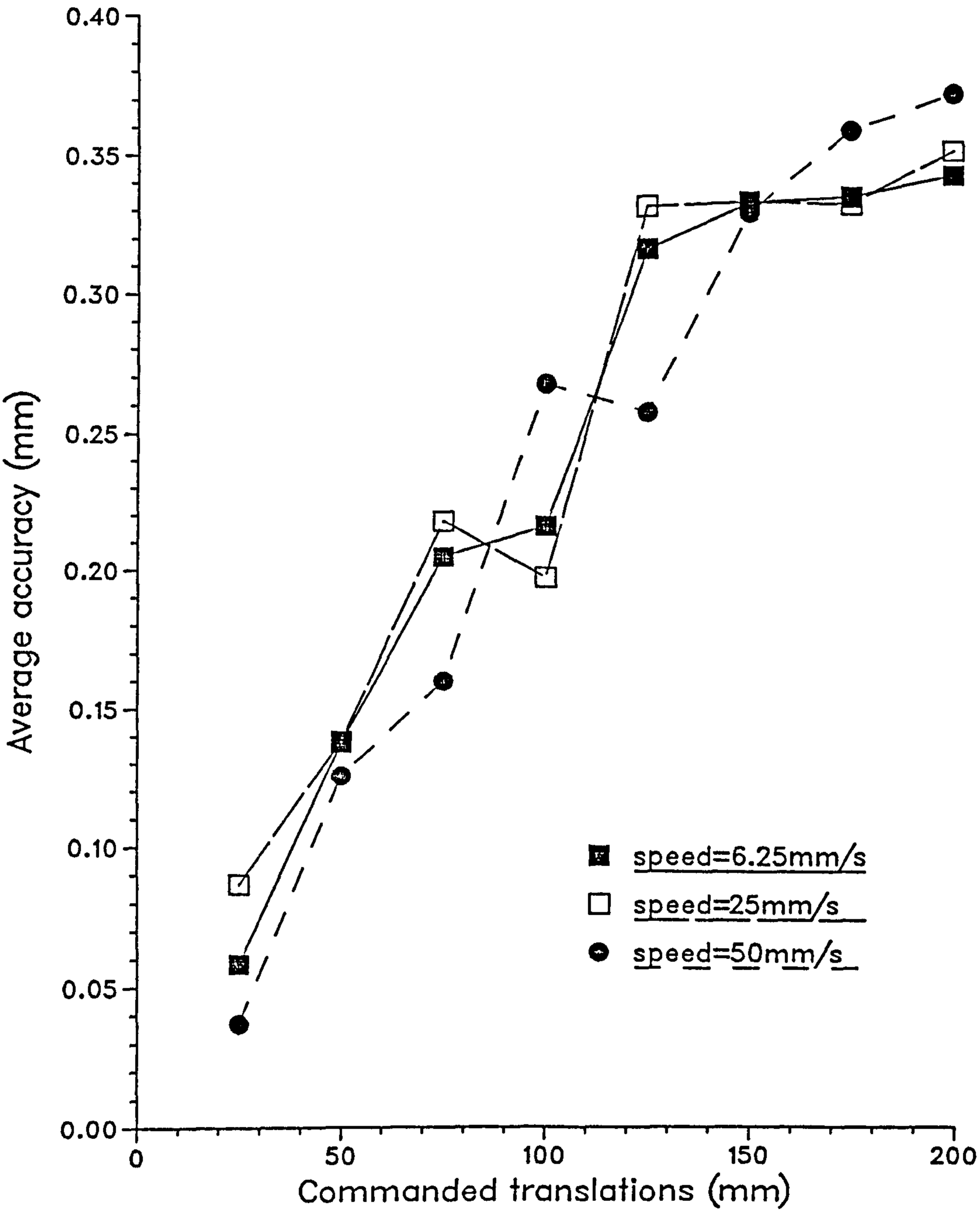
| Commanded Translation $X_i$(mm) | Variation in distance (dial gauge reading) in 0.001" | | | | | Average $Y_i$ (0.001 mm) |
|---|---|---|---|---|---|---|
| 25 | 2.5 | 2.0 | 2.5 | 2.25 | 2.25 | 58.42 |
| 50 | 5.5 | 5.4 | 5.3 | 5.5 | 5.5 | 138.18 |
| 75 | 8.0 | 8.0 | 8.25 | 8.2 | 8.0 | 205.49 |
| 100 | 8.7 | 8.7 | 8.7 | 8.6 | 8.0 | 216.92 |
| 125 | 13.0 | 12.5 | 12.25 | 12.25 | 12.25 | 316.23 |
| 150 | 13.7 | 13.0 | 13.0 | 12.75 | 13.0 | 332.49 |
| 175 | 13.25 | 13.1 | 12.9 | 13.0 | 13.5 | 334.01 |
| 200 | 13.75 | 13.5 | 13.25 | 13.5 | 13.25 | 341.63 |

Readings taken at actual speed of 25 mm per second

| Commanded Translation $X_i$(mm) | Variation in distance (dial gauge reading) in 0.001" | | | | | Average $Y_i$ (0.001 mm) |
|---|---|---|---|---|---|---|
| 25 | 3.5 | 3.6 | 3.4 | 3.2 | 3.4 | 86.87 |
| 50 | 5.5 | 5.5 | 5.8 | 5.0 | 5.5 | 138.68 |
| 75 | 9.0 | 8.25 | 8.5 | 8.5 | 8.75 | 218.44 |
| 100 | 8.5 | 8.0 | 7.5 | 7.5 | 7.5 | 198.12 |
| 125 | 13.0 | 13.25 | 13.0 | 13.0 | 13.0 | 331.47 |
| 150 | 13.2 | 13.2 | 13.0 | 13.1 | 13.0 | 332.74 |
| 175 | 13.5 | 13.0 | 13.25 | 13.0 | 12.5 | 331.47 |
| 200 | 14.0 | 14.0 | 13.50 | 13.75 | 13.75 | 350.52 |

Readings taken at actual speed of 50 mm per second

| Commanded Translation $X_i$(mm) | Variation in distance (dial gauge reading) in 0.001" | | | | | Average $Y_i$ (0.001 mm) |
|---|---|---|---|---|---|---|
| 25 | 2.0 | 0.5 | 1.5 | 1.3 | 2.0 | 37.08 |
| 50 | 5.0 | 4.9 | 5.3 | 4.8 | 4.8 | 125.98 |
| 75 | 6.65 | 6.35 | 5.55 | 6.85 | 6.25 | 160.78 |
| 100 | 10.7 | 10.4 | 10.5 | 10.5 | 10.7 | 268.22 |
| 125 | 10.1 | 10.5 | 10.2 | 9.8 | 10.2 | 258.06 |
| 150 | 12.95 | 13.0 | 13.0 | 12.65 | 13.15 | 328.93 |
| 175 | 14.25 | 14.0 | 13.8 | 14.2 | 14.2 | 357.89 |
| 200 | 14.7 | 14.5 | 14.5 | 14.8 | 14.5 | 370.84 |


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Average accuracy (mm)
0
50
100
150
200
Commanded translations (mm)
speed=6.25mm/s
speed=25mm/s
speed=50mm/s

Since the actual speed for electronic component insertion is chosen to be 50mm per second, the processing of measured data is illustrated below. Due to the non-linear and time varying characteristics of robotic system, accurate equations or models are difficult to obtain. We consider the best fit straight line that passes through these data presented in the graph. Based on linear regression method, a straight line is represented by an equation in the form of

$$Y_i = A_0 + A_1 * X_i$$

Using the least squares method, the best fit straight line can be found by solving the following two equations with two unknowns $A_0$ and $A_1$.

$$n*A_0 + A_1 * \sum_{i=1}^{i=n} X_i = \sum_{i=1}^{i=n} Y_i \text{----------(1)}$$

$$A_0 * \sum_{i=1}^{i=n} X_i + A_1 * \sum_{i=1}^{i=n} X_i = \sum_{i=1}^{i=n} X_i * Y_i \text{----------(2)}$$

where n is the total number of reading sets

$A_0$ is the unknown initial accuracy value for the graph

$A_1$ is the unkown gradient of the graph

$X_i$ is the commanded translations and

$Y_i$ is the corresponding measured accuracy value

Illustration of Working Procedure

n = 8

$$\sum_{i=1}^{i=n} X_i = 900$$

$$\sum_{i=1}^{i=n} Y_i = 1.9078$$

$$\sum_{i=1}^{i=n} X_i = 127500$$

$$\sum_{i=1}^{i=n} X_i * Y_i = 264.5022$$

substitute these data into equations (1) and (2),

$8*A_0 + 900*A_1 = 1.9078$ ---------- (3)

$900*A_0 + 127500*A_1 = 264.5022$ ---------- (4)

multipy (3) by 112.5, it becomes

$900*A_0 + 101250*A_1 = 214.6275$ ---------- (5)

subtract (5) from (4) and rearranging,

$A_1 = 0.0019$

We then substitute $A_1 = 0.0019$ into (3) and rearranging,

$A_0 = 0.0247$

The best fit straight line through the graph is described by an equation :

$Y_i = 0.0247 + 0.0019 * X_i$ (mm)

It is found that the gradient of the best fit line is 0.002 (3msd). This gradient can also be interpreted as the relationship between the robot accuracy and the commanded translation. Thus the robot accuracy is about 0.2% of the distance translated.

Since the upper and lower assembly tolerance limits were known and shown earlier, we can determine the critical assembly range beyond which assembly is considered unsuccessful based on the robot accuracy along. The critical assembly range at 95% and 99% confidence intervals can be determined.

At 95% confidence interval,

if assembly tolerance is at its upper limit,

$0.4015 = 0.0247 + 0.0019 * X_i$ (mm)

$X_i = 198.32mm$

if assembly tolerance is at its lower limit,

$0.3550 = 0.0247 + 0.0019 * X_i$ (mm)

$X_i = 173.84mm$

At 99% confidence interval,

if assembly tolerance is at its upper limit,

$0.3953 = 0.0247 + 0.0019 * X_i$ (mm)

$X_i = 195.05mm$

if assembly tolerance is at its lower limit,

$0.3613 = 0.0247 + 0.0019 * X_i$ (mm)

$X_i = 177.16mm$

Applying the same procedure to other approach speeds, we obtain equations

$$Y_i = 0.0629 + 0.0016 * X_i \quad (mm)$$

and

$$Y_i = 0.0685 + 0.0016 * X_i \quad (mm)$$

which represent the robot accuracy in terms of commanded translation for the actual speed approach of 6.25mm/s and 25mm/s. From these equations critical assembly ranges can be assessed.

APPENDIX D.3 : SPECIFICATION DATA SHEET OF THE NC DRILLING MACHINE USED

## Technical Specifications

| **Model Type** | **MM470** | **MM470L** | **MM600** | **MM600L** | **MM470DH*** | **MM470DHL*** |
|---|---|---|---|---|---|---|
| Spindle | KAVO 4025 | KAVO 4031 | KAVO 4025 | KAVO 4031 | KAVO 4025×2 | KAVO 4031×2 |
| Speed rpm | 20,000 – 60,000 | 10,000 – 60,000 | 20,000 – 60,000 | 10,000 – 60,000 | 20,000 – 60,000 | 10,000 – 60,000 |
| Power | 125 W | 450 W | 125 W | 450 W | 125 W | 450 W |
| Drill sizes | .3 – 3.5mm (.012" – .130") | .3 – 6.5mm (.012" – .25") | .3 – 3.5mm (.012" – .130") | .3 – 6.5mm (.012" – .25") | .3 – 3.5mm (.012" – .130") | .3 – 6.5mm (.012" – .25") |
| Collet size | 3.175mm (.125") | 3.175mm (.125") | 3.175mm (.125") | 3.175mm (.125") | 3.175mm (.125") | 3.175mm (.125") |
| Brushed pressure foot with integral swarf extraction. | | | | | | |
| **Table** | | | | | | |
| Work table dimensions | 485×325mm (19"×12.75") | 485×325mm (19"×12.75") | 660×485mm (25"×19") | 660×485mm (25"×19") | 660×485mm (26"×19") | 660×485mm (26"×19") |
| Maximum panel size | 560×650mm (22.6"×25.6") | 560×650mm (22.6"×25.6") | 735×950mm (29"×37") | 735×950mm (29"×37") | 2×330×485mm (13"×19") | 2×330×485mm (13"×19") |
| Programmable area | 470×310mm (18.5"×12.2") | 470×310mm (18.5"×12.2") | 620×470mm (24.4"×18.5") | 620×470mm (24.4"×18.5") | 2×310×470mm (12.2"×18.5") | 2×310×470mm (12.2"×18.5") |
| Tooling | Pin and slot 3.175mm (.125") | Pin and slot 3.175mm (.125") | Pin and slot 3.175mm (.125") | Pin and slot 3.175mm (.125") | Pin and slot 3.175mm (.125") | Pin and slot 3.175mm (.125") |
| Positioning speed | > 2000mm/min (80 ins/min) | > 2000mm/min (80 ins/min) | > 2000mm/min (80 ins/min) | > 2000mm/min (80 ins/min) | > 2000mm/min (80 ins/min) | > 2000mm/min (80 ins/min) |
| Positional accuracy | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") |
| Repeatability | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") | ± .01mm (.0004") |
| Drilling accuracy (typical) | ± .05mm (.002") | ± .05mm (.002") | ± .05mm (.002") | ± .05mm (.002") | ± .05mm (.002") | ± .05mm (.002") |
| **Z Axis movement** | | | | | | |
| Variable stroke | 1 to 12mm (.04" to .5") | 1 to 12mm (.04" to .5") | 1 to 12mm (.04" to .5") | 1 to 12mm (.04" to .5") | 1 to 12mm (.04" to .5") | 1 to 12mm (.04" to .5") |
| Feed rate | 500 – 3050mm/min (20 – 120 ins/min) | 500 – 3050mm/min (20 – 120 ins/min) | 500 – 3800mm/min (20 – 150 ins/min) | 500 – 3800mm/min (20 – 150 ins/min) | 500 – 3800mm/min (20 – 150 ins/min) | 500 – 3800mm/min (20 – 150 ins/min) |
| Hit rate | 150 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. | 150 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. | 180 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. | 180 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. | 180 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. | 180 typical for 7mm (.25") stroke over a 2.54mm (.1") matrix. |

## Physical Specifications

| | | | | | | |
|---|---|---|---|---|---|---|
| **Drill table (including base)** | | | | | | |
| Height | 1220mm (48") | 1220mm (48") | 1250mm (49.25") | 1250mm (49.25") | 1250mm (49.25") | 1250mm (49.25") |
| Width | 795mm (31.25") | 795mm (31.25") | 996mm (38") | 996mm (38") | 996mm (38") | 996mm (38") |
| Depth | 990mm (39") | 990mm (39") | 1260mm (49.5") | 1260mm (49.5") | 1260mm (49.5") | 1260mm (49.5") |
| Weight | 320kg (704 lbs) | 330kg (726 lbs) | 390kg (858 lbs) | 400kg (880 lbs) | 410kg (902 lbs) | 430kg (946 lbs) |
| **Monitor and computer** | | | | | | |
| Height | 450mm (17.75") | 450mm (17.75") | 450mm (17.75") | 450mm (17.75") | 450mm (17.75") | 450mm (17.75") |
| Width | 490mm (19.25") | 490mm (19.25") | 490mm (19.25") | 490mm (19.25") | 490mm (19.25") | 490mm (19.25") |
| Depth | 480mm (19") | 480mm (19") | 480mm (19") | 480mm (19") | 480mm (19") | 480mm (19") |
| Weight | 20kg (44 lbs) | 20kg (44 lbs) | 20kg (44 lbs) | 20kg (44 lbs) | 20kg (44 lbs) | 20kg (44 lbs) |

Source of information: Dorniver Limited,
Sanders Lodge Industrial Estate,
Welingborough Road, Rushden,
Northants, England.