THE UNIVERSITY *of* EDINBURGH

Combining Vision Verification with a High Level Robot Programming Language

YIN Baolin

Ph. D. Thesis

Department of Artificial Intelligence

University of Edinburgh

1984

Abstract

This thesis describes work on using vision verification within an object level language for describing robot assembly (RAPT). The motivation for this thesis is provided by two problems. The first is how to enhance a high level robot programming language so that it can encompass vision commands to locate workpieces of an assembly. The second is how to find a way of making full use of sensory information to update the robot system's knowledge about the environment. The work described in this thesis consists of three parts:

(1) adding vision commands into the RAPT input language so that the user can specify vision verification tasks;

(2) implementing a symbolic geometrical reasoning system so that vision data can be reasoned about symbolically at compile time in order to speed up run time operations;

(3) providing a framework which enables the RAPT system to make full use of the sensory information.

The vision commands allow partial information about positions to be combined with sensory information in a general way, and the symbolic reasoning system allows much of the reasoning work about vision information to be done before the actual information is obtained. The framework combines a verification vision facility with an object level language in an intelligent way so that all ramifications of the effects of sensory data are taken account of. The heart of the framework is the modifying factor array. The position of each object is expressed as the product of two parts: the planned position and the difference between this and the actual one. This difference, referred to as the modifying factor of an object, is stored in the modifying factor array. The

planned position is described by the user in the usual way in a RAPT program and its value is inferred by the RAPT reasoning system. Modifying factors of objects whose positions are directly verified are defined at compile time as symbolic expressions containing variables whose value will become known at run time. The modifying factors of other objects (not directly verified) may be dependent upon positions of objects which are verified. At compile time the framework reasons about the influence of the sensory information on the objects which are not verified directly by the vision system, and establishes connections among modifying factors of objects in each situation. This framework makes the representation of the influence of vision information on the robot's knowledge of the environment compact and simple.

All the programming has been done. It has been tested with simulated data and works successfully.

Contents

List of Figures and Tables

Glossary

BODY: an object which is modelled by the user in RAPT.

POSITION: a transformation in 3-D space represented by a 4x4 matrix.

SITUATION: distinct state of the world in which body positions will be specified.

ACTION: change of the world state from one situation to the next.

NOMINAL POSITION: the planned position of a body which is deduced by the RAPT reasoning system. PNbi refers to the nominal position of body b in situation i.

VERIFIED POSITION: the actual position of a body obtained by using vision verification. PVbi refers to the verified position of body b in situation i.

ACTUAL POSITION: in the thesis the term ACTUAL POSITION has the same significance as verified position.

SPECIFIED POSITION: the destination of a body which is constrained by certain spatial relationships between the body and another. The relationships must be explicitly expressed by the user.

UNSPECIFIED POSITION: the destination of a body which is not constrained by any explicitly expressed spatial relationships.

SPECIFIED ACTION: an action which moves a body to a specified position.

UNSPECIFIED ACTION: an action which moves a body to an unspecified position.

ACTUAL ACTION: movement from an actual position to another actual position.

MODIFYING FACTOR: a factor which indicates the discrepancy between the nominal position of a body and its actual one. FMbi refers to the modifying factor of body b in situation i using the prefix convention PVbi = FMbi * PNbi

Chapter 1. Introduction

The industrial robot has been in existence for some twenty years and used in real applications for a decade. During this period of time it has shown that it is an efficient automation device for performing a variety of tasks: it can work in an environment which is harmful or dangerous to people; it can work almost twenty four hours a day without rest; it is reliable in that it does not make mistakes like human beings when they are tired. The adoption of the industrial robot can raise productivity significantly and therefore more and more industries are trying to employ the robot. As robot techniques have been developing rapidly, their numbers and their applications have also been increasing. This introduces new areas of research, for example, in programming and in the use of sensory information. This thesis is on the topic of including sensory information into robot programming languages.

1.1. The Development of the Industrial Robot

The industrial robot is controlled by electronic equipment, usually computers, and has developed from two sources: the tele-operator and the numerically controlled machine tool.

The tele-operator, which is also sometimes referred to as a master-slave manipulator, is a device which allows a human user to perform a task at a distance. The first tele-operator was developed during the second world war to handle radioactive materials. The user was separated from a radioactive task by a concrete wall with one or more viewing ports through which the task could be monitored. The user would

move a "master" hand in the safe environment while the "slave" hand would copy the motion in the hostile environment. Feedback was primarily done by the user's observation. In order to sense the force which was applied to the "slave" arm, force feedback was introduced to back-drive the "master" in some later models of the tele-operator [PAU81].

The numerically controlled (NC) machine tool is a piece of automatic equipment which cuts metal under the control of digitized information. The first NC machine tool was developed in the early 1950's and made use of developing digital techniques to coordinate servo controlled axes [PAU81].

The first industrial robot appeared in the 1960's [ENG80]. It was a device which combined the articulated linkage of the tele-operator with the servoed axes of the NC machine. The industrial robot could be "taught" to perform simple jobs by driving it by hand through the sequence of task positions. These positions were recorded in digital memory. When the robot executed the specified task, the individual joint axes of the robot replayed the recorded positions.

Since the appearance of the first industrial robot, many different types of robot have been developed. They range from the simple ones (with two degrees of freedom) to the sophisticated ones such as Unimation's PUMA (with six degrees of freedom) [JAP82]. The increase of the complexity of the robot enables it to perform complex tasks. As more complex behaviour becomes possible, new techniques are required if such benaviour is to be readily specified. In order to ease the job of specifying the task for the robot, versatile programming methods have

been developed. A number of sensing techniques have also been introduced to enable the robot to adapt its performance to a change of environment.

1.2. Applications of the Robot

The computer controlled industrial robot is a general purpose automation device. It has been widely used in various tasks such as surface coating, spot welding, arc welding, machine tool servicing, forging and packaging. Some non-mechanical industries, such as the textile industry [KEM83a] and the food industry [CRO82], are also attempting to employ robots.

The combination of the industrial robot with the problem of spot welding automobile bodies was the first important application area which allowed the versatility of the industrial robot to be properly exploited [PUG82]. Robots have achieved an established role in paint spraying and their use in sensor guided arc welding has achieved an advanced development stage. In all these areas precise contact with the work piece is not essential.

In marked contrast assembly is a real challenge to the robot. Compared with the tasks mentioned above, the assembly task is more varied and complex. The assembly task usually requires the robot to have at least six degrees of freedom so that it can put workpieces at any position and orientation within a certain range of space. It requires the robot to have high accuracy so that the task can be performed with sufficient precision. It also requires powerful robot programming tools to

be provided so that complex tasks can be described easily. The motivation of much of the current research in robot programming tools and other related robotics problems is the requirement of the assembly task.

1.3. Robot Programming Languages

A robot programming language enables the user to describe an assembly task in textual mode, thus avoiding the disadvantages of teach mode discussed in Chapter 2. Robot languages are quite different from each other in syntax, structure and capability. Some languages have stemmed from NC machine tool control languages, some have evolved with the addition of robot control commands into conventional computer languages, and some have been specially designed for describing assembly tasks. In the most basic language, the user must specify the movement of each actuator of the robot for each action. In an ideal high level language, the user should only be required to indicate the starting state and the final goal state of the assembly task, the structures of the robot, the sensors available, etc; the language system should be able to decide the correct robot action sequencing. Currently, in most robot languages, the user must make a plan for an assembly task; determine a collision free path between the origin and the destination of all motions; modify assembly strategies to fit the particular geometric environment and examine the strategies. When sensory information is to be used, the language must also provide facilities for the user to describe a sensing task. Typically, the user must be able to tell the robot system when and how to obtain and use the information.

1.4. The Robot and Sensors

Theoretically speaking, a robot with many degrees of freedom can perform many kinds of task within the physical limits set by its dimensions, load capacity, mechanical tolerance and sensor resolution. However, this potential capability of the robot may be limited by an inadequate ability for sensing both the state of the environment and the robot itself and by the interpretation of sensory data. Even if the precision and repeatability of the robot is high enough to perform a certain task, a specified task may still fail if there are some uncertainties or unexpected disturbances in the environment. A remedy to this problem is to use sensors to detect the changes in the environment and in the performance of the robot.

The importance of the use of sensors was already recognized at an early stage in the use of robots. In 1961, MIT [ERN61] developed a computer controlled robot arm which was equipped with touch sensors. For this robot, tasks were defined as a sequence of touch-defined goal states. Since then more types of sensors have been introduced. These sensors can be divided into two types: contact sensors and non-contact sensors. Contact sensors include touch sensors, torque sensors, force sensors, and skin-like or hand-like tactile sensors [HAR81]. Non-contact sensors contain visual sensors and approximate range sensors. Each type of sensor can be installed to monitor different tasks.

Vision sensors are examples of an important class of sensors which provide information about the external environment of the robot. In general, vision is suitable for acquiring and orienting workpieces while contact sensors such as force sensors and touch sensors are especially

useful in detecting and removing small positional errors between parts when they are being fitted together [KEM81]. Compared with contact sensors, visual sensors work faster in locating objects since sensors (usually TV cameras) do not need to move during operation, while most contact sensors need to grope around. Visual sensors do not introduce additional disturbance to the environment since no contact between sensors and parts is made. Thus, since the first robot arm cooperated with a camera in 1970 [FEL71], vision has been used to help the robot in recognizing and locating workpieces and in guiding the robot. Vision works well in these areas although in most cases the control of the vision system and the communication between vision and the robot are performed in an inflexible way which does not permit account to be taken of relevant knowledge about the workplace.

There are two classes of method for using sensors. Firstly, the sensor is used in a closed loop which controls the state of a robot when it is executing a specified action. Methods belonging to this class usually fall into the domain of control theory and are outside the scope of this thesis. Secondly, the sensor is used to examine the environment in order to adjust the command or command sequence to be executed by the robot. For example, if the sensory information shows that a "picking-up" action has failed, then the robot can retry this action. In order to make use of the sensory information, corresponding control facilities must be provided in robot programming languages.

Accurate information is the basis of making good use of sensors. This problem is more concerned with the sensor hardware. For example, in order to acquire more accurate sensory information, touch sensors must be more sensitive; visual sensors must have nigher resolution. The

pre-processing methods for the sensory information are also important. The discussion of these is also outside the scope of this thesis. The vision part of the research work is merely used as an experimental tool and the major work of the thesis is concerned with how to specify vision tasks and how to make full use of accurate vision data.

1.5. The Motivation of the Thesis

The motivation for this thesis is provided by two problems. The first is how to enhance a high level robot programming language, such as RAPT [AMB82], so that it can express vision commands to locate workpieces of an assembly. RAPT is an object level language in the sense that it has an explicit representation of the objects in the robot's world, and their disposition in space. The second problem that this work covers is how to find a way of making full use of sensory information to update the knowledge of the robot system about its environment.

1.5.1. Specifying Vision Tasks in a High Level Robot Language

Some robot language systems provide the facility to control the acquisition and processing of sensory information. However, most of them deal with simple sensory information like touch sensor or force sensor information. The sensory information is used as a binary signal to terminate or start an action. There are some languages such as VAL [UNI79] which can control the operation of a vision system and obtain information from it, but the level of these languages is lower than RAPT. There has been no development of a high level robot language like

insert the following paragraph at the end of Section 1.5.1. in p8

In this research work, vision data is used to deal with errors in object positioning caused by feeders, inaccurate fixtures and delivery systems. It is assumed that objects are accurately represented by their models and that apart from an initial mounting error, the robot performs perfectly.

RAPT with combined vision facilities. RAPT is an object level language which has knowledge about the assembly task. The work to be described here shows how such knowledge can be combined with sensory information in an intelligent way.

This research work combines a special kind of vision, verification vision [BOL77], with RAPT. Verification vision is used when there is already expectation about the objects to be seen and their layout. In order to specify vision tasks, a set of vision commands have been added to the RAPT input language. These commands allow the user to describe how he wants to start a vision task. They also allow the user partially to predict how to interpret the information. Using the information provided by the associated assembly program, the vision system works out details of vision tasks, such as which areas of the image should be examined. Subsequently, the vision system reasons about the vision information in order to find out the exact position of the object to be verified. This reasoning is done at RAPT compile time in order to make the run time system work fast. Since the vision information cannot be obtained until run time, the reasoning is symbolic. At run time, when relevant vision information is available, the symbolic result of the reasoning can be evaluated and the actual position of the verified object can be decided.

1.5.2. Intelligent Use of Vision Information

One of the most important problems with sensory information in programming and controlling the robot is in planning how to use the knowledge obtained from the information to adjust the subsequent actions

insert the following paragraph at the end of line 17 in p9:

As the work uses the robot language RAPT as its vehicle to achieve its goal, the limitations of RAPT impose constraints on the work. For example, the incompleteness of the RAPT modelling system restricts the capability of the vision system so that the user has to specify the vision tasks in greater detail than with a full solid modelling system. The lack of a flow control facility in RAPT limits the method of using vision information. For example, the vision system can only adjust the planned actions and cannot choose between alternative actions. In spite of all these constraints, the thesis work successfully combines a vision facility with RAPT in a general way.

of the robot. For instance, if a touch sensor touches an object at an unexpected position, what should the robot do? If the sensory information shows that the position of an object is different from the planned one, what could the robot system know about the effect of the position discrepancy on the assembly task and how should the robot adjust the following actions? At the moment, most strategies for using sensory information are very simple. For example, the robot manipulator may simply stop a specified movement when a touch signal is received. Although vision can provide a lot of information about the environment, in most current robot systems the positional information is used to update the knowledge of the robot system in relation to the corresponding object only. No consideration of the effect on other objects in the same environment is made. In the work to be described in this thesis, a new approach is introduced to make full use of the information obtained from the verification vision so that the knowledge of the robot system about the environment, rather than that of a single object only, can be updated in the light of this information.

1.6. Structure of the Thesis

This thesis is structured as follows:

Chapter 2 reviews the two main modes of robot programming: the teach mode and the textual mode, discusses the classification of robot programming languages and the control facilities of sensory information in these languages.

Chapter 3 discusses details of a high level robot language RAPT, on

which the research work of the thesis is based. These details include the input language and the geometrical reasoning system of RAPT.

Chapter 4 surveys the development of computer vision systems and their application in robotics.

Chapter 5 discusses the new vision commands which enable the user to specify the vision verification task in RAPT.

Chapter 6 discusses a symbolic reasoning system which is used by the verification vision.

Chapter 7 discusses the principle of the framework which handles vision information and describes how the information influences the knowledge of the robot about the actual positions of objects. This chapter also discusses the possible ways of adjusting the actions of the robot at run time taking account of the vision information.

Chapter 8 establishes the rules for making and simplifying the modifying factor array which is the heart of the framework described in Chapter 7.

Chapter 9 describes the implementation of the vision system and discusses the experimental results, and concludes the research work.

Finally, Chapter 10 lists some work to be done in the future and discusses the possible ways of realizing this.

Chapter 2. Robot Programming Languages

Robot programming is an important process in which the user specifies the tasks that he wants the robot to do. During the programming process, the user has to provide sufficient information for a robot executable specification to be produced, which will cause the robot system to perform the desired tasks. Various programming methods have been developed to provide efficient programming tools. The power of a programming tool is judged by a number of attributes. These attributes include ease and economy of programming, facilities for interaction between the user and the robot at programming time, facilities for interaction between the robot and its working environment, etc. As there is a trend to use robots to fulfil more and more complex tasks, more advanced programming tools are needed: the formal robot command language is an important one among them.

2.1. Two Modes in Programming Robots

There are two major approaches in programming robots. One is referred to as teach mode programming. It is also known as programming by showing or guiding [LOZ82]. Teach mode programming has been used in programming industrial robots since their introduction in the early 1960s, and is well developed. Most commercial robots are equipped with some facilities so that they can be programmed in teach mode. The other approach is referred to as textual programming and this makes use of some formal language. There is a lot of interest in the development of textual programming both in academia and in industry because it adds to the versatility of robot use.

2.2. Teach Mode

In teach mode programming the robot arm is positioned at various places through which it must pass while performing the task. Small manipulators with back drivable actuators are usually pushed and pulled through moves. Medium manipulators are usually moved by the use of buttons, keyboards, etc. Large manipulators which are difficult or dangerous for the programmer to control directly are often equipped with "teaching arms" which can be used to learn the positions and motions [KEM81]. Sometimes these techniques can be used together. This position and/or motion information is recorded by the robot system. At execution time, the robot arm repeats the motions through the positions which it has been taught. Therefore, this mode of programming is also referred by some authors (e.g. [TAY76]) to as tape recorder mode.

The information acquired in teach phase can be stored either on a magnetic tape or in computer memory. The basic data is of the position. However, information about velocity and trajectory can also be recorded. When teaching a trajectory, the programmer leads the robot arm moving through the specified trajectory at a real speed or a proportionally ratioed speed. The positions of the robot arm are recorded as a function of time. At execution time, the robot arm moves under the control of a clock so that the trajectory and velocity which have been taught can be repeated. Changing the frequency of the clock can speed up or slow down the execution of the robot arm.

The main advantage of teach mode programming is its simplicity. It

is easy to do and its results can be perceived directly during programming. Also, it does not need powerful computational facilities. However, as Koutsou [KOU81] among others pointed out, its disadvantages cannot be disregarded. This approach of programming makes the fundamental assumption that the task being programmed can be described adequately by specifying a sequence of absolute positions. Therefore it relies on a very high degree of repeatability of the manipulator and usually needs to use some specially designed tools or feeders to guarantee the accuracy of the positions of the workpieces to be manipulated. It is difficult to use sensory data to interact with the world and therefore the operation of the manipulator is not adaptive to changes in the work station and the robot cannot deal with unanticipated events. The most likely sensory data which can be used in teach mode programming are some switch signals which can start or stop a sequence of operation. This interaction can provide a degree of synchronization between the robot and other machines. There are no flow control facilities such as branches or loops available in this mode of programming so that the "program" cannot express anticipated alternative operations. This method of programming is error prone for complex tasks and difficult to edit. It is inefficient in terms of human effort since programs produced in this way are not general: usually no programs or their parts designed in this mode for one task can even be re-used for another similar task. For example, suppose the user wants the robot to pick up ten bolts which are arranged in a row on a table with every adjacent pair the same distance from each other. Although some systems have a limited palleting ability, in most cases the user has to teach the robot the ten similar actions with each action having only a small difference from the previous one. This method is inefficient in terms of equipment utilization since the method is on-line, and during the programming period, the

robot arm is used as an information collector and therefore cannot do its real job. The lack of the capability of using sensory data prevents this method from making available the full capabilities of the robot.

In order to overcome some of the problems of pure teach mode programming several improved methods, such as the augmented teach mode method [TAY76] and off-line guiding mode programming [LOZ82], have been suggested and developed. For example, in augmented teach mode, a number of "built-in" functions can be added into the system. These functions provide the user with some commands to specify differential motions and use some simple sensors. Thus the robot can be taught to use touch sensors to determine the location of some object to be manipulated. After the object is located, subsequent motions are made relative to the object's coordinate frame. This behaviour can be referred to as a "search" action. The built-in functions can also provide some primitives for the user to specify some common tasks. Furthermore, some systems also provide simple control structures [LOZ82], although the capabilities of the control structure are limited. The augmented teach mode retains many of the advantages of pure teach mode. The principal new advantages offered by the technique are:

1. More tasks can be performed.
2. Absolute accuracies need to be less, thus reducing fixturing costs.
3. Programs may be shortened, since some of the special functions may include several motions.

The main disadvantages of this mode in addition to those of pure teach mode are:

1. Augmented mode needs a better programming understanding on the part of the user.
2. It requires a more powerful run time system than pure teach mode.

In off-line guiding mode, the CAD model or mockup of the task together with a robot model can be used to define the program. The system simulates the motions of the robot in response to a program or to guiding input from a teach device. Since this mode is off-line, it is efficient in terms of equipment utilization. It is also safer than pure teach mode for both the user and the equipment.

Although the modes described above make some improvements to teach mode programming, they are still teach mode. Therefore they retain the main disadvantages of pure teach mode. For example, it is difficult to use variables and adopt a complex control structure. These improvements are rather limited and cannot solve most of the important problems which are inherent in teach mode.

2.3. Textual Programming

Textual programming is usually referred to as off-line programming though it does not necessarily preclude on-line use of the robot manipulator during the programming phase. This mode of programming produces programs as text written in formal languages. Like ordinary computer language programs, these textual robot programs are easy to edit and correct by the use of various programming support facilities. It makes communication between robot programs and a CAD/CAM data base possible.

Some common parts of programs can be used in several different tasks simply by means of editing or subroutine call. This style of programming is therefore more efficient than teach mode in terms of human effort. It is also more efficient in terms of robot utilization since the robot is not involved in the programming phase and can therefore remain in operation while a new program is being written. Ideally, the user both writes and checks his programs in off-line mode. He can simulate robot operations in order to check the correctness of his programs before running them. This mode of programming also makes it possible to use sensory information so that changes in the environment can be detected and the robot can adapt its operation to the new conditions. This method of programming, therefore, has the capability of dealing with more complex environments and tasks than can teach mode programming. The main disadvantages of this approach are that it requires more run time support and off-line computer power. It also requires the user to have specialized programming knowledge. However, these problems are becoming less important since progress in micro-electronics is bringing about cheaper computing facilities and research on robotics is producing languages more friendly to the human user. Meanwhile, the advantages of the textual programming method are becoming more essential as the complexity of the robot environments and tasks increases. Thus, the textual programming method has better prospects.

In some robot languages, such as VAL [UNI79], the robot may be used on-line during the programming phase in order to obtain positional information for textual programs. Theoretically speaking, this is not indispensable for programming in these languages. On the other hand, it increases the flexibility of the programming method and sometimes may be convenient to the programmer.

2.4. Classification of Robot Command Languages

Robot programming languages can be classified into different levels depending upon the subjects on which the programmer focuses his attention in describing tasks. Different schemes exist for the classification. One of the most commonly used classification was first proposed by Latombe [LAT79, KEM81, KOU81]. In this scheme, robot languages are divided into four levels. They are

1) objective level,
2) object level,
3) end-effector level,
4) actuator level.

The objective level is the highest conceptual level. In languages at this level the task would be described in terms of the final objective, and the programmer would only need to provide the robot system with knowledge about workpieces and assemblies, and then tell the robot system which kind of product he wants. The robot system would then be able to plan the whole task, decide the sequence of operation, choose proper tools, select a collision avoiding trajectory for the robot manipulator, and so on in order to control the robot to achieve the final objective. The realization of languages at this level needs more research work done in areas such as problem solving and planning, and in applying such concepts to the field of industrial robotics. There are no languages at this level existing at the present time.

Object level languages are the next level down. At this level the programmer concentrates his attention on operations on the objects. He does not have to worry about the position and actions of the robot manipulator and the details of the manipulation. What he needs to specify is the state of objects to be manipulated at each step of an assembly task. Objects and the environment are represented symbolically in programs. These symbolic representations are referred to as models. The task is described by defining the positions and moves of the objects which are to be handled by the robot. This information is then converted by some computational system into run time commands that the robot can obey. It is the user's duty, at this level, to plan the sequence of the operations in the task, to take into account the necessity for avoiding collisions and to specify some other details of the task. However, all these can be done at the object level in a way which is natural to the human user. Some robot languages which can be classified in this level have been developed. Some well known examples are LAMA [LOZ76], AUTOPASS [LIE77], RAPT [POP78] and LAMA-S [FAL80]. Almost all languages at this level are still in the laboratory development stage.

End-effector level languages are also referred to as manipulator level ones. In languages at end-effector level the task is described in terms of the displacement and operation of the end-effector. At this level, the robot system has little knowledge about the objects which it handles and the environment in which it works, and therefore the user has to plan the task in great detail, such as the routes of the moves of the end-effector, opening and closing of the gripper and so on. The robot system knows about the kinematic structure of the robot and so the positional information of the end-effector can be transformed into

specifications of the actions of each joint and actuator of the robot. Some languages at this level make use of the "frame" to represent positions of objects. The frame is a local coordinate system. It provides a way for describing the positions of properties such as holes and some special parts of an object in terms of relative positions and gives the robot system limited knowledge of the objects. Knowing the position of a frame and relative positions of properties, the robot system can deduce the positions of the properties of the corresponding object in terms of absolute coordinates. Sometimes the programming at this level takes place on-line in order to use the manipulator to collect positional information for describing moves. A number of languages belonging to this level have been built up, such as WAVE [PAU77], VAL [UNI79], EMILY [WIL75], LM [LAT81], AL [FIN75] and MAPLE [DAR75]. Most current commercial robot languages can be classified into this level.

In languages at actuator level, the programmer has to consider the sequence of actions of each joint and actuator of the robot in order to specify the operation of the robot. Languages at this level are designed for instructing particular robots and usually used by higher level language systems to generate robot executable commands.

Some authors (e.g. Kempf [KEM81]) classify the teach mode programming as programming by the use of actuator level languages. However, the author can not agree with this: his argument is that although most teach mode programming produces records about the actions of the robot actuators, it usually does not create any forms of textual results in terms of formal languages, and the programmer's attention is mainly focused on the position and operation of the end-effector of the robot rather than of the actuators. Furthermore, in some teach mode

programming systems the data to be recorded can be of the positions of the end-effector of the robot rather than of the actions of actuators [TAY76]. In this case, there are no direct connections between teach mode and the actuator level programming at all.

2.5. Sensory Information in Robot Languages

The versatility and adaptability of robots can be greatly increased if they are able to make use of sensory information. In order to inspect the performance of the robot and adapt the task specifications to the environment when the robot executes the task, various kinds of sensors must be used. Some authors [WAN76] think that in order to make full use of the robot, for example, in order to perform an assembly task in which the planned path of the manipulator may be cluttered, the following sensors could be usefully, if not necessarily, employed:

1. A gripping force sensor for controlling gripping force.
2. A multi-degree of freedom force sensor to resolve externally applied forces and moments.
3. A whisker sensor to test if the gripper is touching anything.
4. An area tactile device to show the shapes of patterns produced on gripping an object and to detect slippage of the parts.
5. A "range finder" or a "clear path ahead" sensor.
6. An optical surveillance sensor to analyse the assembly environment.
7. Safety sensors which trigger when any object interferes with a certain space around the manipulator.
8. Position, velocity and acceleration sensors for position and

dynamic motion control of the manipulator system.

Of course, the list is endless. For example, vision recognition and verification may also be needed. New requirements may emerge, and new technologies may provide better solutions for each requirement, either old or new.

In order to control these sensors in textual robot programming, robot languages must provide suitable means for the user to specify when and how to use these sensors, and how the behaviour of the robot should be modified. However, the control of sensory information is still a difficult problem in robot language design. No current robot language has been able to specify systems to control so many kinds of sensors successfully. Most languages which have the capability to use sensory information control only one or two kinds of sensors in a relatively simple way. They are not able to take full advantage of sensory information to update the knowledge of the robot system about the environment.

A number of robot programming languages have been reported using sensory information. The languages concerned include those of both end-effector level and object level. The sensor types which have been dealt with involve force, torque, contact and vision.

2.5.1. Sensory Information as Binary Signals

The most commonly used sensory information is of force or torque sensors. This kind of sensory information is analogue. However, some

languages which handle this kind of information, such as AL [FIN75], LM [LAT81], AUTOPASS [LIE77], use the information as a binary signal. In these languages, the user can specify a threshold value for an expected sensor data. This specified value can be used either as a termination condition or as a constraint description of an action.

In contrast to this, WAVE [PAU77] uses binary type sensory information in a more complex way. In the WAVE system, the user's programs are expanded at planning time into position dependent coordinated motions. Dynamic, configuration dependent effects are pre-computed since only minor deviations from the plan are expected during execution. Sensory information is adopted in order to detect the deviation and modify subsequent sections of the plan to eliminate further differences. The gripper of the robot which WAVE controls has a binary touch sensor on the inside of each finger tip. When a command CENTER is used to adjust the position of the gripper with respect to an object, the fingers start closing, the status of the touch sensors being monitored. If one finger sensor touches the object, then the gripper starts to translate towards that side at such a rate as to keep that finger fixed in space while still closing the fingers. When the second sensor touches the other side of the object, the gripper stops translating and the fingers continue closing until the required force level is met. Hence the CENTER command provides a method to perceive actual positions of objects. It also provides the necessary compliance between the robot and the world. The user can use other commands to specify the modification of the subsequent plan of actions. Of course, the touch sensor information can also be used as the condition to terminate some actions.

LAMA [LOZ76] also uses force or torque sensor data. The techniques

that LAMA adopts to deal with sensory information are distinct from the other systems. In the LAMA system a feedback planner is used to expand the user's program into manipulator programs. The feedback planner simulates, incompletely and qualitatively, the assembly operations, considers the effects of each operation such as whether a contact will happen and so on, and roughly decides what the force or torque sensory value will be. The information provided by the simulation is necessary in order to evaluate the feasibility of a proposed assembly operation and simplify the interpretation of the feedback information obtained during execution. The sensory data is used to detect whether a specified action terminates at a correct position or not. If not, an error is reported.

2.5.2. Vision Information

Visual information can also be handled by some robot languages. For example, WAVE can use visual feedback to detect the discrepancies between a planned position and an actual one, and guide the gripper to achieve the correct position. In the work reported by R. Goldman [GOL77], a vision system cooperated with the AL system to fulfill the verification task. Whenever the manipulator program needed visual feedback it signalled the verification vision system. The vision system then took a picture and computed the needed information which was stored into the appropriate variable in the manipulator code. This vision system could be used to check whether a specified operation, e.g. picking up a workpiece, had succeeded or not. In the case of failure of an operation, the vision system could signal the manipulator and ask it to re-do the operation. The vision system could also be used to determine

precise locations of some features, e.g. holes, of a workpiece.

Some vision commands have been added to the basic VAL language to enable it to use a Unimation Inc. UNIVISION system. A UNIVISION system consists of a Unimation VAL-controlled industrial robot and a Machine Intelligence Corporation (MIC) VS-100 vision system. The vision system uses a binary picture taken by a TV camera and produces a two-dimensional interpretation of the picture in order to recognize objects and locate them. Before the vision system can identify objects, it must be trained by showing it prototypes and names of objects. During the training stage, the system learns about features of prototypes of objects, such as area, perimeter, minimum and maximum radii and so on. The recognition is done by comparing the blobs in the picture with the characteristics of the trained prototypes in an effort to find a match. The location of objects is done by calculating the centroids of images of the objects. The vision system determines the orientations of objects via a user-selectable criterion such as the angle from the positive X-axis of the camera reference frame to the largest radius. This location information is then used to update a position variable of the user's program in order to control the movement of the manipulator.

MCL [BAU81] is an end-effector level language. It consists of substantial additions to the basic capabilities of the numerical control language APT. These additions allow the user to direct the operation of industrial robots and a variety of sensory devices as well as standard NC machines. MCL can control two classes of sensors: "simple" sensors and "complex" sensors. Simple sensors include contact sensors, temperature sensors and a variety of proximity sensors. The common characteristic of these devices is that they all perform a simple monitoring

function for a single physical property and return a scalar value for the property. Complex sensors, on the other hand, perform a relatively large amount of local computation in order to detect features, recognize patterns, and compare observed input with internal expectations. A vision sensor is an important complex sensory device which can be controlled by MCL. Control of a vision system in MCL involves two processes. The first is to model the set of views of the part which may be seen by the vision system. MCL uses image models which are defined as sets of two-dimensional regions. Each region represents a basic component of the part view such as a visible hole. A combination of regions defines a complete model of a part. The second process is to use the model to either locate a part or inspect a region in an image.

2.5.3. Sensory Information in Describing Servo Process

In RSS [GES83], sensory information is used to define servo processes rather than to detect objects in the environment of the robot. RSS is an end-effector level language. It is different from most other robot languages in that it does not contain statements which command the robot to perform some actions. Instead, the programmer must declare servo processes which cause the robot to perform that action. RSS permits servo processes to be defined for each of the four major aspects of manipulation: position, orientation, force and torque. A condition monitor accesses sensory information in order to inspect conditions defined by the user program and to decide corresponding actions of the manipulator. In the RSS system, data from external sensors are expressed as dynamically changing functions, identical to functions which refer to the robot state. The functions R$FORCE and R$TORQUE evaluate to vectors

which contain the external force and torque at the robot wrist, as determined by an estimator by observing the joint velocity and knowing the analog servo set-point. Positional data is obtained from a separated vision system which, from the robot's point of view, is merely a sensor which accepts simple commands and returns the three-dimensional position of objects it sees. The RSS system is not concerned with how objects or features are located. It requires the vision system to contain all necessary algorithms to locate objects. The commands that can be used to access the vision system are: the VISION statement, which declares a vision function whose value is determined by the vision system, the LOCATE statement, which asks the vision system to determine and update the corresponding location of a declared vision function, and the TRACK statement, which causes the vision system to continually update the value of a declared vision function on the basis of the stream of images it is processing.

2.5.4. Summary

The languages which have been investigated above represent different kinds of robot programming languages. VAL, WAVE, AL, RSS, MCL and LM can be classified as end-effector level languages, and LAMA and AUTOPASS are at object level. Among these languages VAL is a well known commercial language while others are still in laboratory stage. The methods used by these languages to deal with sensory information are also typical. Except RSS in which sensory information is used to monitor the inner state of the robot, all other languages use sensory information to detect the environment of the robot. In these languages, the user has to specify not only when but also how to use the sensory information.

The force and torque information is usually used as a condition in a decision tree while the vision information is used to identify objects, and instantiate and update some position variables. No languages have tried to handle complex tactile sensors with a matrix of elements. In most cases the sensory information is used strictly in the way the user has explicitly pointed out, such as terminating an action or updating the position of an object. No further explanations of the sensory information are made. Vision information has only been handled by end-effector level languages. Most vision systems used in these languages only work in a way to compare two-dimensional images with two-dimensional image models in order to locate an object or inspect a region in an image. Languages at this level have very limited knowledge about the robot environment so that they can not give vision systems enough guidance of how they should work and how to interpret the vision information. Object level languages, although having rich knowledge about the environment, have not been combined with vision systems properly.

Chapter 3. The Current RAPT System

This chapter is devoted to RAPT, an object level robot language, on which the main work of the thesis has been based. Since a knowledge of RAPT is essential for understanding the major part of the research work described in the thesis, the principal features of RAPT, its input language, internal data structures and reasoning systems will be discussed in detail.

3.1. Main Features of RAPT

RAPT has been developed in the Artificial Intelligence Department of Edinburgh University since the late 1970s by R. J. Popplestone, A. P. Ambler and their group. It is a model-based object level robot command language designed mainly for programming assembly operations. This means that in RAPT programs the objects which are to be manipulated by the robot are explicitly represented. Its syntax is quite similar to that of APT, a widely used NC machine tool programming language, and the name RAPT stands for Robot APT.

RAPT allows the user to specify a set of objects (bodies), spatial relationships and movements. Objects are specified by the use of its modelling system. Spatial relationships which are to hold between features of the objects in each distinct state are specified by the use of relationship specifications. Movements of the objects can be specified explicitly by the use of action statements or implicitly by the use of relationship constraints imposed upon objects. All the information is transformed by the RAPT system into specifications of the movements

of the robot which will bring about the desired states. The work of the RAPT system is divided into two stages: compile time reasoning and run time execution. Since RAPT is an off-line programming language, the major part of the computational work is done at compile time, and it is only this part which will be discussed in this chapter.

The compile time system of RAPT can be divided into two parts according to their functions and implementation. The first part is the input system which reads RAPT programs and transforms the information in the programs into RAPT internal data structures. The second part is the geometrical reasoning system which, making use of data structures created by the input system, reasons about the positions of objects and the actions of the robot. The output of normal RAPT is a series of positions of objects (including the robot manipulator) in each situation. These positions can then be translated into a suitable form, usually a lower level robot command language, e.g. VAL commands and data, which can then be executed by the robot.

A number of RAPT programs have been tested successfully (e.g. [KEM81]). The RAPT work has shown that the use of spatial relationships is a powerful tool for describing assembly tasks in an object level robot language. It provides a natural way for the user to specify the desired goal states in robot assembly tasks.

3.2. The RAPT Input Language

The RAPT input language is the interface between the programmer and the RAPT geometric reasoning system. By the use of the input language,

the user can define three-dimensional objects, desired states, and the actions between every two adjacent states of the assembly task. The desired states are defined in terms of the spatial relationships holding between objects rather than in terms of the absolute positions of objects. This makes programming in RAPT potentially much easier than in other lower level robot languages, since the tedious computational work will be done by the computer rather than by the user himself. The information given by the user about the assembly task is transformed into internal data structures and then sent to the geometric reasoning system.

3.2.1. RAPT Models

RAPT provides a rudimentary three dimensional modelling system. Objects to be handled by the manipulator, and the objects in the world are modelled in this system in terms of their surface features. The modelling system is capable of modelling complex objects with planar, cylindrical and/or spherical surfaces. Each object in the RAPT environment is represented by those of its surface features which are to be used in the associated RAPT program. The RAPT model is an incomplete one. This means that only the features used by the associated program need to be described in the model. The RAPT models are simply collections of related surface features of objects and therefore the RAPT system has no knowledge about the space occupancy of the object modelled.

3.2.1.1. Body Definition

In RAPT, objects, tools and robot gripper parts are referred to as bodies. Each body has its position and local coordinate system. The positions of bodies within the RAPT system represent the positions of the corresponding bodies in the actual work station and the local coordinate system provides a reference frame for the positions of features. Bodies are declared by body definition statements, possibly with some feature descriptions of the bodies.

There is a special body in the RAPT environment called the "world". Its coordinate system has fixed relations with that of the actual robot working station and its position is fixed. The "world" is created automatically at the beginning of each RAPT program, and is used to provide a frame of reference in which all actions take place.

A body description has the form

```
BODY/<body-name>;
     <body-definition-statements>;
TERBOD;
```

where <body-name> is the logical name of the body being defined, and <body-definition-statements> can be none, one or more RAPT modelling primitive defining statements.

3.2.1.2. Geometric Primitives

There are two kinds of primitive in the RAPT modelling system: geometric primitives and body features. Both of them are associated with the body within whose definition they are declared. If a primitive is defined outside any body declaration package then it belongs to the world. The main difference between these two kinds of primitive is that the geometric primitives are only used in the construction of models whereas feature primitives are used in goal state specifications.

There are three sorts of geometric primitive. They are points, lines and circles. Each geometric primitive is defined by a statement specifying its name, type, position, and, where appropriate, dimensions.

A RAPT point is purely a geometric point defined by its X-, Y- and Z-coordinates in a three dimensional space. The syntax of a point definition statement is

<point-name>=POINT/<point specification>;

where <point specification> specifies the coordinates of a point. For example, the user can say:

p1 = POINT/20, 30, 40;

This specifies a point named p1 with its X-, Y- and Z-coordinates being 20, 30 and 40 respectively. There are four forms of specifying the coordinates of a point. Detailed discussion about these and forms of defining all other primitives can be seen in [AMB82].

A RAPT line is a geometric line segment determined by its start point and end point. It therefore has a length and direction. The syntax of a line definition statement is

<line-name>=LINE/<line specification>;

where <line specification> specifies the position, direction and length of a line. There are seven forms of line specification. For example, a statement:

line_1 = LINE/p1, p2;

defines a line "line_1" which passes through points p1 and p2 which have already been defined.

A RAPT circle is defined by its centre and radius. It is limited, (version 1.1a) in the current version of RAPT, to one which lies on a plane which is parallel to the X-Y plane of the coordinate system of the associated body. The syntax of a circle definition statement is

<circle-name>=CIRCLE/<circle specification>;

where <circle specification> specifies the centre and radius of a circle. There are two forms of circle specification. For example,

p1 = POINT/10, 20, 30;

c1 = CIRCLE/CENTER, p1, RADIUS, 10;

defines a circle c1 which has radius of ten units at point p1.

3.2.1.3. Feature Primitives

There are six types of RAPT features. They are faces, holes, shafts, edges, spherical faces and vertices. Each feature is defined by a statement specifying its name, type, position, and, where appropriate, dimensions. Some types of feature have a direction associated with them. This direction is important in defining relations between features. The positions of the features indicate their locations in the local coordinate system of the body to which they belong. The general form of the feature definition statement is as follows:

<feature-name> = <feature-type>/<feature definition>;

A RAPT face is defined by a point on the face and a direction vector of the normal to the face. The direction of the normal is defined in such a way that if the face is imagined to represent a surface of a solid body, then the direction of the normal points outwards from the solid body. A face definition does not include information about its extent and therefore a RAPT face is an infinite plane feature. The syntax of a face definition statement is

<face-name>=FACE/<face specification>;

where <face specification> specifies the position of a point on a face and the direction of its normal. There are nine forms of face specification. For example,

```
f1 = FACE/p1, p2, p3, xlarge;
```

defines a face f1 which the points p1, p2 and p3 lie on and whose normal points in the direction with a positive X-component.

A RAPT shaft is a convex cylindrical surface represented by its axis and radius. The direction of the axis points towards the end of the shaft. The syntax of a shaft definition statement is

```
<shaft-name>=SHAFT/<shaft specification>;
```

where <shaft specification> specifies the axis and radius of a shaft. There are two forms of shaft specification. For example,

```
l1 = LINE/p1, p2;
bolt = SHAFT/AXIS, l1, RADIUS, 5, ylarge;
```

defines a bolt to be a shaft with radius 5. The axis of the shaft is along the line l1 and points to the direction with a positive Y-component.

A RAPT hole is very similar to a shaft. The only difference is while a shaft is a convex cylindrical surface, a hole is a concave one. A hole is represented by its axis and radius, just like a shaft. The direction of its axis points out from the hole. The syntax of a hole definition statement is

```
<hole-name>=HOLE/<hole specification>;
```

where <hole specification> is the same as a shaft specification. There are also two forms of hole specification. For example,

```
c1 = CIRCLE/CENTER, p1, RADIUS, 15;
h1 = HOLE/c1, zlarge;
```

defines a hole h1 with radius 15 at a point p1. The X-axis of the hole points upwards.

A RAPT edge is considered as a special case of a shaft of zero radius. Therefore, it could be defined by the use of a shaft definition statement. For convenience, however, it has its own distinct definition statements. The syntax of an edge definition statement is

```
<edge-name>=EDGE/<edge specification>;
```

where <edge specification> specifies the axis and direction of an edge. There are two forms of edge specification. For example,

```
l1 = LINE/p1, p2;
e1 = EDGE/l1, xsmall;
```

defines an edge which matches the line l1. The X-axis of the edge points to the direction with a negative X-component.

Although shafts, holes and edges are all dealt with by the RAPT reasoning system as infinite in extent, they may in fact represent finite features, and the user may specify their lengths in some forms of feature definition statements. Although this dimensional information is

ignored by the RAPT reasoning system, it can be used for other purposes, e.g. in a sensory system.

A RAPT spherical face is a convex spherical surface like a ball. It is a finite feature defined by a centre and a radius. There is no associated direction. The syntax of a spherical face definition statement is

<sphface-name>=SPHFACE/<centre>, <radius>;

where <sphface-name> is the name of a spherical face, and <centre> is the position of the centre point of the spherical face, and <radius> is a number indicating the radius of the spherical face. This is the only form of spherical face specification.

```
p1 = POINT/0, 5, 10;
ball = SPHFACE/p1, 5;
```

defines a spherical face named "ball" which centers at the point p1 with radius 5.

A RAPT vertex is a special case of a spherical face of zero radius. It approximates the corner of an object. A vertex is defined by specifying the point at which it lies. The syntax of a vertex definition statement is

<vertex-name>=VERTEX/<point>;

This is the only form of vertex specification. The following statements

```
p2 = POINT/2, 3, 4;
v1 = VERTEX/p2;
```

define a vertex at the point (2, 3, 4).

The following example shows a RAPT model of the object in Fig. 3.1. The object is named as body1 here. Notice that not all the features of the actual object are modelled.

```
body/body1;
     p1=point/0,0,0;
     p2=point/20,0,0;
     p3=point/20,-20,0;
     p4=point/20,-20,30;
     p5=point/20,0,30;
     p6=point/0,0,30;
     p7=point/0,-20,0;
     p8=point/10,-10,0;
     p9=point/10,-10,-20;
        f1=face/p1,p2,p5,ylarge;
        f2=face/p1,p2,p3,zsmall;
        f3=face/p1,p6,p7,xsmall;
        f4=face/p4,p5,p6,zlarge;
        f5=face/p3,p2,p4,xlarge;
     l1 =line/p2,p3;
     l2 =line/p1,p6;
     l3 =line/p8,p9;
        e1=edge/l1,ysmall;
        e2=edge/l2,zlarge;
     s1=shaft/axis,l3,radius,5,zsmall;
terbod;
```

3.2.2. Relations

In an assembly task, bodies are moved from place to place to bring about desired goal states. Assembly operations can also be thought of in this way, although there is the question of forces. In RAPT, each

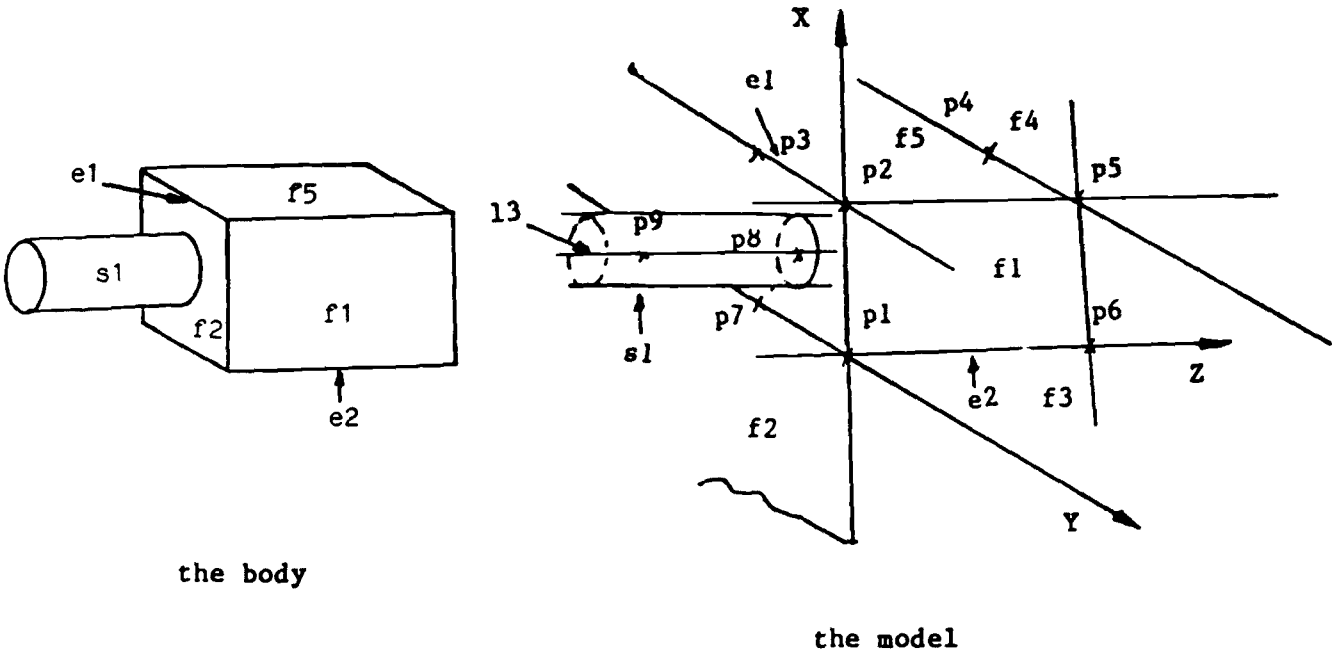


Fig. 3.1 An example of a RAPT body model

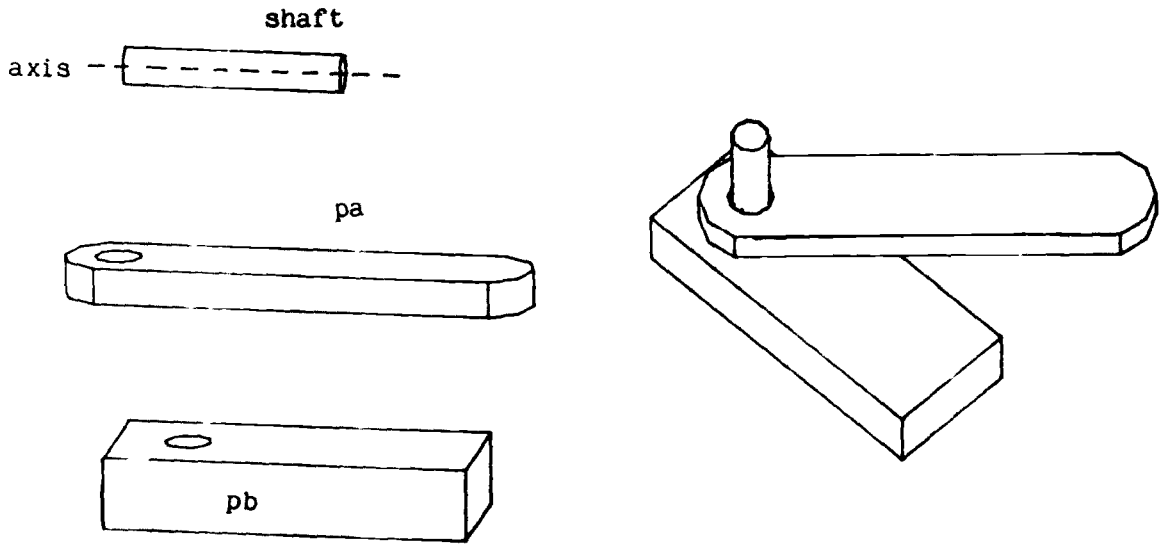


Fig. 3.2 An example of a subassembly

state of the whole environment between any two actions is referred to as a situation. A situation can be described explicitly in terms of spatial relationships which will hold between features of bodies in that situation, or implicitly in terms of the action which occurred between the situation and the previous one, or by a mixture of both explicit and implicit information. The state of an individual body in a situation is referred to as a body instance. Therefore, if there are M bodies and N situations then there are MxN body instances. It is easy to see from these definitions that spatial relationships hold between features of body instances rather than of bodies. If there is no possibility of confusion then the term "body" may sometimes be used instead of "body instance" in the following discussion.

There are six types of spatial relationships available in the RAPT input system to describe situations. They are called AGAINST, COPLANAR, PARAX, PARALLEL, FITS and ALIGNED. The syntax of a relationship statement is

<relation-type>/<feature-1>, <feature-2>;

where <feature-1> and <feature-2> specify the two features between which the relationship holds in the current situation. There are two forms which can be used to refer to a feature: either

<feature-name> of <body-name>

or, when there is no ambiguity, the feature name can be used. The following is an example of a relation specification:

AGAINST/bottom of block, work_top;

The AGAINST relationship can hold between two faces, a face and a shaft or an edge, a face and a spherical feature (a spherical face or a vertex), or two spherical features. The general meaning of this relationship is that the surfaces of the two features share a common tangent plane. A face is against another face when they lie in the same plane with their normals in opposition. A face is against a convex cylindrical feature (a shaft or an edge) when the X-axis of the cylindrical feature lies in the plane parallel to the face, and removed away from the face by a distance equal to the radius of the cylindrical feature. A spherical feature (a spherical face or a vertex) is against a face when the centre of the spherical feature lies in a plane parallel to the face, and removed away from the face by a distance equal to the radius of the spherical feature. Two spherical features are against each other when the distance between their centres is equal to the sum of their radius. COPLANAR holds only between two faces. It differs from the AGAINST relationship between two faces only in that the face normals point in the same direction.

FITS holds between two cylindrical features (shaft, holes or edges). This relationship means that the axes are collinear but point in opposite directions. ALIGNED is the same as FITS except that the axes point in the same direction.

PARAX can hold between two faces or two cylindrical features. Its general meaning is that the X-axes of the features must be parallel but point in opposite directions. This relationship does not require the two features between which the relationship holds to share a common

tangent plane, therefore it is the general case of AGAINST with a specified distance between the faces or cylindrical features. PARALLEL is the same as PARAX except that the X-axes of the features point in the same direction.

Table 3.1 lists all the RAPT relationships with valid types of the associated features.

relations	features		relations	features	
AGAINST	face	face	FITS & ALIGNED	shaft	hole
	face	shaft		shaft	edge
	face	edge		shaft	shaft
	face	vertex		hole	edge
	face	sphface		hole	hole
	vertex	vertex		edge	edge
	sphface	sphface			
	sphface	vertex		shaft	hole
			PARAX & PARALLEL	shaft	edge
COPLANAR	face	face		shaft	shaft
				hole	edge
				hole	hole
				edge	edge
				face	face

Table 3.1. Spatial Relationships in RAPT

3.2.3. Actions

In RAPT, actions transform one situation into another. There is always a single action between each pair of contiguous situations. The action statements serve to identify which bodies may move between one situation and the next, and can also be used to describe the change in the position of bodies. Actions are described in terms of movements of particular bodies --- this can be relative to features of other bodies, and can also be restricted to pure translational movements or pure

rotational movements. It should be noticed that a movement of one body can imply the movement of other bodies that are in some way connected to it (see TIES and SUBASSEMBLIES). In RAPT, between one situation and the next, all bodies are assumed to move unless it can be shown that they have not (see Section 3.2.6). Similarly, it is assumed that spatial relationships holding in one situation do not necessarily hold in any subsequent ones. There are two action types allowed in RAPT: MOVE and TURN, corresponding to translational and rotational actions respectively. According to the detailed specification of the action, the statements can be classified into three categories: explicit action statements, implicit action statements and general move statements. The explicit action statement serves to specify completely the absolute motion of the body being moved. It takes the form:

<action type>/<body>,<relation>,<feature>,<amount>;

For example, a statement:

MOVE/ block, PERPTO, bottom of block, 20;

asks the robot to move the body "block" 20 units along the direction which is perpendicular to the bottom of the block.

If <action type> is MOVE then <relation> can be either PERPTO (for perpendicular to) or PARLEL (for parallel). If <action type> is TURN then <relation> must be ABOUT. <amount> is a real number. The unit of the movement depends upon the type of the action. Rotational actions are measured in degrees while the unit of translational actions is determined by a post-processor. This statement specifies completely the

motion of the body relative to its original position.

There is an exception in this kind of statement. When the action type is MOVE and the action is said to be parallel to a face feature, it is not a explicit one. Although the amount is specified, the direction of the action cannot be determined. In order to determine the destination of the action, relations between the moving body and others must be specified.

While the explicit statement completely specifies the displacement of a body, two variants of action statement are allowed in RAPT which specify the action incompletely. The first of these is the implicit action statement, which gives incomplete information about the movement relative to other bodies. It has the following syntax:

<action type>/<body>,<relation>,<feature>;

This version specifies only the type of the action and direction of the movement or the rotation axis. The amount of action in this case is determined by position constraints obtained from the spatial relationships that must hold between this body and others in both the starting and the destination position.

The other variant is the general move statement, which only identifies which bodies are moved. It has the following syntax:

MOVE/<body>;

The general move does not make any connection between the positions of

the body being moved before and after the action. Position constraints on the start point and destination of the body determine both the direction and amount of motion.

In RAPT, only some special bodies specified as "agents" can be the source of an action. This means that a body to be moved must be either an agent itself or connected in some way, such as TIE or SUBASSEMBLY (both will be discussed later), to an agent. An agent is defined by the statement

AGENT/<body>;

3.2.4. Ties

It often happens that bodies become fixed together so that they cannot move relative to each other during some period in an assembly process. For example, when two components are bolted together, they cannot move relative to each other until they are unbolted. RAPT provides a TIED statement to describe this kind of phenomenon.

In RAPT when two bodies are tied together this means that they maintain the same relative position before and after any actions. Therefore, any descriptions of the motion of one body must apply to the motions of any other bodies tied to it. TIEs are made and revoked by the statements:

TIED/ <body 1>, <body 2>;

and

UNTIED/ <body 1>, <body 2>;

Once two bodies are tied in a situation description, they are assumed to remain tied for all subsequent actions until the tie is broken by saying explicitly that they are UNTIED. Both TIED and UNTIED statements have effect in the action which follows the situation in which they have been declared. For example,

TIED/block, world;

ties the block with the world so that any attempt to move the block in subsequent situations before the revocation of the tie is an error and will be reported to the user.

3.2.5. Subassemblies

A subassembly is a set of bodies between whose features certain specified relationships hold for the duration of the existence of the subassembly. Subassemblies differ from ties in that there may be more than two bodies within a subassembly and the components of a subassembly can move with respect to each other during the existence of the subassembly, provided that the relations remain valid. For example, a gripper is usually defined as a subassembly whose two palms can move relative to each other in a direction perpendicular to the palm faces.

A subassembly is declared by the statements

<subassembly-name>=SUBASS/<duration>;

```
    <relationship definitions>;
TERSUB;
```

where <duration> indicates whether the subassembly is a permanent one or a temporary one. A permanent subassembly exists throughout every situation while a temporary one is brought into effect by ISSUB and revoked by NOTSUB statements. Their syntax is

```
ISSUB/<subass-name>;

NOSUB/<subass-name>;
```

The temporary subassembly is usually used to describe some intermediate combination of workpieces which is partly assembled. For example,

```
link = SUBASS/temp;
     FITS/axis of shaft, hole of pa;
     FITS/axis of shaft, hole of pb;
     AGAINST/f1 of pa, f2 of pb;
TERSUB;
```

defines a temporary subassembly which is shown in Fig. 3.2. The permanent subassembly, on the other hand, is usually used to describe instruments or tools such as the manipulator.

3.2.6. Unmoved Bodies

In RAPT, only one body can be moved directly by an action statement between one situation and the next. Other bodies can be moved indirectly by connections with the moved body via a tie or a subassembly. The input system checks between every two contiguous situations to see which bodies are moved by these connections. If a body has no way to be affected by a moved body then it is considered to be unmoved, and its name is recorded in a table, called the unmoved body table, which is indexed by situations.

3.3. The RAPT Inference System

In order to decide how to move the robot in order to achieve the assembly task, the relational specification given by the user must be transformed into positional information. This is performed by the RAPT reasoning system. There are two versions of the implementation of the reasoning system. The first one is referred to as an equation solving system which is described in full in [POP80]. In this system the relations between features of body instances are expressed by rewriting the position of one body as an algebraic expression involving the position of the other and so producing a set of simultaneous equations which can be solved. The second reasoning system is referred to as a cycle finding system and this is discussed in detail in [POP79, POP81, AMB83]. This works by applying a set of reasoning rules to the original relations given by the user, replacing them by new ones which are fewer and more restricted. This rewriting is repeated until no further progress can be made. The first version is more powerful than the second one in

solving some difficult relation pairs while the second one works about 100 times faster than the first. The current RAPT system employs the second version of the implementation, i.e. the cycle finder. There is a plan to combine these two methods in the future in such a way that firstly the cycle finder will be used to solve easy relationship cycles and then the equation solver will be applied to tackle the few remaining hard relationship cycles.

3.3.1. Representations of Positions

In RAPT, a position is a product of a translation and a rotation. It is represented by a homogeneous coordinate transformation [AMB75b]. In the homogeneous coordinate representation [FOR69], a point in N-dimensional space is represented by an (N+1) dimensional vector. An N-dimensional space transformation is represented by an (N+1) x (N+1) matrix. For a three dimensional space, it is a 4x4 matrix of the form:

$$\begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}$$

Since it is easy to use the homogeneous coordinate form to describe various kinds of transformation such as rotation, translation, scaling etc., it is widely used in the fields of robot control [PAU81], image analysis [DUD73] and computer graphics [FOR69, FOL82]. When the homogeneous coordinate representation is used as a transformation matrix in a post multiplication system, the upper-left part represents scaling, shear and rotation; a14, a24 and a34 specify the vanishing point in a

perspective projection; a41, a42 and a43 represent translational shift in the direction of X-, Y- and Z- axes of its reference frame respectively; a44 represents a general scaling. In RAPT, this matrix is used to represent the position of a body or a feature, and therefore scaling, shear and perspective projection have no significance. Thus the upper-left part of the matrix represents rotation only; a41, a42 and a43 represent translational shift; a14, a24 and a34 are all zero, and a44 is always equal to 1.

Both bodies and features have their own local coordinate systems attached to them. The position of a body instance represents the position and orientation of the local coordinate system with respect to that of the world. In practice, the X-Y plane of the world coordinate system is usually coplanar with the top of the work table. Its Z-axis points vertically upwards. Similarly the position of a feature represents the position and orientation of the local coordinate system of the feature in the coordinate system of the body to which the feature belongs. For example, in Fig. 3.3, f1 is the position of the cylindrical feature F1 of the body b1. Position transformation is defined by the post multiplication. Therefore the position of a feature with respect to the world frame can be expressed by a transformation

f ** p

where ** denotes matrix multiplication, f is a feature position and p is the position of the body to which the feature belongs.

For a face, the X-axis is along the direction of the normal of the face and the origin lies in the face. For a cylindrical feature, the

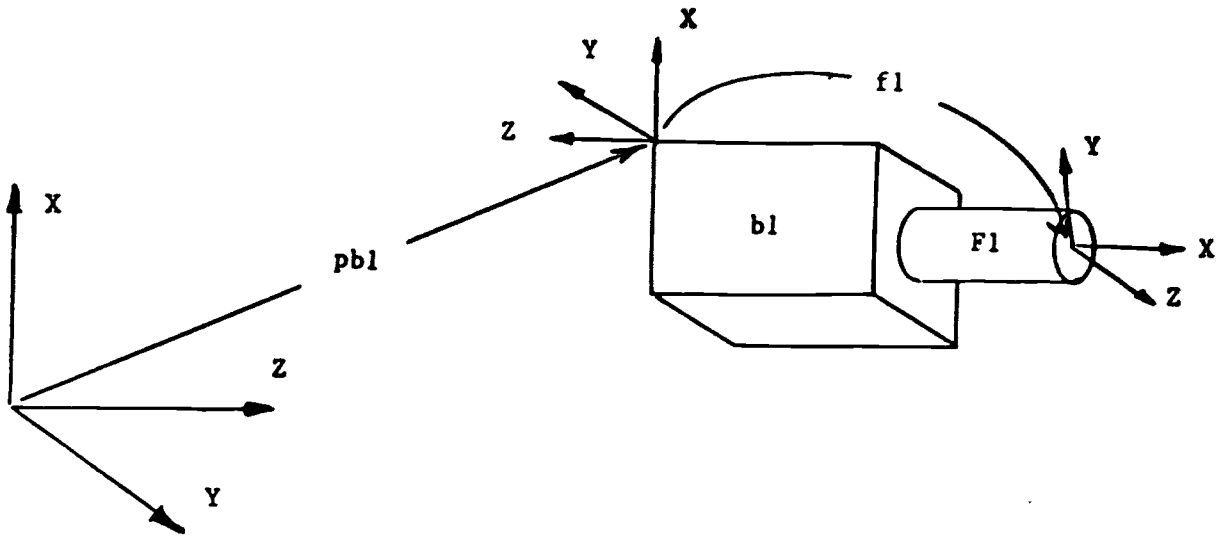


Fig. 3.3 Positions of the body and the feature

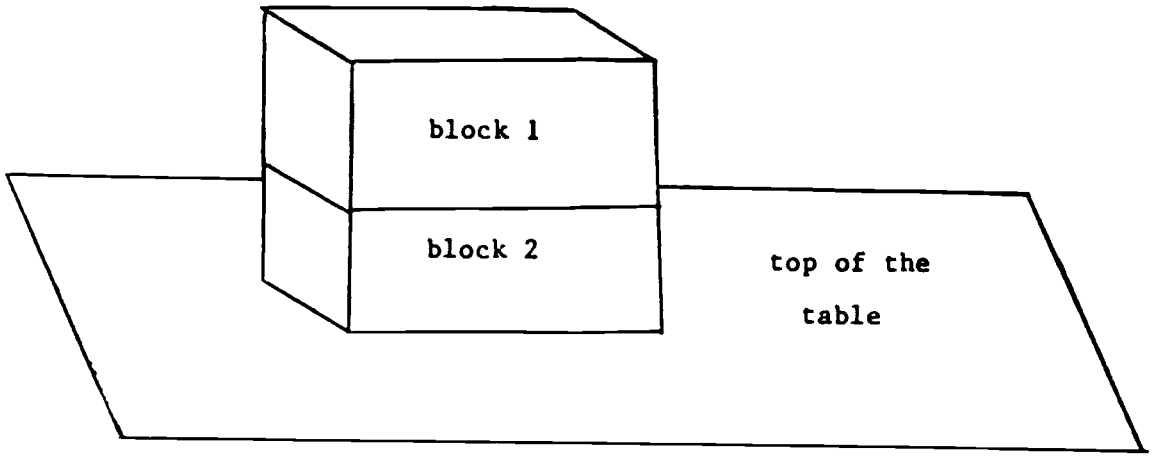


Fig. 3.4 A situation in which creation rules can be applied

X-axis is along the direction of the axis of the feature and the origin lies somewhere on that feature axis. For a spherical feature, the origin of its coordinate system is coincident with the centre of that feature while the direction of the X-axis is assigned arbitrarily by the RAPT system. The direction of the X-axis and the position of the origin of a feature coordinate system are also usually determined systematically by the RAPT reasoning system during the inference processes. In contrast to the X-axis, the direction of the Y- and Z-axes are usually chosen arbitrarily by both the input and reasoning system provided they constitute a Cartesian coordinate system.

3.3.2. The Equation Solving System

This is the first version of the RAPT inference system [POP80]. It translates a collection of information about bodies, situations and actions into a "tree of knowledge", and then controls the production and solving of algebraic equations.

In this system, the relations are represented by equations. Each equation has the form

$$f2 \;**\; p2 = rel(v) \;**\; f1 \;**\; p1 \qquad (3.1)$$

where f1 and f2 are the positions of two features between which the relation holds, p1 and p2 are positions of the bodies to which the features belong respectively, v is a vector of n variables unique to a given relation, and "rel" is an algebraic function depending upon the type of the actual relation. For example, if the two features are faces

and they are against each other, then the equation is

$$f2 ** p2 = M ** twix(\theta) ** trans(0,y,z) ** f1 ** p1 \qquad (3.2)$$

where M is a matrix representing a rotation about the Y-axis which brings the positive X-axis to the negative X-axis, twix(θ) is a rotation transformation about X-axis by θ, and trans(a,b,c) is a translation matrix with the measure of a, b and c along X-, Y- and Z-axes respectively. By the use of this equation, the position p2 can be expressed in terms of f1, f2 and p1 with the vector of three variables

$$p2 = f2^{-1} ** M ** twix(\theta) ** trans(0,y,z) ** f1 ** p1 \qquad (3.3)$$

Similarly, action statements (excluding general move) also specify relationships between positions of body instances and therefore can also be expressed by algebraic equations with the form

$$f ** p2 = rel(v) ** f ** p1 \qquad (3.4)$$

where f is the position of the feature which the action is relative to, p1 is the position of the body before the action, and p2 the position of the body after the action.

The fundamental work of the equation solving system is to form pairs of non-linear simultaneous equations connecting positions of two body instances. The general form of a pair of equations is

$$p2 = expression1(v1) ** p1 \qquad (3.5)$$

$$p2 = expression2(v2) \;**\; p1 \qquad (3.6)$$

where each expression is the combination of one or more relation functions and feature transformations, such as that in (3.3). The equation solver combines this pair of equations and attempt to solve them so that a new equation

$$p2 = expression3(v3) \;**\; p1 \qquad (3.7)$$

is formed, where the number of variables contained in v3 is less than or equal to the minimum of that of v1 and v2. Usually the new equation created by the equation solver is a more restricted one and contains fewer variables than both of the two original ones, and can be used by the equation solver in the further inference in the same way as the original ones. If the complexity of the equations is not beyond the capability of the equation solver, then this inference process will continue until all equations have been simplified as much as possible. If any variables are left then this shows that the information given by the situation and action descriptions is not enough to determine completely the relative position of all bodies with respect to the world.

3.3.3. The Cycle Finding System

This is the reasoning system used in the current RAPT, and also the one used in the work described in this thesis. The cycle finding system performs the same function as the equation solving system but in a different way. Its main advantage over the equation solver is its speed and compactness. The method used in this system comes from the

realization that in many cases there are commonly occurring standard solution patterns to equation pairs. For example, if two face pairs of two bodies are against each other, the resulting constraints usually indicate that the relative position of one body with respect to the other has only one translational degree of freedom. The cycle finding system therefore looks at the relational network representing the current state to see if it can find such standard patterns, and then replaces them with standard solutions.

3.3.3.1. Relationships in the Cycle Finder

In the cycle finding system, each relation is represented by a 4-tuple

(r, F1, F2, s)

Here r is the type of the relation, F1 and F2 are the body features related by the relation, and s is an integer indicating the situation in which the relationship must hold. The relations in the cycle finder can also be considered to have the same form as (3.1):

$$f2 ** p2 = rel(v) ** f1 ** p1 \qquad (3.8)$$

except that here the algebraic function "rel" and the variable vector v are implied by the type of the relation, and p1, p2, f1 and f2 are implied by the features F1 and F2 and the situation s.

There are ten types of relationships used in the cycle finder.

Because of the way in which the cycle finder recognizes and treats standard pairs of relationships, the relationships used in this system are more finely categorised than those used in the input language. Thus the AGAINST relationship of the input system is divided into four subcategories depending upon the type of features involved. Also some new types are introduced: for example, the LIN relationship has only one degree of freedom, a translational one. This relationship is deduced by the system when two pairs of faces of two bodies are AGAINST each other. The cycle finder relationships are named as AGPP, AGPC, AGPS, AGSS, FITS, PARAX, LIN, LINLIN, ROT and FIX. The AGAINST relationship of the input system is replaced by four new relationships distinguished by the types of the related features. This is because the mathematics of AGAINST relationship differ fundamentally for different cases. Thus AGPP holds between two faces, AGPS between a face and a spherical feature, AGPC between a face and a convex cylindrical feature (a shaft or an edge), AGSS between two spherical features. COPLANAR is transformed into an AGPP by changing the direction of one of the related faces. ALIGNED and PARALLEL are treated by the cycle finder as FITS and PARAX respectively after their directions have been adjusted. LIN holds between two cylindrical features and indicates that the X-axes of the two features are collinear and their Y-axes are parallel. This relationship can either be created by a MOVE statement or be deduced as an intermediate result by the cycle finder during reasoning. LINLIN holds between two faces. It is similar to an AGPP except there is no relative rotation allowed. This relationship can be deduced by the cycle finder as an intermediate result. It can also be generated by the MOVE statement which specifies a movement parallel to a face feature. ROT holds between two cylindrical features indicating that the X-axes of the features are collinear and their origins are coincident. This

relationship can be either created by a TURN statement or deduced by the cycle finder during reasoning as an intermediate result. FIX holds between any kind of feature and indicates that their local coordinate systems coincide. This relationship is an important one since it represents a completely determined position of one body instance relative to another

$$f2 \;**\; p2 = f1 \;**\; p1 \qquad (3.9)$$

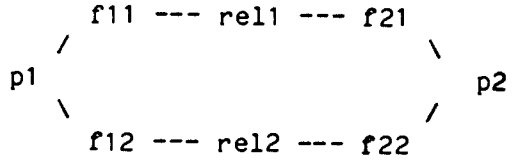
It is the most restricted relationship. The purpose of the reasoning of the cycle finder is to attempt to establish FIX relations between body instances in place of the relations given by the user. If a body instance is "fixed" to the world directly or indirectly (i.e. "fixed" to another body instance which has been "fixed" to the world) then the position of the body instance in the world can be completely determined.

3.3.3.2. The Relational Network and Cycles

In the cycle finder a relational network is built up from the input data, in which body instances are nodes and relations constitute undirected arcs. Cycles occurring in the network are used in the reasoning system. The size of the cycle is the number of nodes occurring in it, and one of size n is referred to as an n-cycle. The fundamental step of the cycle finder during reasoning is to find standard relation cycles and then apply some standard solutions in the form of reasoning rules to them according to their sizes and natures.

3.3.3.3. Reasoning Rules

Two sorts of reasoning rules are used in the cycle finder. The first sort is referred to as the combination rule. This set of rules is applied to 2-cycles and is listed in Table 1 in Appendix I. A 2-cycle is a relation pair holding between two body instances like



(3.10)

Rules in Table 1 are indexed under types of pairs of relations. If there is a suitable entry in the combination rule table for the two relations in a 2-cycle then a new relation is produced to replace the two original ones. For example, there is an entry in Table 1 for the pair AGPP FITS and this rule will be applied whenever a 2-cycle is found in which rel1 is AGPP and rel2 is FITS. The rule defines how the two spatial relationships constrain each other, and how they can be replaced by a single relationship. This single relationship is usually more constrained than either of the originals. For example, two AGPPs together can usually be replaced by a single LIN relationship. (If the plane faces involved are parallel then they will be replaced by a single AGPP). The introduction of the LIN relationship means that two new (virtual) features have to be invented by the system for this relationship to hold between. The rules in Table 1 include details of how to construct these new features. The result of applying the combination rule can be expressed in a general form as:

p1 -- f13 --- rel3 --- f23 -- p2 (3.11)

The process of applying combination rules will continue until either there are no 2-cycles existing in the relation network or there are no suitable entries for those that do exist.

When a relationship pair is to be combined, the cycle finder will examine relationships between the body features involved. The purpose of this examination is twofold. Firstly, it will decide whether the two relationships are compatible or not, i.e. whether the specified relationships really hold simultaneously for the given positions of the features. Sometimes, the positions of the features given by the user are not very accurate. The system itself introduces numerical errors. The cycle finder will check whether this inaccuracy is within a range which is pre-specified by the user. Secondly, the examination will decide whether any special conditions hold between body features so that the cycle finder can select the appropriate reasoning rules. For example, when AGPP is combined with FITS, the general result is a FIX relationship. However, if the two X-axes of the features of one body are parallel then the newly generated relationship is a ROT relationship. If the X-axes are perpendicular to each other then the result is LIN.

The second sort of reasoning rules are the creation rules. This set of rules is used for applying inference around the relation chains in the cycles whose sizes are larger than two. They are listed in Table 2 in Appendix I. The rules in Table 2 are also indexed under types of pairs of relations. The relation chain has the form

p1 -- f11 --- rel1 --- f21 -- p2 -- f22 --- rel2 --- f31 -- p3 (3.12)

If there is a suitable entry in the creation rule table for the two

relations in a relation chain, then the corresponding creation rule is applied to create a new relation between p1 and p3. This new relation holds between two newly created features.

p1 -- f12 --- rel --- f32 -- p3 (3.13)

For example, in Fig. 3.4, block 1 is on the top of block 2 while block 2 is on the top of the table. If the top and the bottom of block 2 are parallel to each other then according to the creation rules a new virtual face feature is generated which belongs to block 1 and is against the top of the table. Thus a new relationship AGPP is created between block 1 and the table. Creation rules are applied successively to a large cycle until either the size of the newly created cycle becomes two or there are no suitable entries for the existing relation chain in the creation rule table. Once a 2-cycle has been produced the combination rules can be applied.

During reasoning the cycle finder applies these two sorts of rules alternately until there are no entries in any reasoning rule tables for the existing relations in the relation network. Each time after the application of the combination rules a special process called "merge" takes place. In this process, each relation is checked to see whether it is of FIX type or not. If it is so then the position of one related body instance, say p2, is completely determined with respect to the other, say p1. Thus the position p2 is removed from the set of unsolved positions, and any relations involving p2 are transformed into ones involving p1. If the position p1 is solved, it can be transformed to produce the position p2.

The size of the relational network decreases during the reasoning process. Combination rules try to replace a pair of relationships by a single relationship. Thus they decrease the size of the network by reducing arcs in it. Creation rules generate new arcs in the network, but only the relationships which can be deleted later by combination rules are kept in the network. Thus creation rules do not increase the size of the network. The merging process attempts to merge one body instance into another one. It absorbs nodes in the relational network. The final result of the network after the reasoning by the cycle finder may contain only one node which is the world if every distinct situation is fully specified by the user and solvable to the cycle finder. Otherwise, it is still a network to which the cycle finder cannot do anything further, i.e. some body instances have unresolved degrees of freedom.

The cycle finding system has its limitations. It is less powerful than the equation solving system. The cycle finder needs the newly generated relationship to be of one of the ten relation types. Otherwise, reasoning rules cannot be applied to the corresponding relationship pair, and a "-" is marked in Table 1 to represent this case. For example, the general result of combining a pair of two AGPC relationships is not one of the ten types which are allowed in the cycle finder. Thus there is no suitable entry in Table 1 for this pair. Sometimes, the results which are produced by the cycle finder contain two solutions. This kind of result has a mark (2) after the resulting relation type in Table 1. This happens usually when cylindrical or spherical features are involved in the relationships. For example, when ROT is combined with AGPS, the cycle finder produces two possible FIX relationships as shown in Fig. 3.5. The cycle finder cannot decide which solution is appropriate simply by the reasoning rules. Thus this makes difficulties

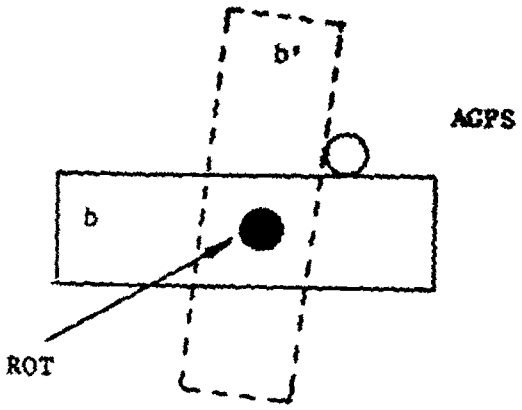


Fig. 3.5. An example of two-solution FIX

Here b and b' are the same body in two different positions which both satisfy the ROT+AGPS relations

for subsequent reasoning.

Chapter 4. Computer Vision in Robotics

Computer vision is a research field in which efforts are made to enable the computational system to "see" a scene. The input is usually received by visual sensors such as TV cameras in terms of large arrays of digitized brightness information. The arrays of brightness values represent projections of a three-dimensional scene. Several input arrays may provide information in several spectral bands (colour) or from multiple viewpoints (stereo or time sequence). The desired output of a computer vision system varies: for instance, it may be a symbolic description of the scene, or the specification of information relevant to special goals of the system. The exact nature of the output depends upon the task of the vision system. It could involve descriptions of objects and their inter-relationships. It may also include such information as the three-dimensional structure of surfaces, their physical characteristics and so on.

Computer vision is a research field, which is considered to be the enterprise of automating and integrating a wide range of processes and representations used for vision perception [BAL82]. It includes as parts many techniques that are useful by themselves, such as image processing (which transforms, encodes and transmits images), statistical pattern recognition (which applies statistical decision theory to general patterns), and geometrical modelling (which represents the geometrical aspect of objects in the scene). Research in cognitive psychology and computer science is, of course, relevant to computer vision. The principal purpose of the research in computer vision is to develop an artificial vision system capable of high performance in a broad range of visual domains.

In this chapter, the state of general purpose computer vision is surveyed briefly, but more attention is paid to the computer vision in robotics. In robotics, as in some other special areas, some special conditions and constraints may be applied to the environment in which the vision system works, and the requirement of vision performance may be limited to some extent so that the vision task can be simplified. Some of the descriptions given below are substantially based on the abstracts and text of the original authors' descriptions of their research.

4.1. A Brief Survey of Computer Vision

Computer vision systems are used in a number of domains [HAN78] such as aerial image analysis [BRO81], astronomy, medical research, chemistry and robotics [PER78, AGI80] in modes that are heavily dependent upon domain-specific constraints and techniques. For example, the current practical industrial vision systems usually require high contrast to obtain binary images and use overhead cameras to minimize variations in object appearance. Although these are special purpose systems, they do provide practical tools for research work in certain areas. However, it has proved exceedingly difficult to construct effective general purpose computer vision systems which are capable of dealing with less predictable and less structured scenes [HAN78, BAR81]. The research into special purpose systems, together with the research in areas of cognitive psychology, image formation, knowledge representation, etc. provides new insights into the computational nature of vision that could lead to effective general purpose vision systems.

4.1.1. Modelling in Computer Vision

A major computational principle of vision is that competence depends upon the models available. When a human is looking at the environment around him, he has prior knowledge about the environment, and this knowledge guides the interpretation of what he has seen. Similarly, a computer vision system also requires knowledge about the perceived objects, not only the general knowledge about the properties of objects in the world like surface smoothness and reflectivity characteristics, but also specific knowledge about the structure of objects, relationships among primitives of objects and so on, if the vision system is expected to understand the scene well. This is because an image encodes much information about the scene, and the information is compounded in the single brightness value at each point. Moreover, information about the three-dimensional structure of the scene is lost in the projection onto the two-dimensional image. On the other hand, the meaning conveyed by an image usually is not derivable solely from the sensory data being processed. It is dependent upon the goals of the vision system and prior knowledge of various aspects of natural world and image domain. In order to decode brightness and recover a scene description, prior knowledge embodied in models of the scene domain, the illumination, and the imaging process must be provided and exploited.

Scene models describe the three-dimension world in terms of surfaces and objects. Surfaces can be described in terms of continuity, smoothness, reflectivity, etc.. Objects can be described in terms of surfaces, boundaries, and other primitives. Objects may be specific or

generic, and their interrelationships are governed by some physical constraints.

Illumination models describe the primary light sources, their positions, spatial extents, intensities, colours, and so on. A complete (or nearly complete) model of surface illumination must also take into account secondary illumination caused by reflection and scattering of light by nearby surfaces, and shadowing caused either by the surface facing away from light sources or by an interposing body.

Sensor models describe the photometric and geometric properties of the sensor, which determine how points in the three-dimensional world map to points in the image and how the received light is numerically encoded. They predict how a particular scene, observed from a particular viewpoint, and under particular illumination conditions, is transformed into the two-dimensional array of brightness values that constitutes the input.

Computer models usually only represent some aspects of corresponding world models of objects. The first reason for this is that the space of the computer memory is limited. The programmer must ignore some trivial aspects of the world models so that his computer models can be accommodated in a practical computer. On the other hand, ignoring some trivial aspects makes the computer vision system work more efficiently. The second reason is that we have not fully understood vision processes. Our understanding and knowledge about world models, therefore, is incomplete. We may have neglected many aspects of world models, some of which may be very important to decoding an image. The incompleteness of computer models restricts the performance of general purpose vision

systems. In many vision applications, however, the incompleteness of computer models may not be a big problem. The goals of a specific system may be achieved by the use of a subset of the knowledge of the world and the image, as well as a subset of the available techniques.

4.1.2. Low Level and High Level Vision

A vision system is naturally structured as a succession of levels of representation [BAR81]. The initial levels are constrained by what can be computed directly from the image, while higher levels are directed by the information required to support the ultimate goals. In between, the order of representations is constrained by what information is available at the preceding level and what is required by succeeding ones. For example, Kanade [KAN77] considers the representations in a vision system to be of five levels. For a region-based vision system, these levels are:

1) pixel, which is an image point;
2) patch, which is a group of contiguous pixels having similar properties;
3) region, which is a meaningful group of patches corresponding to a surface of an object;
4) subimages, which are parts of an image corresponding to an object or a set of objects consisting of several regions;
5) object, which is a real entity.

For a line-based vision system, these levels are:

1) pixel;
2) line segment, which is a group of contiguous pixels separating two contrasted patches;

3) line, which is a group of adjacent line segments with similar directions;

4) subimage which is a set of lines corresponding to an object.

5) object, which is a real entity.

Processing in the pixel-to-patch level is often called low level processing while the region-to-subimage level is high level. The patch-to-region level may be called intermediate level. Sometimes the intermediate level may also be considered as a part of the low level. Generally speaking, low level vision refers roughly to the study of those processes which operate close to the numeric arrays of sensory data which represent an image [HAN78]. High level vision refers to the study of those systems necessary for interpreting the relevant components of an image in the context of the goals and the prior knowledge of the system.

4.1.2.1. Low Level Vision

The primary task of low level vision is to segment an image into some basic features such as edges, regions and so on. For some vision systems, these features can be used by the high level vision facility directly. For others, these features must be reorganized into geometric primitives such as surfaces or volumes, depending upon the requirement of the high level vision facility.

The basic concept of segmentation can be viewed as building a description of the patterns of intensity, colour, etc., in an image [ZUC78]. Since these patterns can be described in two complementary ways, i.e. as similarity patterns and as difference patterns, two

different classes of approach have evolved to compute the low level descriptions. The first class is region growing, which has been developed to take advantage of the similarity relationships over patterns. The second class, which attempts to find the edge and the line content in an image, is based upon local difference (e.g. [HOR73]). Since these two classes of algorithms operate on data models which are strongly complementary, many systems attempt to use one technique to improve on the result of the other. For example, edge-based heuristic rules can be used to enhance an ambiguous response from a region growing algorithm, while region-based growing rules can be used to improve the interpretation of responses from edge detection operators. As low level vision has to deal with many sources of ambiguity and noise, relaxation techniques are also used (e.g. [DAV78], [ZUC78]).

4.1.2.2. Intermediate Level Vision and Intrinsic Characteristics

Many current vision systems segment an image by the use of intensity directly to establish pictorial features, such as regions of uniform intensity or step changes in intensity as an initial level of description. They then jump directly to descriptions at the level of complete objects. This approach is straightforward. However, it is not suitable to general purpose vision or high performance special purpose vision. The features obtained in this way are not reliable enough. Matching these features to a large number of object models is difficult, and there is no way to cope with objects which have no explicit models. In order to overcome these problems, some authors suggest introducing intermediate level representations of image data. For example, Barrow and Tenenbaum [BAR78] suggest using intrinsic characteristics, rather

than intensity values, in segmentation.

Intrinsic characteristics such as colour, range, orientation, reflectance, etc. give a more invariant and distinctive description of surfaces than raw light intensities. Thus, they greatly simplify many basic perceptional operations, and make some tasks possible. For example, if an object is unknown to a vision system, then it can be described in terms of invariant surface characteristics and subsequently recognized from other viewpoints.

The main problem in recovering intrinsic scene characteristics is that a single intensity value encodes all the intrinsic attributes of the corresponding scene point. While the encoding is founded upon the physics of imaging, it is not unique, and the measured light intensity at a single point could result from any of an infinitude of combinations of illumination, reflectance, surface orientation, and observer viewpoint. In order to decode the compounded information, it is necessary to make assumptions about the world and exploit the constraints they imply. In images of three-dimensional scenes, the intensity values are constrained by various physical phenomena. For example, surfaces are continuous in space and often have approximately uniform reflectance. Thus, distance and orientation are continuous. Incident illumination can also be assumed to vary smoothly except at shadow boundaries. Therefore, step changes in intensity usually occur at shadow boundaries or reflectance boundaries. Intrinsic surface characteristics are continuous through shadows. These kinds of physical phenomena and assumptions, together with interactions among fragments resulting from assumed constraints, can potentially lead to a unique interpretation of the whole image [BAR78]. Although individual components of intrinsic

characteristics have been investigated (e.g. [HOR75]), there are no successful integrations and there are no practical systems working on the basis of this.

4.1.2.3. High Level Vision

The information provided by low level vision (or intermediate level vision) describes the scene iconically in a viewer centered coordinate frame. By the use of this information, high level vision produces a more concise symbolic representation that captures global properties in a viewpoint independent coordinate frame.

Given a description of a scene in terms of surface, volume or line primitives, the system must be able to recognize instances of objects in order to explain the scene or achieve other goals. Objects are modelled by three-dimensional configurations of surface, volume or line primitives either geometrically or symbolically. Geometric models are most appropriate for describing specific objects, particularly man-made objects with regular structures; while symbolic models are appropriate for natural objects that are better defined in terms of generic characteristics than precise shape. Object recognition involves matching an object model to scene description to determine the identity of the object and its position in space. The nature of the recognition process depends upon the form of the object model.

When descriptions of scenes and object models are represented symbolically by graph structures, recognition can be formulated as a graph matching problem. Since scene descriptions are fragmented and include

many partially obscured objects, it is necessary to match parts of the scene graph with parts of object graphs. In order to minimize the combinatorics of such subgraph matching, techniques such as maximal cliques finding algorithm [BAR76, BOL80], and filtering techniques [TEN77] have been developed.

4.2. The Use of Vision in Industrial Robotics

For various reasons much uncertainty exists in the real world in which the robot operates. For example, the real world is so complex that it cannot be represented completely and adequately using currently available techniques [KEM83]. Objects which the robot is to deal with may not be exactly as designed because of design tolerances and manufacturing error. The objects may not be delivered exactly to the expected place because of the inaccurate operation of the workpiece feeder. The robot may not operate as precisely as required due to such things as mechanical tolerance in its parts and imperfections in its feedback transducers. Many robot tasks are so designed that the uncertainty in the world is tolerable, such as in painting and spot welding tasks. For tasks which require more precision, such as assembly tasks, some special methods can be adopted. For example: the quality of workpieces can be improved; workpiece positioning can be done with jigs and fixtures; robot accuracy can be improved by building robots more along the lines of NC machine tools. However, these are all expensive propositions, especially since all these measures would be required in concert. The alternative way is to use sensors to detect the differences between the real world and the robot system's idea of the world and to correct the robot's operation in subsequent actions. Vision, as an important kind

of noncontact sensor, has an important part to play in an intelligent robot system.

Vision systems are used with the robot mainly for detecting, identifying and locating objects in assembly tasks, inspecting situations, and guiding the recovery from catastrophes. They can also be used in manipulator servos in assembly [AGI77, SAR81, GES83] and in arc welding [CLO82].

Generally speaking, vision tasks in industrial robotics are simpler than those of general purpose systems. The objects that the vision system needs to observe are usually man-made workpieces with regular features and structures. The environment in which the vision system operates can usually be controlled. The robot system usually has prior knowledge about the appearance and expected location of objects in the scene. All these make it possible and easier to adopt some engineering methods to solve vision problems in robotics. As general purpose vision techniques progress and the cost of the hardware declines, the trend is to use more general vision techniques in industrial vision systems. These techniques will enhance the capability of industrial vision systems, enabling it to solve some complex problems that the engineering solutions cannot do.

4.2.1. Obtaining Descriptions of Parts

There are two ways for the vision system to acquire knowledge about the geometry of the objects to be identified and located. The first is training by showing prototypes of the objects, and the second is by

providing models of objects.

4.2.1.1. Training

In the training mode during the teaching phase the vision system is shown the prototype of every object which is to be recognized by the system. The named prototype is usually shown several times, each time in slightly different positions. The system extracts special features of the prototype, such as the length of the contour, the longest radius from the center, the shortest radius, the angle between the radii, the number of straight edges, corners, the number of holes. These features can be further processed: for example, they may be clustered. The knowledge obtained in this phase can be used at run time to recognize and locate objects by matching taught features with observed features.

The training mode is straightforward and has been used in some commercial vision systems (e.g. [UNI80]). However, this mode has disadvantages. The main disadvantage is that there is no ready way in which the user can enhance the capabilities of such a system and all objects likely to be encountered have to be taught. Due to the nature of the information, this mode can only be used in two-dimensional vision systems, for in three-dimensional space, considering the infinitude of appearances of an object, training is almost impossible. Furthermore, the vision system cannot be used in any productive work while it is being taught to recognize objects. Therefore, efficient use cannot be made of the vision equipment.

4.2.1.2. Geometrical Models

Geometrical modelling is one way of modelling rigid bodies. It has been widely used in computer aided design (CAD), computer aided manufacturing (CAM), robot programming and some other aspects of the modern technology. In the specific context of computer vision and graphics, geometrical modelling refers to the construction of computer representations of physical objects so that, together with knowledge about cameras, the physics of the imaging process and light source, predictions can be made about the images [BAU74]. Geometrical modelling can be space oriented in which objects to be modelled are represented in terms of space primitives such as elements of a three-dimensional space array, three-dimensional density functions, or two-dimensional surface functions. More convenient modelling systems are object oriented ones in which objects are modelled in terms of geometrical properties of objects. Since this kind of modelling is easier for the user to use and more compact in computer representation, it is commonly used. Properties used in modelling may be surface primitives like polygons, volume elements like generic cones, or others like skeletons.

Geometrical models usually provide more information about the objects that the vision system is to deal with than that provided by the training method and they can be used not only in two-dimensional vision systems but also in three-dimensional systems. It is quite likely that geometrical models of all objects to be observed by the vision system will already have been constructed because they will have been required elsewhere in the total manufacturing system (e.g. in their design, machining, and in the robot programming). Using geometrical models in vision systems in robotics, therefore, is efficient in terms of both the

human effort and the vision equipment.

4.2.2. Industrial Robot Vision Systems

For a computer vision system to be applied in industrial automation with the robot it has to be cost-effective, which means that its speed and reliability have to be high with respect to its cost [BOL81b]. The industrial vision systems usually meet these criteria by taking advantage of task-specific engineering for simplifying the scene to be analyzed. For example, the lighting conditions can usually be controlled so that shadows can be avoided. The contrast between the objects and the background can be strong so that the geometric features can be found more reliably by simple low level vision operators. Since the robot system usually has a large store of prior knowledge about the appearance and location of objects in the scene, the vision task can be directed better. Simple and cheap instruments may also be adopted in order to make the vision system cost-effective. For example, in Taylor's system [TAY82a] a 32 x 32 bit light sensitive array is mounted behind a 13 mm lens as a camera. This makes the system cheap both in instrument cost and in computation time since the resolution is low. The following section discusses some of the vision systems currently in use.

4.2.2.1. Two-Dimensional Vision Systems

The performance of currently available vision systems in robotics is still limited. Most current commercial vision systems such as VS-100

and AUTOVISION 1 are classified by Loughlin [LOU81] as of the first generation. Usually these systems can recognize and locate objects within a second and are mainly suitable for circumstances where objects can be identified by their silhouettes. The general characteristics of this class of vision systems are:

1. The image that the system deals with is binary. This usually means that the object must be in strong contrast to its background so that its silhouette image can be easily obtained.
2. The objects must be separated from their neighbours.
3. Each object must have a limited number of stable states in which it can rest on a horizontal surface.

The restrictions on the scenes and lighting conditions are designed to maximize processing speed by minimizing the data required and simplifying the decision-making procedure. These restrictions may be expensive in some cases and limit the use of the vision systems.

The constraint of working on binary images is largely a function of the computation time available for recognition in the environment. Complex segmentation (either line based or region based) based on grey level images is currently capable of separating a component from realistic backgrounds, but the computation time on serial computers is very long. In order to reduce the computation time, parallel processing is necessary.

The constraint of non-touching objects results partially from the binary data of the image and partially from the requirement of the recognition algorithms used. Any systems that can cope with touching or overlapping objects must adopt more powerful algorithms than those used

by the first generation systems.

The constraint of the objects to be recognized having a limited number of stable states is due to the techniques used in two-dimensional vision systems. The thorough solution to this problem is the three-dimensional vision system.

Some algorithms and systems have been designed to solve the problems encountered by the first generation vision systems. For example, the local-feature-focus method designed by Bolles [BOL82] can solve a high percentage of instances of partially visible objects. Perkins' system can determine the position and orientation of complex curved objects in grey level noisy scenes [PER78].

Another type of vision system currently available is one that transfers an input image into another domain, such as the Fourier domain, computes a fixed set of features of the transformed image, and like the other systems, applies some sort of pattern recognition procedure to make the final decision (e.g. [KAS77]). Such systems can recognize and locate isolated objects and can compare two images for difference. It is dubious whether they can be extended to cope with the analysis of complex scenes.

4.2.2.2. Three-Dimensional Vision Systems

Vision systems need to adopt quite different methods than those used in two-dimensional systems in order to deal with three-dimensional scenes. For example, ~~some advanced systems have shown that~~ it is

important to embed models and an understanding of the scene to image transformation in an intelligent vision system. With these components, a system can predict how an object will appear given its current beliefs about the scene rather than having to rely on image models. Several systems or algorithms have existed or have been suggested which can recognize and locate three-dimensional objects. Among these systems some are designed specifically for tasks in industrial automation. Others are of more general purpose.

Luh et al. developed a syntactic method [LUH81]. The system accepts the three-dimensional geometrical information of a given object and systematically generates structure descriptive sentences for all possible topologically distinct two-dimensional views of the object. The processing time for compiling a directory from the geometrical image is long, but it is done off-line, and has to be done only once for each job assignment. The on-line determination of the two-dimensional to three-dimensional correspondence is achieved by means of directory look-up.

The IMAGINE system [FIS83] matches surface regions to object models in order to recognize and locate projections of three-dimensional objects in two-dimensional images. The approach is data driven with three major stages. The first stage matches image regions to model surfaces with the goal of estimating the three-dimensional orientation parameters for the image region. This information is used to make hypotheses about specific object surfaces. The second stage relates the hypotheses according to the structural relationships embodied in the object models. The third stage verifies that the hypothesized objects are consistent with real world constraints, such as boundary, adjacency and surface ordering. Recognition is considered successful if a set of

data is found that adequately accounts for all features of a model. It has been demonstrated that the system can recognize partially obscured objects with arbitrary orientation and make reasonable estimation of their location.

The ACRONYM system is a model driven image interpretation system. It incorporates viewpoint-insensitive mechanisms. Its performance depends upon domain-independent capabilities rather than upon special domain-dependent tricks. The operation of ACRONYM can be divided into four phases. Based on object models designed by humans, ACRONYM builds an object graph and a restriction graph. The object graph is a geometrical representation of the objects expected in the task domain in terms of both the subobjects and the spatial relations holding between an object and others. The restriction graph holds sets of constraints on algebraic expressions over parameters which are used in the object graph to specify the objects. During the prediction phase, the object graph and the restriction graph are used to produce a prediction graph. The nodes in the prediction graph are predictions of image features, and the arcs represents the relations expected to hold over features in the image. The prediction graph provides a coarse filter for hypothesizing object-image feature matches and contains instructions concerning the extraction from the image feature of three-dimensional information about the object model to which the feature has been matched. During the description phase, ACRONYM uses a "line finder", an "edge mapper" and the prediction graph to produce a picture graph. The picture graph, the prediction graph, the restriction graph, and the algebraic system for reasoning over constraints are used to build the interpretation graph during the interpretation phase. Image interpretation proceeds by matching image features to predicted features. The manner in which the

matching is done and the manner in which the system represents its knowledge allow ACRONYM to interpret partially obscured objects. The ACRONYM capabilities for domain-independent, viewpoint-independent image interpretation make it attractive for resolving real world uncertainty in robotics [KEM83].

There are some other three-dimensional vision systems which take advantage of the possibility of controlling the lighting conditions to make use of special purpose instruments and techniques. Among these special instruments the laser range finder is commonly used.

The laser range finder consists of a light source, a camera, and a computer. There are two basically different techniques which can be used to measure ranges [NIT76]: triangulation and time of flight. In one kind of triangulation range finder [POP77], the light source casts a plane of light on the scene, and this, when viewed from a different point of view by a TV camera, will appear as a broken curve, or a set of straight line segments, depending upon the nature of the surface on which the stripe is falling. One stripe gives the range of a cross section of the scene. By scanning the stripe across the scene, a complete range map of the scene can be built up. The information can then be processed by the computer so that the objects in the scene can the recognized and located, or an interpretation of the scene can be given. In a time-of-flight range finder, range is determined from the time needed for the light to travel from the light source to the target and back. The time of flight can be determined either directly by using a pulsed laser and measuring elapsed time, or indirectly by using a modulated beam and measuring the phase shift. A range image is obtained by using a scanning system to sweep the beam over the scene.

Such range finders have been used for different purposes. For example, they have been used to form three-dimensional models for objects [POP77] and to locate three-dimensional objects [BOL81a]. It is also considered that the laser range finder, together with other special purpose instruments, can fit naturally into the general perceptual framework [BAR81].

4.3. Verification Vision

The term verification vision is suggested by Bolles. It denotes a special kind of vision system whose purpose is to verify and refine the location of specific objects in the scene rather than to recognize them.

The concept of verification vision seems useful and suitable in robotics because automatic assembly is not haphazard but carefully planned. Most uncertainty in robotics is of the position rather than of the appearance of the objects. The robot system usually needs to update its knowledge about the location of the objects rather than to recognize these objects. Also, the robot system usually has a great deal of prior knowledge about the scene which can be used by the vision system to estimate the appearance of the scene. For a verification vision system, one needs some way of making the prior knowledge about the scene available to the system.

The verification vision system designed by Bolles [BOL77] uses object models and image models. It is intended for inspection and visual control in repetitive manufacturing tasks. The verification

vision system makes use of three-dimensional models, but it requires that sensed images be very similar to image models, and therefore it is necessary that the difference between the expected position and the actual position of the object be within a certain range.

There are four stages in the operation of the system. At programming time, the user states the goal of the task, calibrates the camera, and chooses potential operator/feature pairs. At training time the system applies the operators to several sample pictures and gather statistical information about their effectiveness. At planning time the system ranks operators according to their expected contribution, determines the expected number of operators needed, and predicts the computational cost of accomplishing the task. At execution time the system applies operators in their order of cost-effectiveness, combines the results into confidences and precisions, and stops when the desired confidence has been achieved, or cost limit exceeded. The system is restricted in viewpoint since it primarily depends upon small correlation windows as features [BIN82].

Bolles' system is a special case of verification vision in which he uses taught image models. It is not necessary to use taught models. Baumann [BAU81], for example, uses image models which are defined as sets of two dimensional regions. Each region represents a basic component of the expected image of the object. In the work which will be described in this thesis, the system uses the knowledge about the expected positions of the modelled objects to predict the expected positions of the edges in the image. Generally speaking, verification vision has a great deal of prior knowledge about the scene. This knowledge can be used to guide the interpretation of the visual

information. Thus, in order to refine the location of the objects to be verified, the verification vision system needs only to know the positions of the images of certain selected features. The features which can be used by verification vision can be edges, corners, holes or any others which are easy to detect and locate.

Chapter 5. New Vision Commands in RAPT

The current RAPT system, as discussed in Chapter 3, has no commands for using sensory information to perceive and interact with the surrounding environment. This is inadequate if the language is to be used for complex tasks in which unexpected events may happen, where anticipated changes cannot be determined exactly before the program is run, or where tolerances are such that special accommodation is needed for each part. The motive behind the research work which will be discussed in this and succeeding chapters is to provide a set of vision commands and associated facilities to overcome this. The semantics of these vision commands and the functions of the associated facilities are determined by the nature of the variabilities of the environment and the way in which the RAPT system needs to use sensory information. In this chapter, the author will discuss the role of the vision tasks in the RAPT programming environment and define the vision commands which are necessary in describing such vision tasks.

5.1. Vision Tasks in RAPT

RAPT is an object level language. It has some knowledge about the surrounding environment in which the robot will perform assembly tasks. The output of normal RAPT is a series of positions of bodies in each situation. These positions are, however, only planned ones in as much as they have been determined taking no account of the inaccuracies inherent in the real world. These planned positions are referred to as nominal positions. Discrepancies between nominal positions and actual positions of bodies may be caused by inaccuracies of the mechanical

structure and performance of the robot, unexpected disturbance to the bodies, tolerances in the parts and so on. Usually the discrepancies are not too large. For some tasks, the program can be designed to tolerate this amount of inaccuracy in the nominal position. However, for some precise operations, inaccuracies which are inherent in the robot and the environment may cause failures. Furthermore, if work-pieces or subassemblies are delivered by a cheap belt conveyer, part feeder or fast-moving robot with poor accuracy then the inaccuracies of the positions of the bodies may be easily beyond the tolerance of the program and the assembly robot. In these cases, some system is needed to verify and refine the positions of the bodies whose nominal positions are in doubt. A vision system can be used for this.

RAPT reasons about spatial relationships between body features. The identification of an image feature with a feature of a RAPT body in effect defines a relationship between that feature and an (imaginary) feature of the camera. This relationship, in conjunction with others, can be used to refine the estimate of the position of the body.

In order to specify vision verification tasks in RAPT programs a number of vision commands have been added to the RAPT language. They are the LOOK statement which is used to indicate the feature to be verified and to describe the vision environment, the INVIOLATE statement which is used to specify the constraints on the actual position of the body to be verified, the TOLERANCE statement which is used to specify the maximum translational error along all the three axes of the body coordinate system, and the COMBINE statement which provides a vision command package and combines the information given by the LOOK, INVIOLATE and TOLERANCE statements within the package.

Some auxiliary statements are also needed to specify cameras which will be used by the vision system. These statements specify types and parameters such as focal length and position of cameras. They are also used to specify the default camera which will be used when the user does not indicate the camera to be used in LOOK statements explicitly.

5.2. The LOOK Statement

The LOOK statement is used to indicate the feature to be verified and to describe the vision environment. The syntax of the LOOK statement is as follows:

```
LOOK/<feature of body> [,<camera name>];
```

where <feature of body> specifies the feature to be verified. This feature will already have been defined by the RAPT modelling system. The <camera name> specifies the camera to be used. The <camera name> is optional. If it is omitted, the default camera name which has been specified by the programmer will be used. For example, the statement

```
LOOK/edge_1 of block, camera_a;
```

asks the vision system to use a camera called "camera_a" to refine the position of the body "block" using "edge_1".

In the current research, edges are chosen as the feature type to be verified, since images of edges are easy to detect. However, there are

no theoretical obstacles to the use of other feature types in the verification vision system.

As mentioned in Chapter 3, the RAPT reasoning system treats edges as infinite features although some edges can be represented in the RAPT model as finite with a specified extent. The extent of the edge has no significance to the reasoning system. However, as features to be verified, the edges must be finite since the verification vision system has to know the approximate position and dimension of the physical edge to be verified. Thus, only edges which have been defined with specified length are allowed to appear in a LOOK statement as features to be verified.

The LOOK statement has three effects:

(1) formation of symbolic features and relationships in the RAPT reasoning network.
(2) sending all necessary information to the vision facility for it to decide where and how to find the expected edge.
(3) making the command to actually use the camera at run time.

These will be discussed in detail.

5.2.1. Forming the Symbolic Features and Relationships

Once a camera has been used to find a body feature in the scene, it establishes a relationship between the camera and the body. It is known, from projection transformation, that an edge in an image

corresponds to a plane in a 3-D space. If an edge feature is to be verified, an AGAINST relationship between a face which is relative to the camera and the edge will be established. This relationship is denoted as AGPE (AGAINST/FACE,EDGE), a special case of AGPC which is a standard relationship in the current RAPT system. This AGPE is introduced in order to simplify the symbolic reasoning rules and is the same as AGPC except that the cylindrical feature involved is restricted to being an edge. In the AGPE relation formed as a result of a vision command, the edge feature has already been defined by the programmer through the RAPT modelling system, while the imaginary face feature is to be created by the LOOK statement. From the definition of the relationship AGAINST, it is easy to see that the face feature is so located that both the edge feature to be verified and the centre point of the camera lens to be used must lie on the face. The position of the face feature can be so determined that its origin coincides with the centre point of the camera lens and its normal is perpendicular to any two rays which point from the centre point of the camera towards different points on the edge feature. The vectors of these rays can be calculated by the inverse perspective projection transformation [DUD73] using measurements of the positions of points on the image of the edge.

However, the vision data (that is, the position of the image of the edge feature in the scene) is not available at compile time. Therefore, the face feature only exists symbolically at compile time and its position cannot be resolved until run time. The relation AGPE therefore holds between a fully defined geometrical feature and one whose position is not known at compile time. In order to do the geometric reasoning about the position of the body at compile time, a name which is referred to as a symbolic feature will be created for the face feature. This

symbolic feature has type "face" and an unknown position. At run time a real position will be assigned to it. Relations which refer to symbolic features are called symbolic relations and will be dealt with by the symbolic reasoning system which will be discussed in detail in Chapter 6.

5.2.2. Information Used by the Vision Facilities

In order to predict the position of the image of the edge, the following items of information must be available to the vision facilities.

(1) the feature to be verified,
(2) the model of the body,
(3) the nominal position of the body,
(4) the translational tolerance of the position of the body,
(5) the physical name, position and parameters of the camera to be used.

The nominal position of the image of the edge feature of the body is easy to work out from this information. It is done in the following way. From the model the vision facilities know the exact position of the feature in terms of the body's local coordinate system. The nominal position of the body together with the body model, tells the vision facilities the anticipated position of the edge feature in terms of the world global coordinate system. The translational tolerance of the body position restricts the discrepancies between the nominal and actual positions of bodies. Therefore, the vision facilities are able to decide the range in which the expected edge image is likely to appear in

the scene. Finally, by using the inverse perspective transformation the range of possible image positions can be calculated.

5.2.3. Calling the Vision Facilities

The last effect of the LOOK statement is to create a run time command to use the camera. The run time command will specify the operation of the vision facilities and indicate the symbolic feature to which the vision data will be sent after being processed by the vision facilities.

5.3. The INVIOLATE Statement

The INVIOLATE statement specifies the constraints on the actual position of the body to be verified. The syntax of the INVIOLATE statement is

INVIOLATE/<relation>, <feature of body 1>, <feature of body 2>;

where the <relation> specifies a relationship which must hold between <feature of body 1> and <feature of body 2>. Theoretically, it can be any relationship allowed by the current RAPT input system. The <feature of body 1> is a reference feature while <feature of body 2> is a feature of the body to be verified by LOOK statements in the same vision command package as the INVIOLATE statement. The reference feature <feature of body 1> must be either a feature of the world or a feature of a body which has been fixed with respect to the world. For example, the following statement

INVIOLATE/AGAINST, top of table, bottom of block;

specifies that the body "block" is on the table and its bottom is against the top of the table.

5.3.1. Inviolate Relations in Vision Verification

The reasons for the introduction of the INVIOLATE statement are two-fold. The first and the most important reason is that the verification vision system needs an explicit declaration of the relations which must hold whatever the actual position of the body to be verified. In RAPT, the position of a body is defined in terms of relations holding between features of the body and those of others. However, not all these specified relations will actually be realized in physical situations. Some of the relations are vulnerable to inaccurate movement, tolerance, etc. while others are guaranteed by certain physical constraints. For example, if a body is at rest on a flat table and not supported by any other bodies, then its bottom must be against the table top. On the other hand, the user may have intended its front surface to be coplanar with some other surface. Whether it is actually coplanar depends upon how the user asks the robot to put it there. The aim of verification is to ascertain the actual positions of bodies, and these may differ from nominal ones because some relations have not been properly achieved. If there are no explicit statements about which relations are inviolate (i.e. must hold whatever the history of the assembly) then the symbolic reasoning system will explain, without any restriction, the vision data it has obtained about the positions of

verified features of the body. The explanation may conflict with the physical constraints on the position of the body to be verified. For example, suppose the vision system verifies the position of a block B which lies on the top face of a work table, and the top face is parallel to the X-Y plane of the world coordinate system. Six parameters are needed to determine the position and orientation of a body in a Cartesian coordinate system: three for translation and three for rotation. If the position of the block is to be verified without any restriction over its actual position, then the six parameters are all subject to modification and since it is almost certain that there will be some errors in the vision system, there may be some peculiar results. For example, the result may indicate that the bottom of the block was beneath the top of the table though the block lies on the top of the table. On the other hand, if an INVIOLATE statement is used to indicate the fact that the bottom of the block must be "AGAINST" the top of the table then the explanation of the vision data is constrained in such a way that the parameters for translation along the Z-axis and rotation about the X- and Y-axes are fixed and no matter what the vision data is, the relationship "AGAINST" between the bottom of the block and the top of the table must be kept. The INVIOLATE statement specifies a constraint on the position of the body to be verified in terms of a relationship that must hold between the body and another. From the view point of geometric reasoning, it provides a reliable relationship in the relationship network.

If the reference feature does not directly belong to the world then it must belong to a body whose position has been determined completely with respect to the world. This is so that the position of the reference feature can be expressed in terms of the world coordinate system.

The second reason for introducing the INVIOLATE statement is to do with the capability of the reasoning system. If all the relations which will be used to deduce the position of the body to be verified are of the AGPC type then the cycle finding reasoning system cannot usually do any inference. This is because the relationship AGPC or AGPE contains four degrees of freedom: two translational ones and two rotational ones. There are no specified relations holding between the coordinate axes of the cylindrical feature and those of the face, except that the two X-axes must be perpendicular to each other. This characteristic of the relationship AGPC or AGPE brings about some ambiguities which can be eliminated only when some special conditions hold. The combination of relations consisting of AGPC or AGPE type alone is therefore possible only for a few special cases [POP81]. The general case of combining two or three AGPC relations gives a number of possible solutions and even the equation solving system cannot disambiguate them. Of course there are no suitable entries in the reasoning rule table of the cycle finding system. The INVIOLATE statement helps by providing other types of relations which enable the inference system to reach a solution.

In order to demonstrate the idea of using inviolate relationships in reasoning about vision data, the relation type AGPP, i.e. an AGAINST between two plane features, is employed in the INVIOLATE statement. The relationship AGPP is used because it is the most common situation encountered in a real assembly process, and the combination of an AGPP and an AGPE is solvable under the verification condition. The necessary reasoning facilities have been implemented in the current system. However, there is no theoretical restriction on the use of other kinds of relationships. The only practical restriction to this is the capability

of the symbolic reasoning facility. The type of relationship used in the statement can be any one allowed by the RAPT system, provided the symbolic reasoning facility can deal with it.

5.3.2. Local and Global Vision Command Package

In a RAPT program some bodies may be "TIED" together throughout a number of contiguous situations. This means that these bodies must keep the same relative positions before and after each action. Usually the relative positions of the TIED bodies stay the same before and after a set of vision commands. Sometimes, however, the user needs the vision system to verify and modify some relations between two bodies which have been TIED together. For example, the programmer can specify an operation of picking up a shaft by saying that the gripper moves a certain distance along a specified direction, and after this action is completed the two face features of the gripper are against the shaft. He then says that the gripper and the shaft are TIED together before the gripper moves upwards. If at this step the programmer is not convinced of the relative position of the shaft with respect to the gripper, for example, because the position of the shaft on the table was uncertain, then he can order the gripper to move to the front of a camera and ask the vision system to refine the position of the shaft. The actual position of the shaft in front of the camera may be quite different from the nominal one, but the relations that the face features of the gripper are against the shaft must hold provided the shaft has really been picked up. The programmer can use INVIOLATE statements to indicate this restriction. In this case the result of the verification may change the relative position of two bodies which have been TIED (e.g. the shaft

shifted along its axis) but the relations declared by INVIOLATE statements will still hold.

In a future implementation, some new statements could be introduced to indicate explicitly which TIES must be kept during vision verification and which TIES are subject to modification. At the moment, the status of a TIE during vision verification is deduced from the INVIOLATE statements. If the reference body (BR) in an INVIOLATE statement in a vision command package has been TIED in the associated RAPT program to the body to be verified (BV) then the relative position of BV with respect to BR is subject to modification. Otherwise the relative positions of the two bodies which have been TIED together will not be changed.

Any INVIOLATE statement referring to two bodies which have been TIED together is called a local INVIOLATE statement and, as discussed above, it indicates that changes may occur in the relative positions. A similar treatment is also applicable to subassemblies. An INVIOLATE statement in which BR belongs to the same subassembly as BV is also called a local INVIOLATE statement. A vision command package which contains a local INVIOLATE statement is referred to as a local vision command package and a vision command package which does not contain a local INVIOLATE statement is referred to as a global one. The verified position obtained from a local vision command package will only affect the position of BV and therefore the relative position of BV with respect to any other body is subject to modification (but the constraints of the INVIOLATE relationship are maintained). Detailed discussion of the influence of these two kinds of vision command package over body positions can be seen in Chapter 8 where "linking rules" for TIE and

SUBASSEMBLY statements are discussed.

5.4. The TOLERANCE Statement

Verification vision works under the assumption that the position error between the nominal position of the body to be verified and its actual position is not very large. This assumption is important especially to the verification vision system working together with the current RAPT system. As mentioned before, the modelling system used by the current RAPT system is incomplete, and therefore it cannot provide enough information for the verification vision system to judge whether a mismatch is made between an image and an model feature. For example, an image of a feature which is not represented by the body model may be considered as that of a model feature whose real image is near by, and this mismatch cannot be discovered by the system by consulting the model of the body. In order to assure the correct match between the image and the feature, the position error between the nominal position and the actual one of the body should not be too large so that in the whole range on the scene in which the expected feature may appear no other features may appear. This needs the programmer to select the features to be verified carefully and reckon the maximum possible positional error correctly. The estimated error should not be so small that the expected image may fall outside the range suggested by the error. It also should not be so large so that image of other features may fall into the range.

In order to express the maximum possible error in a nominal position, a new statement TOLERANCE is introduced. This statement indicates

the possible translational error range of the nominal position of a body along each coordinate of the world frame. It has two possible syntactic forms. The first one is

TOLERANCE/<body>,TRAN, <no>;

where <body> is the name of the body for which the statement specifies the maximum deviation from the nominal position, and <no> is a positive real number. This form of the statement is referred to as a global tolerance statement. It is used outside any vision command packages and is valid throughout the associated RAPT program. The second form of the statement is

TOLERANCE/TRAN, <no>;

where <no> is a positive real number. This form of the statement is referred to as a local tolerance statement and can only be used within a COMBINE command package. It specifies the tolerance of the nominal position of the body which is to be verified by the package of vision commands, and has no effects outside the package. If there is no tolerance statement given to a body then a default tolerance which has been set by the vision system will be used to restrict its nominal position.

The rotation error tolerance will be discussed in Chapter 10 of this thesis. There is no implementation of rotation error in the current system. However, the restriction of the translation error should take into account the effects of possible rotational errors on the nominal position and indicate the range in which the feature is likely to be found.

5.5. The COMBINE Statement

The COMBINE statement provides a package for the vision commands. It invokes the symbolic reasoning facility to deduce the symbolic position of the body by using all the information included in the package. To this end, it must check whether the statements in the package are compatible, and combine information given in the TOLERANCE and INVIOLATE statements in order to deduce a more accurate translational error tolerance over the nominal position of the body to be verified. The COMBINE statement is also used to declare a new situation in the assembly task.

5.5.1. Checking the Statements

There are two points that will be checked. Firstly, all the LOOK and INVIOLATE statements in the package must create relationships between one particular body and the world. This means that the features to be looked for by the LOOK statements must belong to the same body and this body must be the same as BV of the INVIOLATE statement. Also the reference features specified in the INVIOLATE statements must either belong to the world or belong to a body which has been fixed with respect to the world. If the reference features do not belong to the world directly but belong to bodies whose positions have been determined in the world coordinate system then the positions of these features will be transformed into the positions in the world coordinate system. Secondly, if there is more than one INVIOLATE statement, they must not conflict with each other.

5.5.2. Restricting the Error Range Over the Nominal Position

The TOLERANCE statement specifies the maximum translational error along all three axes of the body coordinate system. This does not necessarily mean that the actual position of the body can differ from its nominal position by this amount in any of the three directions. In fact, the range of positions is restricted by the constraints described by the INVIOLATE statements. The information given by the TOLERANCE statement should therefore be combined with the information obtained from the INVIOLATE statement(s) and an actual error tolerance on the nominal position of a body deduced. This actual tolerance gives the vision facilities more accurate data to decide where to expect the image of the feature. Because most of the cases that the current vision verification system deals with are very simple, it does not need to use sophisticated methods to deduce an estimate of the actual error tolerance. For example, consider a verification vision process in which the bottom plane of the body to be verified is against the top plane of the work table which is parallel to the Y-Z plane in the world coordinate system. In this case the possible error of the nominal position of the body along the direction of the X-axis of the world coordinate system is zero. Occurrences of this kind are quite common in RAPT programs since RAPT is used to describe assembly tasks and most workpieces have regular and simple shapes. In order to deal with these simple cases efficiently the COMBINE command uses some rules of thumb to calculate the actual error range along each coordinate axis. These rules of thumb produce exact results for the simple cases and approximate results for general circumstances.

When working out the more accurate error range the system uses the local tolerance statement, if one has been given. If not, it looks for a global tolerance statement for the relevant body, and failing this, uses the default value. Having decided on the tolerance, it can now use the INVIOLATE statements and rules of thumb to determine the more accurate error range. These rules of thumb depend upon the number of INVIOLATE statements in the current COMBINE package.

When considering the case in which the INVIOLATE statements are all AGPP ones, three different circumstances can be distinguished. In the first there are three non-redundant INVIOLATE statements and the nominal position of the body is accurate and the error range is zero along any coordinate axis. If there are two non-redundant INVIOLATE statements then the body can move in a direction which is perpendicular to the X-axes of both reference faces in the INVIOLATE statements. This corresponds to a LIN degree of freedom. The rule for two INVIOLATE statements is as follows: suppose the cross vector of the X-axes of the two reference faces in the INVIOLATE statements is (a, b, c), then the combined tolerance will be a vector

$$(|d*a|, |d*b|, |d*c|) \qquad (5.1)$$

where d is the nominal tolerance given by the corresponding TOLERANCE statement. This rule gives an accurate result for the actual error range along each coordinate axis when the direction in which the body is movable is parallel to any coordinate axis of the world. In other cases the result is approximate. When there is only one INVIOLATE statement, or two with the reference faces parallel to each other, the position of

the body to be verified may deviate in any direction perpendicular to the X-axis of the reference face. In this case the rule is as follows: suppose the vector of the X-axis of the reference face in the INVIOLATE statement is (a, b, c) then the combined tolerance is a vector

$$(d*(1-a*a),\ d*(1-b*b),\ d*(1-c*c)) \qquad (5.2)$$

Here d is the nominal tolerance. If the X-axis of the reference face is parallel to any axis of the world coordinate system then the algorithm produces an accurate result otherwise it produces an approximate result.

Precise algorithms for calculating the actual error range in general situations may be considered in the future. These rules do, however, depend upon the RAPT system having a knowledge of the space occupancy of the objects, and of some physical laws. Since RAPT does not yet have these, the precise algorithms cannot yet be implemented.

5.5.3. Creating a New Situation

Although the whole vision command package causes neither any actions of the robot manipulator nor any changes of the nominal positions of the bodies, it does change the knowledge of the robot system about the actual positions of the bodies. It therefore changes the state and following actions of the robot. Thus it is necessary for a vision command package to be considered to be a special action command and to create a new situation to distinguish the states of the robot before and after vision verification. Doing this is also a task of the COMBINE statement.

An example of the use of the vision commands is as follows:

```
COMBINE;
  INVIOLATE/ against, bottom of body1, top of table;
  LOOK/ edge1 of body1, camera1;
  LOOK/ edge2 of body1, camera2;
  TOLERANCE/TRAN, 6;
TERCOM;
```

where TERCOM terminates the COMBINE package.

5.6. The Camera Specification Statement

The basic camera model has eight degrees of freedom [BAU74], three in location which indicate the position of the lens centre of the camera, three in orientation which indicate the directions of the axes of the camera coordinate system, and two in projection which indicate the focal ratio and aspect ratio respectively. In the vision verification system, the camera model has seven degrees of freedom. The parameter of aspect ratio is embedded into the system for the currently used equipment so that the user does not need to worry about it. A camera is defined by a statement of the form:

<camera-name> = CAMERA/<name>, <body name>, <F>;

where <name> is the logical name of the physical camera to be used,

<body name> indicates an already defined body whose coordinate system is used as that of the camera and <F> is the focal length of the lens. The axis of the camera lens is assumed to be collinear with the X-axis of the body coordinate system and the centre of the camera lens is assumed to be coincident with the origin of the local coordinate system. This type of statement is referred to as the general camera specification statement, and cameras defined in this way can be moved by the robot.

For convenience, another type of camera specification statement has been introduced. Its format is

<camera-name>=CAMERA/<name>,<centre>,THETA,<no 1>,PHI,<no 2>,
PSI,<no 3>,<F>;

Here <centre> is a point indicating the position of the centre of the camera lens, and <no 1> - <no 3> indicate the rotation of the camera about its coordinate axes in the order of the Z-, Y- and X-axes. The amount is measured in degrees. The camera reference orientation is with the X-axis of the camera coordinate system parallel to the Z-axis of the world coordinate system, and the Z-axis of the camera parallel to the X-axis of the world. This is illustrated in Fig. 5.1. Cameras defined in this way cannot be moved or manipulated by the robot. This type of the statement is therefore referred to as an immovable camera specification statement.

When a LOOK statement does not specify explicitly the camera to be used then a default camera is used. A default camera setting statement has the format

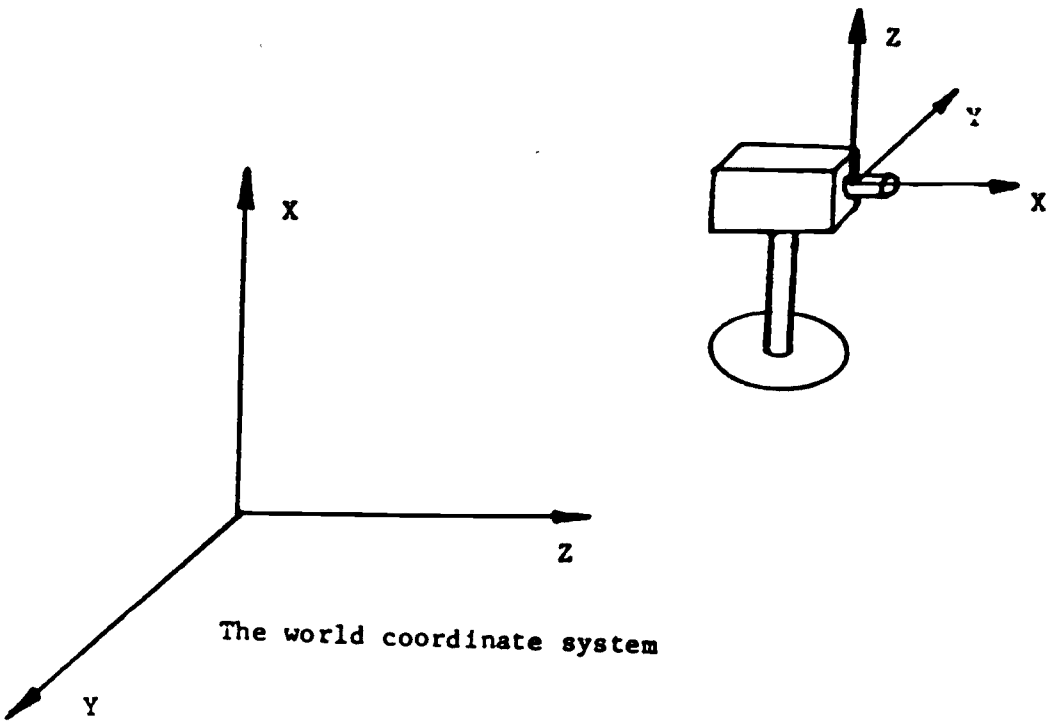


Fig. 5.1 The relationship between the world coordinate system and the reference coordinate system of the camera

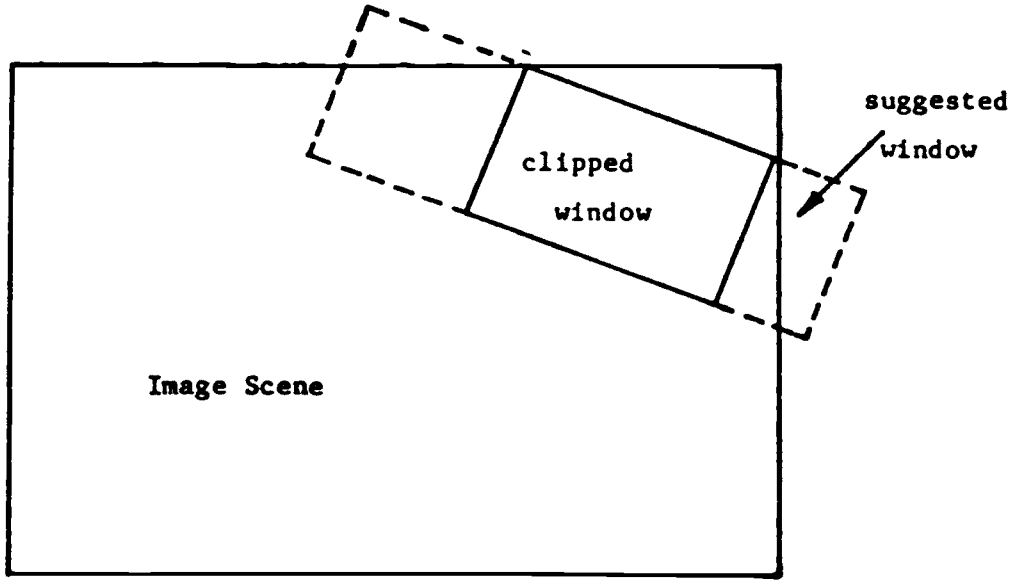


Fig. 5.2 Window clipping

SETCAMERA / <camera-name>;

In the default case the camera set by the latest SETCAMERA statement will be used.

5.7. Vision Facilities

The vision facilities used by the verification vision system include a vision interface and a low level vision processor. They are used as an experimental tool for obtaining real vision data for the use of the verification vision system. At present, the vision facilities contain three parts: a window suggester, an edge finder and a face generator. The window suggester works under the control of the camera commands created by LOOK statements. It tells the edge finder where to look for the image of the edge feature. The vision data obtained by the edge finder is used to determine the position of a new face of the world by the face generator. The actual position of the face is then sent to the RAPT run time system to take the place of the symbolic position of the corresponding face feature created by the LOOK statement at compile time. When positions of all relevant symbolic features have been replaced by actual ones the symbolic position of a body being verified can be evaluated to produce the actual verified position.

5.7.1. The Window Suggester

The window suggester needs the body model, the nominal position of the body, the positional error tolerance and the position of the camera.

The position of any point in the world coordinate system can be represented in terms of the camera coordinates by making use of the position of the camera. Suppose that the position of a point is Pw in the world frame and Pc in the camera frame and C is the position of the camera, and that both Pw and Pc are represented in the form of a homogeneous row vector (x,y,z,1). Then the following equations stand

$$\therefore \quad Pc ** C = Pw \tag{5.3}$$

$$\therefore \quad Pc = Pw ** C^{-1} \tag{5.4}$$

Using the focal length F of the lens, the position of the image of the point can be calculated. Suppose a point has a position Pc = (a,b,c,1). The coordinates of the image of the point can be evaluated by the following equations.

$$y = b.a/F \tag{5.5}$$

$$z = c.a/F \tag{5.6}$$

Using equations (5.4), (5.5) and (5.6), the range of the image position of the edge to be verified under the constraint of the position error tolerance can be worked out in image coordinates and a window can be created. However, before the window suggester creates the window, it examines whether there is any part of the nominal edge image falling within the range of the scene projection, given the current camera parameters. This scene projection is determined by the vision equipment. If the entire image of the edge to be verified is out of the range of the scene projection then the window suggester will report an error to the programmer, and no window will be created. Otherwise a

window will be suggested for the visible part of the image of the edge. A window is a rectangle. It is defined in such a way that a part of the edge image will definitely appear within the window, provided that the position deviation of the body is within the tolerance. The image will also be approximately aligned with the long side of the rectangle. Before the window is sent to the edge finder, it will be clipped. The clipping process first inspects whether any long side boundary of the window is completely out of the image scene. If so this will be reported to the programmer and the window will be abolished. Otherwise the clipping process will clip the window so that only the area of the window which is within the image scene will remain. The clipped window is kept rectangular (Fig. 5.2).

5.7.2. The Edge Finder

The edge finder is a low level vision processor. Given a digitized grey level image and a rectangular window, the edge finder will look for an image of an edge in the window and report the coordinates of the end points of the edge image. The edge finder works in the following way. Firstly, it searches from two short sides of the window towards the centre in order to find some candidates for the end points of the expected edge image. An end point is defined as a pixel whose "weight" (see below) exceeds a specified threshold. If some end points have been found then it calculates the average weight of the linking pixels between each possible combination of end points. It then compares all the average weights in order to decide which pair of end points gives the line segment with the heaviest average weight, and reports the coordinates of the pair. If a segment of a curve appears in the given

window then the edge finder will report the coordinates of the end points of a single straight line which approximates to the curve segment. The main advantage of this method is that it is able to reduce the effects of discontinuities in the image of an edge caused by noise and imperfection of the camera and digitizer. The weight of a pixel is evaluated by a high pass filter suggested by Wong [WON79]. The equation of the high pass filter is

$$i' = | a+b+c-d-e-f | + |c+h+f-a-g-d| \qquad (5.7)$$

This equation is applied to the window shown in Fig. 5.3 where i coincides with the pixel to be processed. The threshold used for distinguishing the end points is set by the programmer before the vision facilities begin the operation.

a	b	c
h	i	d
g	f	e

Fig. 5.3 The window of the high-pass filter

Other kinds of low level vision processing method could be used instead of this edge finder, provided they can determine the end points of the image of the expected edge correctly. Up to now, however, no other methods have been tested since the current edge finder produces satisfactory results as an experimental tool under controlled laboratory conditions.

5.7.3. The Face Generator

The face generator receives vision data from the edge finder. It uses the data to decide the position of the face feature which has been created symbolically by the corresponding LOOK statement relating the camera and the edge to be verified. Suppose the coordinates of the image of a point is (y,z) and the focal length of the lens is F. The vector of a ray which points from the centre of the lens to the point is (F,y,z). Thus, knowing the positions of the images of two points of the edge being verified, the vectors of the two rays which point from the centre of the camera towards the points of the edge can be determined. A position for the face feature is created with its normal perpendicular to the vectors of both rays. The point which is used to decide the origin of the face is coincident with the origin of the camera coordinate system. Using the knowledge of the position of the camera, the position of the face feature is transformed from the camera coordinate system into the world frame.

Chapter 6. Symbolic Geometrical Reasoning About Vision Data

Symbolic reasoning is an inference process in which reasoning rules are applied to uninstantiated variables rather than instantiated ones. The purpose of this inference process is to find out general relationships among variables rather than to calculate special results for some particular assignment of the variables of the reasoning. The results of this kind of reasoning are usually of the form of symbolic equations. The technique of symbolic reasoning has been applied in a number of areas such as electronic circuit analysis [LIN73] and synthesis [KLE78], image analysis [BRO81] and error constraints in robot operation planning [BRO82]. In the RAPT verification vision system, symbolic reasoning is adopted to carry out geometrical reasoning among symbolic relations created by vision commands.

6.1. Symbolic Reasoning Facility in RAPT

The RAPT system reasons about the spatial relations between body features at compile time. For ordinary RAPT programs without vision commands spatial relations are all actual ones with instantiated feature positions, and the reasoning is done by the current implementation of the inference system: the cycle finder. However, when vision commands are introduced in the RAPT program, things are different. At compile time, relations created by LOOK statements are symbolic ones with uninstantiated feature positions. The positions of face features have only a symbolic form and cannot be instantiated until run time, when the vision data is acquired. In this case, the reasoning system has to deal with the symbolic form of the feature positions rather than their

actual values. The result of the process will be evaluated during run time when the real feature positions have taken the places of the symbolic ones. The requirement of reasoning about symbolic relations is beyond the capability of the current cycle finder. A symbolic reasoning facility is therefore essential for combining vision data with the RAPT system.

The symbolic reasoning facility which has been added to the RAPT system is not intended to take the place of the cycle finder: it is only used for reasoning about relations created by vision commands (and in the future possibly other sensors); the cycle finder is still used to handle relations generated by ordinary RAPT statements. Thus the symbolic reasoning facility needs only to deal with a subset of the relation types handled by the cycle finder.

The symbolic reasoning system discussed in this chapter does not apply any special algorithms to deal with symbolic relations. Instead, it provides a general symbolic operation facility so that the necessary geometrical reasoning rules can be applied to symbolic relations. Since relations created by a vision command package can all be represented as holding between the world and the body to be verified, the size of symbolic relation cycles can be limited to two.

The symbolic reasoning system works in a similar way to the current RAPT cycle finder. Being given two relationship chains between two bodies, it will produce a more constrained new relation between the bodies by means of a set of combination rules. The main difference between the symbolic reasoning system and the cycle finder is that the features in both the input and output of the symbolic reasoning system

may be symbolic and their positions may be expressed by symbolic expressions. However, when both the input relations are actual ones with instantiated feature positions the symbolic reasoning system behaves exactly the same as the cycle finder, and its output is also an actual relation with instantiated feature positions. If any one or both of the input relations are symbolic then the output of the symbolic reasoning system is a symbolic relation with feature positions represented by symbolic expressions.

A limitation of the symbolic reasoning facility is that it will not handle conditionals which may change the subsequent reasoning route. In the current RAPT system, a large number of conditionals are encountered during geometric reasoning. The presence of the conditionals is usually due to the possibility of special relationships holding between some body features, and while some conditionals may only change the reasoning rules being applied and the parameters of the consequent relationships others may also change the nature of the new relationships resulting from the reasoning, and therefore change the subsequent reasoning path. The latter type of conditional is referred to as a top level conditional. For example, in Table 1 in [POP81] when a LIN and an AGPC are combined together, the general result will be a FIX. If the X-axis of the face feature in the AGPC is parallel to that of the edge feature of the same body in the LIN then the result is still a FIX but different reasoning rules are applied. Since this conditional does not change the nature of the resulting relation and hence does not change the subsequent reasoning route, it is not a top level one. However, in the same reasoning process if the X-axis of the plane feature in the AGPC is perpendicular to that of the edge feature of the same body in the LIN, then the result will be a LIN. This conditional is therefore a top level

one, since it changes the nature of the resulting relation and the subsequent reasoning route.

In symbolic reasoning, if a conditional can be evaluated at compile time, i.e. the logic value of its condition can be determined then it can be handled at compile time no matter whether it is a top level one or not. Otherwise the conditional must be kept in a symbolic form and will be dealt with at run time. If a symbolic conditional is a top level one then the nature of the resulting relation cannot be determined at compile time and therefore the subsequent reasoning cannot be carried out. Fortunately, most of the top level conditionals which may appear in symbolic reasoning because of vision verification can be evaluated at compile time since they involve questions of geometrical relationship of features of known bodies. There are a few top level conditionals which will inevitably have a symbolic form. In the current implementation the user is expected to avoid them by carefully selecting the edge features to be verified. There are two ways in which the problem can be solved in the future. On the one hand, the capability of the symbolic reasoning facility may be expanded to handle all top level symbolic conditionals. On the other hand, the capability of the whole verification vision system may be expanded so that it will be able to use a complete modelling system to select automatically features to be verified in order to avoid any potential top level symbolic conditionals.

6.2. Bottom Level Versus Top Level Symbolic Reasoning

There are two main ways to implement the symbolic reasoning facility. They can be referred to as the bottom level symbolic reasoning and

the top level symbolic reasoning respectively. In fact they are at the two extremes of a range, and intermediate levels could be used. In an inference system reasoning rules are encoded in a certain computer language as functions or procedures, and a function may call other functions or operations. The functions which do not apply other functions can be referred to as bottom level functions while the functions which apply other functions can be referred to as higher-level functions. The functions which encode the top level performance of reasoning rules are referred to as the top level functions. In bottom level symbolic reasoning, the higher-level functions are applied in the normal way, and call other functions. It is only the bottom level functions which, when applied to symbolic arguments, change their performance and produce proper symbolic results. In top level symbolic reasoning, on the other hand, the top level function tests its arguments, and if any of the arguments is symbolic, then the function will not be applied, but an expression will be formed containing a call to the function with its to-be-instantiated arguments.

Both the bottom level and the top level reasoning facilities have been implemented and tested. The results show that the bottom level symbolic reasoning is not worth using in the verification vision system, while the top level symbolic reasoning is suitable for this purpose. The reasons will be discussed as follows.

6.3. The Bottom Level Symbolic Reasoning Facility

The basic concept of the bottom level implementation of symbolic reasoning is that, during compile time, the reasoning facility will

expand the reasoning rules which have been applied into calls of bottom level functions and basic operations, and do as much evaluation as it can. If the positions of all the features in the relations given to the reasoning facility are instantiated then the reasoning facility will reason about the relations exactly as the cycle finder does and produce an ordinary relation. If one or more features are symbolic then the reasoning facility will instead produce a list of symbolic expressions which contain only the bottom level functions (including basic operations such as addition and subtraction) and symbolic arguments, and may be partially evaluated. In this implementation of a reasoning facility only bottom level functions have the capability of dealing with symbolic arguments.

6.3.1. The Implementation of the Bottom Level Reasoning Facility

The bottom level symbolic reasoning facility is implemented, like RAPT, in Wonder-POP [RAE81], which is a Dec-10 version of POP-2 [BUR77]. In this implementation top level functions and other high-level functions do not check the type of their arguments and have no capability of dealing with symbolic arguments. They are defined in the exactly the same way as ordinary functions. When a reasoning rule is applied the corresponding top level function is evaluated as usual until bottom level functions are called. It is the bottom level functions that check the types of their arguments. If none of these are symbolic then the function is evaluated and the results are returned to the higher-level function which called the bottom level one. If one or more arguments of a bottom level function are symbolic then the evaluation cannot be carried out and so a list containing a symbolic expression (the function

call and its arguments) is returned. The following examples show the basic performance of the bottom level symbolic operation. The items following the "**" are outputs of the functions, and "=>" is a Wonder-POP print command.

```
vars a b c d; 5->a; 6->b;
  comment declares variables a b c d and assigns values
  to a and b whilst c and d remain undefined;

a-b=>          ** -1
a+b-c=>        ** [ 11 - c ]

function fa e f; sqrt(abs(e*f))-f; end;
comment defines a function fa with two arguments and one output;

fa(b,d)=>      ** [sqrt ( abs ( 6 * d ) ) - d ]
```

The bottom level symbolic reasoning facility has been implemented in the existing RAPT system by re-assigning the Wonder-POP error trapping facility. When a bottom level function tries to manipulate symbolic arguments a Wonder-POP error is detected because of the mismatch of the type of operands, and the error trapping facility is called. The re-assigned error trapping facility uses the name of the bottom level function in which the error happens to construct a list which is textually a call of the function applied to its arguments. Then the list is simplified by a set of simplification rules. These simplification rules detect and simplify some expressions (for example, additions and subtractions with an operand zero, multiplications with an operand zero or one, subtractions with two operands having the same superficial

structure). The simplification rules are rather basic. They do not rearrange a polynomial into a canonical form, and they do not extract common factors. These operations would be time consuming and would not be of much use in the polynomials resulting from the use of symbolic reasoning in vision. The fact that all that is needed to implement the symbolic operation is the re-assignment of the Wonder-POP system error trapping facility means that the capability of handling symbolic arguments can be added to the existing RAPT system with very little alteration. The only modification needed is replacing some use of the syntax word "if". This is because in Wonder-POP if the value of a condition is "1" then it is considered as true, otherwise it is considered as false. There is therefore no data type checking for the evaluated condition expressions and therefore no occasion to call the system error trapping facility. Thus, in order to invoke the error trapping facility when a symbolic expression is met as the value of a condition some other word must be employed instead of the standard syntax word "if". The result of a conditional with a symbolic condition value is a list:

[if <expression> then ... ... close]

The expression may be expanded and partially evaluated.

6.3.2. Position Representations and Conditionals

In bottom level reasoning, positions of bodies are usually represented by matrices as in ordinary RAPT except that some elements of the matrices may be symbolic expressions. Each element in the matrix can be accessed and operated on by the reasoning facility. However,

when a position is produced by a conditional clause whose condition cannot be evaluated at compile time then the representation of the position is a list with the form:

[if <expression> then <position-1> else <position-2> close]

Here the positions in the list can be either ordinary ones or symbolic ones. In this case the reasoning facility cannot access or operate on any individual element of the position matrices in the list at compile time, but must deal with the list as one inseparable entity. This means that positions are either represented by matrices or lists, and this spoils the harmony of the reasoning process.

6.3.3.Assessment of the Bottom Level Implementation

The main advantage of the bottom level implementation is that it allows the use of symbolic reasoning in all existing programs automatically via the Wonder-POP system error trapping facility. The only modification needed is for conditionals. This method therefore makes the symbolic reasoning facility implicitly available to all the existing geometric reasoning rules, and the cycle finder can be used directly to handle symbolic relations.

The second advantage is that it needs fewer geometric reasoning facilities at run time than the top level implementation. This is because all the reasoning rules and high-level functions are either evaluated or expanded at compile time to expressions which only contain

basic operations. One matrix component selector is needed in addition to basic arithmetical operation facilities. This makes the run time system very compact.

The main disadvantage of the bottom level implementation is that it needs a lot of time during both compilation and running. At compile time, the symbolic reasoning creates long expressions to represent the positions of objects. Due to the huge size of the expressions it takes a lot of time to produce them. At run time, the evaluation of these huge expressions also takes much time. The reasons for the length of the expressions will be discussed below.

6.3.3.1. The Length of the Expressions

Theoretically speaking, if the majority of the arguments can be instantiated before the reasoning takes place then a bottom level method will be able to do most of the evaluation at compile time and the results will be compact. In practice, for verification vision tasks, very little evaluation can be done at compile time and the number of uninstantiated arguments means that most reasoning rules are expanded rather than evaluated. As mentioned before, a position in RAPT is represented by a homogeneous matrix indicating the origin and three coordinate axes of the local reference frame. If a position is symbolic then it will introduce twelve uninstantiated arguments in the reasoning process. Suppose a relation created by an INVIOLATE statement is to be combined with that created by a LOOK statement. The INVIOLATE relation contains two instantiated feature positions while the LOOK relation contains an instantiated one and a symbolic one. In total there are three

instantiated positions and one symbolic one in this reasoning process and the result is a relation containing two symbolic positions represented by symbolic expressions. If this result is to be combined with a relation created by another LOOK statement then there will be three symbolic feature positions and only one instantiated one in this reason process.

There are some complex sub-expressions which would normally have been evaluated only once appearing several times in the symbolic expression, and these sub-expressions must be evaluated each time when they appear at run time. When the same expression appears in different components of a position matrix, the expression simplifier cannot do anything with it. Even when the expression appears several times in a component, it is still not easy to simplify the component further because of the form of the component which contains the common sub-expression. Other people have met similar problems in other systems in which symbolic calculation was adopted. For example, in the electronic circuit synthesis system SYN which was described by de Kleer and Sussman [KLE78], most resources, including computing time and storage space, were used in computing the greatest common divisors (gcd) of the polynomials. When SYN failed to complete a problem, it was always because just one gcd computation filled up the entire address space of the computer. In the bottom level implementation of symbolic reasoning in the vision system, the common sub-expression problem does not cause failure, since the storage space of the computer being used (DEC-10) is large enough, but a large amount of computing time is taken up.

The representation of the RAPT position is another factor which makes the symbolic expressions very long. As mentioned above, all the

three coordinate axes of a local frame of a position are represented explicitly in RAPT, and this is redundant. Since the local reference frame is a Cartesian coordinate system, only two coordinate axes are needed and the third one can be deduced by the right-hand rule. In fact, in the reasoning process only the X- and Y-axes are used to deduce the position of newly created features although the Z-axis is also represented and calculated in each step of reasoning. The representation form of the position does not cause any problems in ordinary reasoning as the instantiated position is compact in configuration. The redundancies only increase the position representation by the size of three real numbers, and need a bit more time to calculate them. This representation form is used in RAPT because it is convenient for doing matrix multiplication. In symbolic reasoning, however, the redundancies cost very much in terms of both computer time and storage space. In symbolic position representation elements of the position matrix are symbolic expressions which are usually quite long. Therefore, three extra components may increase the size of the position expression considerably. Furthermore, since the components of the Z-axis are calculated from those of the X- and Y-axes, they are usually longer than those of the X- and Y-axes, and calculating the redundant components needs more time than that for working out X- or Y-axis.

The symbolic conditional makes things worse. The results of symbolic reasoning without conditionals have the same data type as that of a real reasoning result, i.e. a position matrix, except that each component is a symbolic expression rather than a number. In subsequent symbolic reasoning operations only the relevant components will be used to form the expressions of the components of new symbolic positions. The result of a symbolic conditional, as mentioned above, is a list

containing an expanded conditional statement. When a reasoning rule wants to access a component of the position matrix resulting from a conditional, it has to deal with the whole conditional itself. The format of the components of the position matrix which is created by a conditional is like this:

subpos(2,[if <expression> then <pos1> else <pos2> close])

where SUBPOS is a component selector and 2 indicates the second component. Thus the whole conditional statement must appear in each relevant component of the new result which depends upon some components of the position represented by the conditional. This is unacceptable especially when the conditional statement is long. Unfortunately, all the conditionals in the symbolic reasoning are very long, since the condition expressions are expanded symbolic operations on symbolic feature positions. Furthermore, an expression which contains this kind of structure cannot be simplified further. Thus the expressions of the final result containing conditionals are very long and require unnecessary repetitive calculations.

The following example shows the complexity of the symbolic expression which is generated by the bottom level symbolic reasoning facility. Suppose there is one INVIOLATE statement and two LOOK statements in a vision command package. The reasoning route is:

AGPP + AGPE + AGPE -> FIX

the final results of the reasoning before merging operation are the positions of two new features of the two bodies in the relationship FIX.

They are referred to as Ea1 and Eb1 respectively. When the edge features to be verified are not perpendicular to the plane feature of the body, the expressions of Ea1 do not contain conditionals and are about 330 lines long with 80 characters in each line when displayed on a terminal. The expressions of Eb1 do contain conditionals and are more than 5,400 lines long. The evaluation of Ea1 takes about 30 seconds while the evaluation of Eb1 takes more than 15 minutes. (The CPU time included garbage collecting time and was measured by a Wonder-POP library routine). Obviously, the requirement of both the storage space for handling the results and the time for evaluating them are not acceptable in a realistic robot control system.

6.3.3.2. Ways of Alleviating the Problems

In order to shorten the resulting position expressions and reduce the time needed for evaluating them at run time, a staged bottom level implementation of symbolic reasoning can be used. The staged method is a way to deal with the sub-expression problem. In a staged bottom level implementation some variables are declared to represent certain intermediate results which are, of course, symbolic expressions. In the subsequent reasoning these variables, rather than the intermediate results themselves, are operated by the reasoning system. The final results are position expressions which contain the variables which represent corresponding intermediate expressions. Since the intermediate symbolic expressions are represented by single identifiers the final results are shortened. At run time, the intermediate results need to be evaluated only once and then assigned to corresponding variables to enable subsequent position expressions being evaluated. This method was tested with

the same example as the pure bottom level implementation which has been described above. The reasoning route

AGPP + AGPE + AGPE -> FIX

were divided into two stages:

AGPP + AGPE -> LIN

LIN + AGPE -> FIX

The intermediate results of the first rule were given new symbolic names and the new names rather than the symbolic expressions obtained from the first step of reasoning were used in the second step. At run time, each of the symbolic expressions of the first rule were evaluated once and the values assigned to the symbolic names which had been used in the second rule before the final results were evaluated.

This technique improves the performance of the method dramatically. The lengths of the expressions of the intermediate results obtained this way are 111 lines and 250 lines respectively when being displayed while those of the final results of Ea1 and Eb1 are 57 lines and 148 lines respectively. The execution requires a couple of seconds at both compile time and run time including the time used by the garbage collector. An example of the measurement of the time used at run time is 0.5 second for calculation and 9 seconds for garbage collection. However, this is still much slower than the top level method, and the size of the results are still too large for a miniature system to handle.

The other thing that could be done to shorten the length of the position expression is to change the position representation. However, this means a lot of change in the current RAPT code. This is out of the scope of the thesis research and therefore has not been done. The ability to introduce the symbolic reasoning facility into other programs via the system error trapping facility causes another problem: system-dependence. The method implemented this way only runs under Wonder-POP. It can not even run under POP-2 on UNIX since there is no suitable error trapping facility available in that system. The programming needed in order to re-assign the error trapping facility is a bit complex because many functions of the system error trapping have to be re-defined in order to produce symbolic expressions and simplify them.

6.4. The Top Level Symbolic Reasoning Facility

The principal concept of the top level implementation of symbolic reasoning is that when the reasoning facility reasons among symbolic relations it only does this at a meta level without any expansion of the reasoning rules or partial evaluation at compile time. It only indicates what the reasoning route is, which top level functions will be applied and what type the resulting relation of each reasoning step is. The positions of the newly created symbolic features are represented in terms of top level functions rather than bottom level ones. When reasoning among instantiated relations the top level reasoning facility behaves exactly the same as the ordinary cycle finder.

6.4.1. The Implementation of the Top Level Reasoning Facility

In the top level implementation of the symbolic reasoning facility only the top level functions differ from those in the ordinary cycle finder while the other functions and operations are exactly the same as their counterparts in the cycle finder. When a top level function is applied by a reasoning rule it checks the data types of its arguments. If all the arguments are of the correct data type then the function is evaluated as a normal one. If any of its arguments are of incorrect data type then a list containing the name of the function with its arguments is produced as its result. In this list if an argument of a function is either a symbolic expression or a non-symbolic data structure then the argument is represented in the list by this expression or data structure. If an argument is an identifier which has been instantiated by either a symbolic expression or a non-symbolic data structure then this argument is represented by the content of the identifier. If an argument is an uninstantiated identifier then it is represented by the name of the identifier, and the identifier needs to be instantiated at run time.

Since the types of all the relations including symbolic relations are instantiated and no top level conditionals appear during the symbolic reasoning, the types of newly created features and relations, and therefore the reasoning route, can be determined at compile time. As no reasoning rule expansion occurs, there are no lower level symbolic conditionals appearing in the symbolic reasoning results. Hence, the results of the top level symbolic reasoning contain top level functions only.

6.4.2. Assessment of the Top Level Implementation

The main advantages of the top level implementation are that:

(1) this method works faster at compile time than the bottom level one;

(2) the results of this method are much more compact;

(3) the speed of evaluating them is much higher than those produced by the bottom level method.

When tested with the same example used for testing the bottom level method (Section 6.2), both the symbolic reasoning at compile time and evaluating the symbolic results at run time needed less than a tenth of a second. The symbolic position expression of the body being verified (this includes the merging operation) is eight lines long when displayed on a terminal. Both the time and storage space requisites are realistic for a practical robot control system. The top level symbolic reasoning facility has therefore been employed in the verification vision system.

Compared to the bottom level method, the top level one has two disadvantages. (1) The first is that the method cannot introduce the symbolic reasoning capability into existing geometric reasoning rules implicitly. In order to do this, some explicit statements have to be added into each top level function of the reasoning rules. The symbolic reasoning system therefore cannot apply the existing top level functions of the cycle finder, but instead, it has to use ones which are defined specifically. (2) The second disadvantage is that the method needs more computational facilities at run time than those required by the bottom level method. Generally speaking, in a symbolic reasoning system it can be decided at compile time what computational facilities will be needed at run time in order to evaluate the symbolic results, and the

definitions of the facilities must be sent to the run time system. Since no reasoning rule expansion is done at compile time, in the top level implementation all the facilities that are called by the top level functions have to be available at run time. This makes the corresponding run time system larger than that needed by the bottom level method.

6.5. The Reasoning Rules for Symbolic Reasoning

Since the decision has been made to use top level symbolic reasoning, it is necessary to provide a new set of reasoning rules that can be used with symbolic relations. Because of the way in which vision is used in verification in the system described here, there are only a limited number of relations which can take symbolic form, and only a limited number of ways that these relations will need to be paired with other relations. Therefore, there is only a limited number of reasoning rules which have to be replaced. These reasoning rules are discussed below.

The size of the relation cycle in vision verification is always two, and so it is only combination rules that have to be rewritten, not creation rules (Section 3.3.3). As discussed in Section 5.2, the look statement creates an AGAINST relation between an edge feature of the body to be verified and a plane feature of the camera. Since the position of the camera must be known when it looks for the edge to be verified, the plane feature of the camera can be considered as a feature of the world. (In the cycle finder, the camera must be "merged" into the world in this case since it is FIXED to the world directly or indirectly.) Thus the AGPC relation holds between the world and the body

to be verified. The INVIOLATE statement creates an AGAINST relation between plane features of a reference body and the body to be verified. Since the reference body must be either the world or a body which has been FIXED to the world (Section 5.3) this relation also holds between the world and the body to be verified.

It is a RAPT convention that when two relationships are to be combined together, the relations are described in such a way that the first feature mentioned in each relation belongs to one body, and the second feature mentioned in each relation belongs to the second body. In the discussion which follows, these are referred to as body a and body b respectively. Thus, if an AGPP and an AGPC are to be combined together then body a has two face features while body b has a face feature and a cylindrical feature. In the symbolic reasoning body a represents the world while body b stands for the body to be verified. Features of the body to be verified will be referred to as body features while those of the reference body and the symbolic feature of the camera are referred to as world features when convenient.

The only relation types that appear in the symbolic reasoning caused by vision verification are AGPP, AGPC, LIN and FIX. The AGPP and AGPC are input relations created by vision commands while the LIN and the FIX are internal ones created by the reasoning system, and the FIX is the final result that the reasoning system infers towards. The ordinary reasoning rule table for instantiated relations which is relevant to the symbolic reasoning is shown in Table 6.1.

No.	R1	R2	RR	condition
1	AGPP	AGPP	AGPP	x_a_parallel
2			LIN	general
3	AGPP	LIN	FIX	general
4			LIN	x_a_perpendicular
5	AGPP	AGPC	AGPP	x_a_parallel
6			ROTYLIN	x_b_parallel
7			LIN(2)	x_b_perpendicular
8			LIN(2)	general
9	LIN	AGPC	LIN	x_a_perpendicular
10			FIX	x_a_parallel
11			FIX	general

Note: (1) R1 and R2 indicate two relationships which constitute a relational cycle.
(2) RR indicates the resulting relationship.
(3) x_a_parallel means X-axes of the two features of body a are parallel to each other, etc.
(4) x_a_perpendicular means X-axes of the two features of body a are perpendicular to each other, etc.
(5) "(2)" means the result is ambiguous.

Table 6.1. Combination Rules Relevant to Symbolic Reasoning

No.	R1	R2	RR	condition
1	AGPP	AGPP	AGPP*	x_a_parallel
2			LIN *	general
3	AGPP	AGPE	LIN	x_b_perpendicular
4			LIN	general
5	LIN	AGPE	FIX	x_a_parallel

Note: "*" indicates the reasoning is not symbolic.
AGPE is a restricted case of AGPC

Table 6.2. Symbolic Reasoning Rule Table

6.5.1. Combining an AGPP and an AGPC

In the existing cycle finder the rules for combining the relation pair AGPP and AGPC (lines 5-8 in Table 6.1) have not been implemented because in most cases the result is ambiguous. The general result of the combination of an AGPP and an AGPC is a LIN, which holds between two newly created edge features. The new edge of body a is determined by the two faces of body a while the new edge of body b is determined by all the four features in the AGPP and AGPC. A LIN relationship indicates that the X-axes of the edge features are collinear and in the same direction and that the Y-axes are parallel to each other. However, since the combination of an AGPP and an AGPC does not contain enough information about how to choose the directions of the edge features in the LIN, there are two LIN relationships which satisfy the constraints. The two relationships differ from each other by a rotation of 180 degrees about an axis which is perpendicular to the face feature of body b (Fig. 6.1). The existing cycle finding system has no way of choosing between the two possible situations.

In symbolic reasoning the nature of the verification task provides the necessary constraint for choosing the unique correct answer. When a body is to be verified the discrepancy between its nominal position and actual one is assumed not to be too big. In practice it is very unlikely that the actual position of the body to be verified differs from the nominal one by more than 90 degrees of rotation. This characteristic presents a basis for the symbolic reasoning system to eliminate the ambiguity in the resultant LIN relation. The symbolic reasoning system is able to select the correct LIN relationship in the following way. The reasoning system assigns the direction of the X-axes of

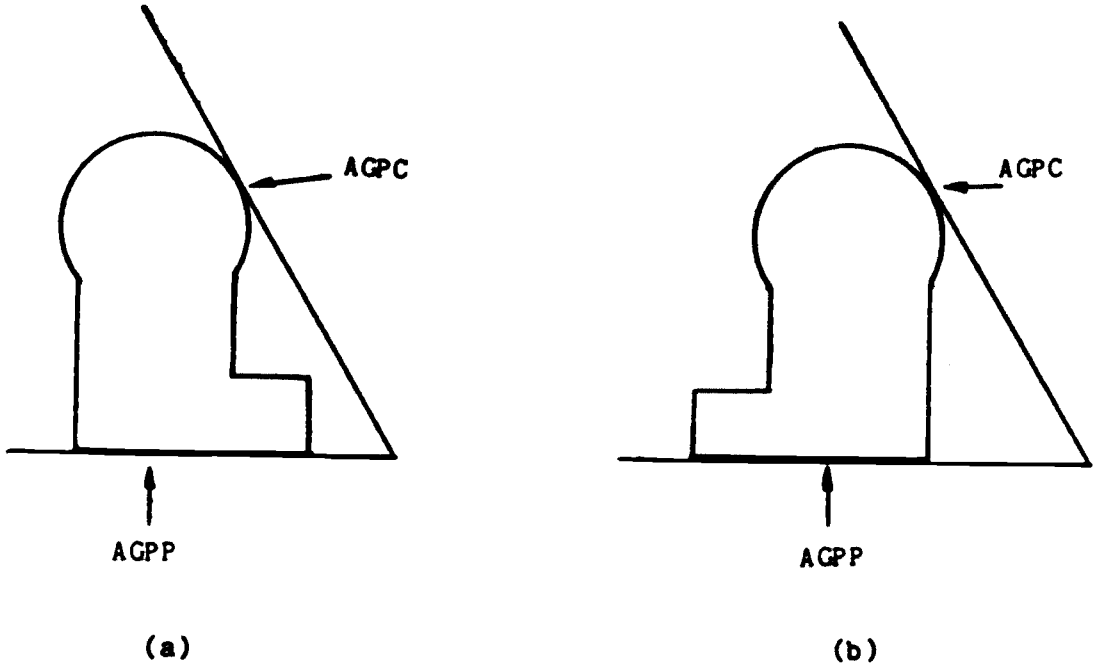


Fig. 6.1. Two situations which both satisfy AGPP+AGPC

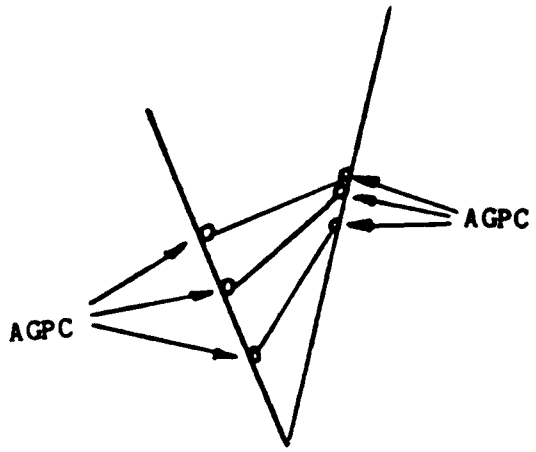


Fig. 6.2. Three possible situations of the infinite number which satisfy AGPP+AGPC+AGPC when the X-axes of edges are all parallel to that of the face feature of the AGPP

the new edge features arbitrarily at first provided they are collinear, and then tests the angle between the nominal body orientation of the body to be verified and the verified orientation under this assignment. (Notice, since a LIN has only one translational degree of freedom, it indicates the orientation of a body). If the angle is less than 90 degrees than the directions which have been assigned are considered correct. Otherwise the direction of the X-axis of the new edge feature of body b is reversed.

As a result of this reasoning process, the position of the new edge feature of body a in the LIN relation must be symbolic since it is determined by the two face features of body a in the AGPP and AGPC relations and the position of the face feature in the AGPC relation will be determined by vision data. The position of the new edge feature of body b is also symbolic, since it is determined by all four feature positions in the AGPP and AGPC relations. Therefore, the angle between the nominal orientation and the verified orientation of body b cannot be calculated until run time, and this means that the position of the new edge feature of body b (the body to be verified) will be expressed by a symbolic conditional. However, changing the direction of the edge feature does not change the nature of the resulting relation, and so the conditional is not a top level one. Therefore, in the top level symbolic reasoning this conditional is not expressed explicitly at compile time. (In the bottom level reasoning, if it were used, the symbolic expressions would be significantly elongated.)

When the X-axis of the cylindrical feature in an AGPC relation is parallel to that of the face feature of body b in an AGPP relation, combining the AGPP and the AGPC will produce a special relation ROTYLIN

(line 6 in Table 6.1). This relationship holds between two new edge features, and means that the body with the cylindrical feature (body b) can move along a line which is determined by the two face features of body a, and can rotate about the axis of the cylinder. This is a difficult relation type for the geometrical reasoning system to deal with since if it is to be combined with other relations in the subsequent reasoning a large number of special conditions must be tested and the results are usually ambiguous or cannot be simply expressed. For instance, when a ROTYLIN is to be combined with an AGPC relation, eight conditions must be examined. Among the nine possible results, five are ambiguous and difficult to use in subsequent reasoning (see Appendix II). For this reason ROTYLIN has not been implemented in the cycle finder. For the same reason it has not been adopted in the symbolic reasoning system either. In combining an AGPP with an AGPC, if the X-axis of the edge feature is parallel to that of the face feature of body b in the AGPP relation then the AGPC is sidestepped by the cycle finder which attempts to find a more suitable pair of relations.

Discarding the relation type ROTYLIN does not cripple the capability of the symbolic reasoning system. It is shown in Appendix II that even if the ROTYLIN were to be implemented it would cause some top level conditionals and increase the power of the symbolic reasoning system very little. In the case of using one AGPP and two AGPC relations to reason about the position of the body to be verified, if the X-axes of the two edge features to be verified were all parallel to that of the face feature of the body then the position of the body would not be completely determined. Instead, the body position would be a complex function of the positions of the features given in the AGPP and AGPC relations. This is illustrated in Fig. 6.2. The reasoning system would need at least

another relation between the body and the world to fix the body position. In order to determine the body position completely by an AGPP and two AGPC relations, there must be at least one edge feature whose X-axis is not parallel to that of the face feature of the body to be verified. Thus the relation which contains this edge feature can be combined with the AGPP relation first to create a LIN relation, and then the LIN relation can be combined with the other AGPC relation the X-axis of the edge feature of which may be parallel to that of the face feature of the body.

In combining an AGPP and an AGPC in the cycle finder, if the X-axes of the two face features of body a are parallel to each other then the AGPC is redundant and does not provide any new constraint on the relation between body a and body b which is fully expressed by the original AGPP relationship (line 5 in Table 6.1). Similar things may also happen in the symbolic reasoning. In vision verification the symbolic face feature generated by the LOOK command may be parallel to the reference face feature used in the inviolate AGPP relationship if the X-axis of the edge feature of the body to be verified is perpendicular to that of the face feature in the inviolate AGPP and the camera is at a certain position (Fig 6.3). When this happens it causes a symbolic top level conditional since the resultant relation is of different type from the general result and examining the condition can only be calculated at run time when the symbolic feature is instantiated. It should be avoided in the symbolic reasoning; at present the user has to avoid the possibility of this happening, in the future (see Chapter 10) the system will select the feature to be verified in order to avoid this.

Fortunately, in most circumstances this type of top level

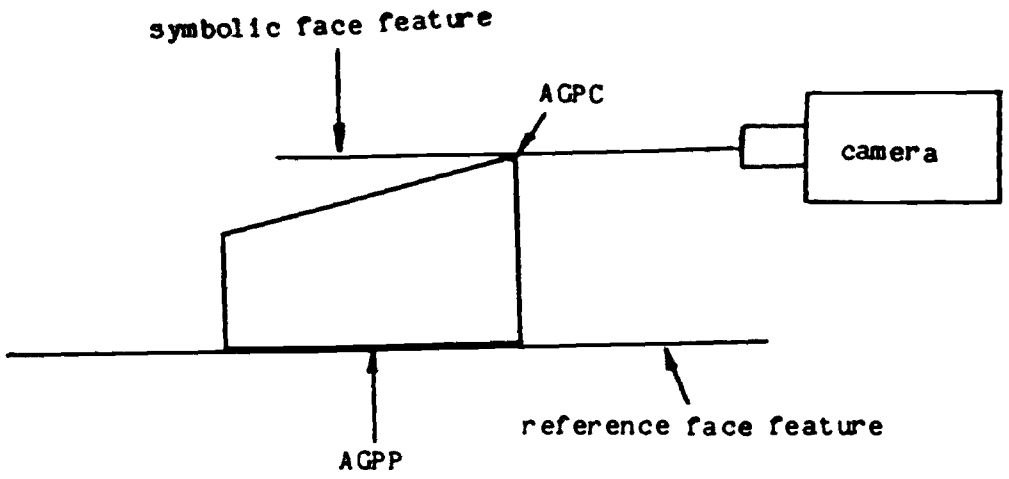
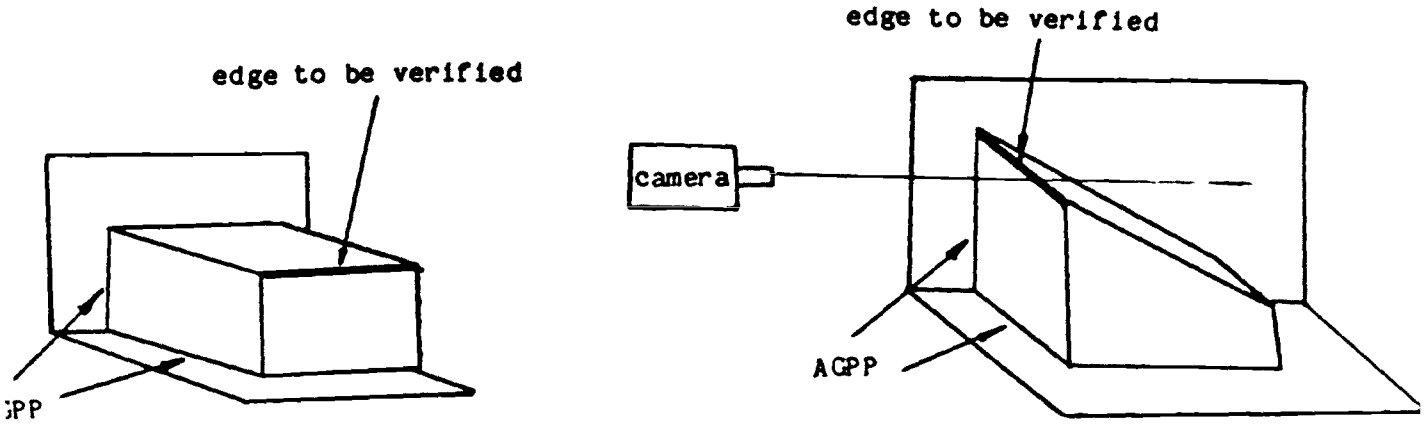


Fig. 6.3. A situation in which the symbolic face feature is parallel to the reference face feature



(a) (b)

Fig. 6.4. Two situations in which LIN+AGPC produces a LIN

conditional can be avoided in verification vision tasks. In an assembly station the assembly task is usually carried out on a work table, and the inviolate relation which is most likely to be used is the one that indicates a face feature of the body to be verified is against the top of the work table. The camera is usually installed so that it is above the work table and looking down on it. The symbolic face feature which is determined by the positions of the camera and the end points of the edge to be verified therefore cannot impossibly be parallel to the top of the work table. For the few cases in which the symbolic face might be parallel to the reference face in the AGPP, the user is asked to select the position of the camera and the edge to be verified carefully. Lines 5-8 in Table 6.1 can therefore be replaced by just two symbolic reasoning rules (lines 3 and 4 in Table 6.2).

6.5.2. Combining a LIN and an AGPC

The rules for combining the relation pair LIN and AGPC have been implemented in the cycle finder (line 9-11 in Table 6.1). In the symbolic reasoning system that part of the rule which leads to a LIN has not been adopted since it would cause a top level conditional. The parts of the rule which produce a FIX have been implemented with some modification so that they are more suitable for dealing with the AGPC relations.

When the X-axis of the face feature of body a in the AGPC is perpendicular to that of the edge feature of body a in the input LIN relation, the AGPC is redundant and the cycle finding system combines the AGPC and LIN to produce the original LIN as result. In the symbolic reasoning

caused by vision verification this condition may happen in various circumstances. Only a very limited number of the circumstances can be anticipated or partially anticipated without knowing the relevant vision data. The input LIN relation may have been derived in either of two ways in the symbolic reasoning. It may result either from the combination of two AGPP relations or from the combination of an AGPP and an AGPC relation. When the LIN is derived from two AGPP relations, the condition which causes the combination of the LIN and an AGPC to be a LIN arises if the X-axis of the edge to be verified is perpendicular to faces of the body to be verified in both AGPP relations (Fig. 6.4 (a)), or if the X-axis of the edge is perpendicular to the body face in one AGPP relation and the camera is at a certain position (e.g. Fig. 6.4 (b)). When the LIN results from combining an AGPP and an AGPC, the only partially predictable circumstance in which this condition arises is that the X-axes of the two edges to be verified are parallel to each other and perpendicular to the body face in the AGPP relation. If these X-axes are also perpendicular to that of the camera then the X-axes of the world edge feature in the derived LIN and the world plane feature in the AGPC are perpendicular to each other, and so the AGPC cannot provide any useful information, and so the resulting relation is the same LIN. It is easy to see from the discussion above that a top level conditional will be produced in most cases when the input is tested for perpendicularity of the X-axes of symbolic world features involved in the LIN, AGPC combination. However, since this case actually arises only when the user has attempted to specify the use of the camera in an unsuitable way, at present the symbolic reasoning system assumes that the special case will never occur and so does not insert any test (and hence does not insert a top level conditional). It therefore has to rely on the user's common sense. In the future, the system will select the edges

and cameras automatically so that the user can be released from the burden of deciding what to look for. This will be discussed in detail in Chapter 10. At the moment, if the user fails to choose correct edge features and cameras then the symbolic reasoning system cannot completely determine the position of the body to be verified. It reports the situation to the user at compile time and then declares that it will ignore the relevant run time vision operation commands if the user does not take any proper action.

In the symbolic reasoning, checking for the special condition in which the X-axes of the world features in the LIN and AGPC are parallel to each other also needs vision data which can only be obtained at run time. However, since this condition does not change the type of the resulting relationship, it does not cause a top level conditional. The reasoning rule for this special case can be merged with that for the general case. Lines 9-11 of Table 6.1 can therefore be replaced by line 5 of Table 6.2.

6.5.3. Combining Other Relation Pairs

In the symbolic reasoning system the AGPP relation is always created by the INVIOLATE statement and all relevant positions in it are instantiated. Thus reasoning among AGPP relations is not symbolic, and the reasoning rules are exactly the same as those used in the cycle finder (lines 1,2 in Table 6.1).

When there are two INVIOLATE statements in a vision command package, the two AGPP relations are always combined first by the reasoning system

to produce either a LIN, in the general case, or an AGPP, in the redundant case (lines 1,2 in Table 6.1). Thus, there is no possibility that an AGPP will be combined with a symbolic LIN, and therefore there are no rules needed in the symbolic reasoning system to deal with this.

The combination rules which are used by the symbolic reasoning system are shown in Table 6.2. It is a subset of Table 6.1. As discussed in Section 5.2, the AGPC relationships created by LOOK statements are of special case. In order to simplify the symbolic reasoning, the special AGPCs are denoted as AGPEs.

6.6. The Control of the Symbolic Reasoning

The control strategy used in the symbolic reasoning system is different from that used in the cycle finder in that the symbolic reasoning system discriminates relation types and reasons among AGPP relations first. This is because, unlike the cycle finder, different types of relations in the symbolic reasoning system come from different sources and have distinct reliabilities. In order to obtain more reliable results, the reasoning system should take advantage of the more dependable relations when the specification is overconstrained.

All the types of relations that occur in the cycle finder are either specified by the user's program or deduced from it. The cycle finder searches through the network for relation pairs and tries to find a suitable entry in the reasoning rule table. Every relation type is considered equal in reliability and importance to others and no relaaton types are given priority. The order of using relation pairs in the

reasoning process is arbitrary and is partially determined by the intermediate results of the reasoning. If the specification is overconstrained in the relation network then the user cannot predict which relation will be actually used and which will be merely tested for consistency by the reasoning system.

In the symbolic reasoning system, however, things are different. There are only two types of input relations in the reasoning process: the AGPP relation and the AGPE relation, but their reliability is distinct because they come from different sources. The AGPP relation created by the INVIOLATE statement describes a relation which must hold in the real world in a situation of the assembly task. It can therefore be considered to be reliable. In contrast to the AGPP relation, the AGPE relation arises from the LOOK statement, and its parameters will be determined at run time by the vision data. Since the accuracy of the vision facility is limited, the AGPE relation is less reliable than AGPP.

In describing a verification vision task, the user is allowed to make more statements than necessary. For instance, the user may make one INVIOLATE statement and three LOOK statements in a vision command package. This gives the reasoning system the opportunity to select suitable LOOK statements from the given three in order to avoid top level conditionals or some undesired combinations such as ROTYLIN. It therefore reduces the burden on the user of selecting edges to be verified. The only case that the symbolic reasoning system can deal with at present is avoiding the ROTYLIN that is produced when combining an AGPP and an AGPE in the special case where the X-axes of the features of the body to be verified are parallel to each other. Suppose, in the above

example, the edge in the first AGPE relation considered is parallel to the body face in the inviolate AGPP relation, then the reasoning system can detect this and can discard that AGPE relation temporarily and try the next. If that failed then it can try the third. In the future, the capability of the symbolic reasoning system will be strengthened so that other cases can be dealt with and the user can give it more candidate edges to be verified and rely on the system selecting proper ones in order to avoid other kinds of the top level conditional in the reasoning process.

When two INVIOLATE statements and more than necessary LOOK statements are made in a vision command package, it is desirable to make use of both the AGPP relations specified by the given INVIOLATE statements and only one of the AGPE relations since AGPP relations are more reliable. To this end, the reasoning process is divided into two stages. In the first stage, only the AGPP relations are fed to the reasoning system. Thus, if there are any AGPP relation pairs then they are combined together. In the second stage, the AGPE relations are appended to the results obtained from the first stage and used.

6.7. Merging

After the symbolic reasoning, the result for each vision command package is checked to see whether the position of the body instance to be verified has been "fixed" or not. The "fixed" body instances are then "merged" (see Chapter 3) into the world. Since the FIX relation deduced from the symbolic reasoning always holds between a feature of the body to be verified and a feature of the world, the merging process

is straightforward. Suppose that f1 is a feature of the world whose position is p1 and f2 is a feature of the body to be verified whose position is p2, and furthermore suppose that a FIX relation holds between f1 and f2. The relationship FIX means that the coordinate systems of the two features are coincident under the world coordinate system. Thus, the following equation stands:

$$f1 ** p1 = f2 ** p2 \qquad (6.1)$$

Since p1 here is the position of the world it is an identity matrix. The position of the body to be verified, p2, can then be represented by the equation:

$$p2 = f2^{-1} ** f1 \qquad (6.2)$$

Of course, the merging operation is symbolic and the resulting symbolic position expression will be evaluated at run time. This process does not involve any extra rules. The modification of the functions of the merging process of the current cycle finder merely enables them to handle symbolic arguments.

The reasoning system will report to the user about the body instances which cannot be "fixed" by the symbolic reasoning. The user then has the opportunity to take some action to deal with this, such as re-specifying the vision tasks. If he does not do any thing then the numbers of the situations in which unsuccessful reasoning takes place are recorded and will be passed to the run time system so that the corresponding vision tasks will be considered invalid, and the relevant camera operating commands will be ignored at run time.

6.8. Summary

In this chapter two approaches of implementing symbolic reasoning capability, the bottom level symbolic reasoning, and the top level symbolic reasoning, are discussed and compared. Because the top level method is overwhelmingly better than the bottom level one, it is used in the verification vision system. In order to handle the symbolic reasoning caused by vision verification, some reasoning rules are re-defined and a new control strategy of reasoning is adopted.

Chapter 7. A Framework for Handling Vision Information

At run time, cameras operate under the control of the commands created by LOOK statements, and symbolic position expressions which result from the symbolic reasoning system are evaluated one by one in corresponding situations after the relevant vision data has been acquired. These evaluated position expressions define the actual positions of the bodies which have been verified by the vision system. Usually, the verified position is different from the corresponding nominal one and the robot system needs to utilize the information in order to update its knowledge about the environment and modify the planned actions of the robot.

The modification of the planned actions is not explicitly described in the user's program. Instead, it is done implicitly by a framework which handles the vision information. This is partially because the current RAPT has no flow control statements available. The more important reason is that the implicit specification of the modification is more natural and convenient to the user. When the user specifies a vision verification task, he shows that he believes that the actual working environment of the robot will in fact differ from that described in the program. He wants to examine the actual environment and adapt the planned actions to possible changes in the environment. If the system can decide how to make use of the vision information automatically in order to fulfill the planned task then the user does not need to worry about how to do this by himself. In fact, it is rather complex to work out how to make use of the vision information properly and completely as the relations between body instances are complex and implicit. Thus, a carefully designed framework can do better than less

experienced users.

7.1. Basic Requirements for the Framework

There are three basic requirements that the framework must meet.

1) Firstly, the framework must be able to use the vision information to update the knowledge of the robot system so that it can find out not only what modification should be done on the planned action directly related to the verified body instance, but also what modification should be made to subsequent actions. Modifications will be necessary for those actions which depend upon other body instances in the following situations which are affected indirectly by the vision information.
2) Secondly, the framework must work efficiently at run time.
3) Thirdly, the framework should be relatively independent of the current RAPT system.

These will be discussed in detail below.

7.1.1. Making Full Use of Vision Information

It is commonly the case that vision data will verify not only the position of the specified body at the current situation (body instance), but also some other body instances whose positions are relevant to or deduced from this body instance. From a given input program the RAPT system produces the expected positions of bodies of the entire sequence of situations in an assembly task. The use of vision data to verify a position will indicate a discrepancy between the expected position and

an actual position and it will be necessary to modify the expectation about subsequent positions in the light of this information. For example, suppose a robot moved a block to a specified position and then moved away a fixed amount waiting for the vision system to operate. If the verified position shows that the block is not exactly at the specified position, then the system should know that the robot hand now is also not at the position where the system supposed it to have been (Fig. 7.1). As another example suppose that the robot is asked to move another body to a place in such a way as to satisfy a set of spatial relationships between this body and the block mentioned above, then the robot system should know that the new position of the body must be different from that expected before the verified position was known (Fig. 7.2); the planned action therefore has to be modified.

The examples given above are simple ones. In RAPT, the relations between bodies are far more complex than those in these examples. The effects of a verified position are not only implied by action statements but also implied by TIE and subassembly statements. The framework needs to use the verified position to update the system's idea of the positions of all relevant bodies in the world.

7.1.2. Efficiency in Time and Space in Run Time

The run time efficiency is an important requirement to a practical robot system. It is desirable that a run time system works fast so that visual information can be processed and utilized in real time. It is also desired that a run time system is compact so that it can be accommodated in a small computer which controls the robot directly. Thus the

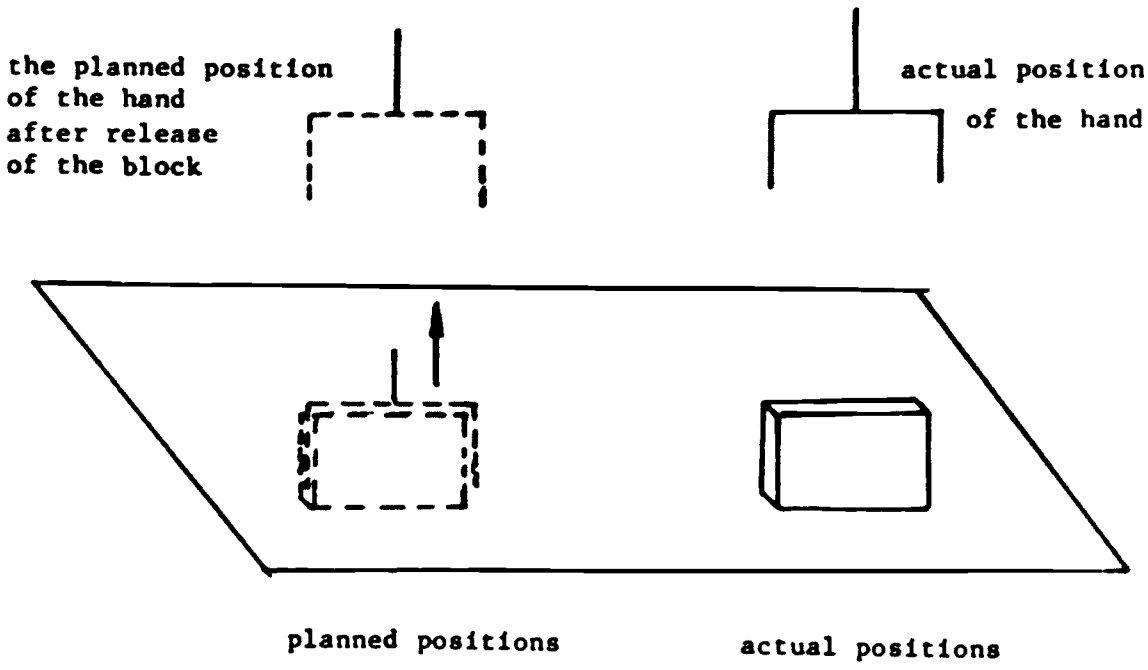


Fig. 7.1 The influence between body positions (I)

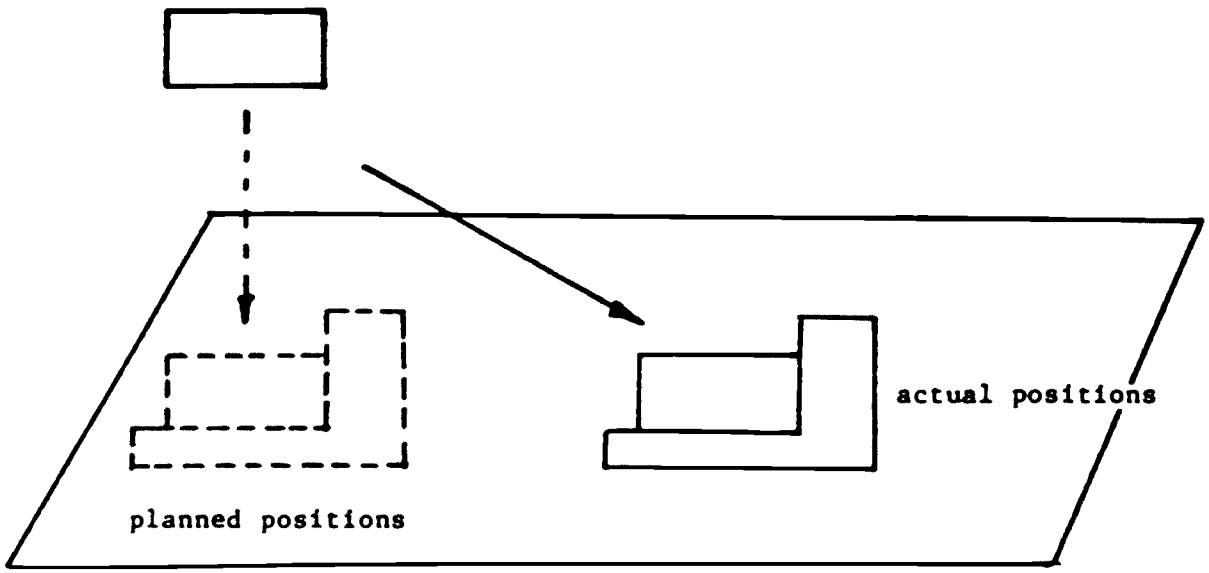


Fig. 7.2 The influence between body positions (II)

framework must do as much work as possible at compile time, especially the work which needs powerful computing facilities, so that the run time calculation is simple and straightforward.

7.1.3. The Independence of the Framework

Since RAPT is a language under development, it is necessary to keep the verification vision system relatively independent of other parts of the RAPT system. This requires two things. From the point of view of the system realization, the implementation of the framework should not need significant changes in the RAPT system. From the user's point of view, the explanation of vision information should not influence the syntax and semantics of the other RAPT statements. If the user wants to use the vision system to verify positions of some body instances then all he needs to do is simply to insert some suitable vision command packages into the proper places in his ordinary RAPT program without any changes or modifications in the program which has been proven to be correct before vision commands are introduced.

7.2. Frameworks for Using Vision Information

There are several ways to establish the relations between a verified position and others. Four ways will be discussed in the following.

7.2.1. Symbolic Reasoning Method

The relations can be set up by a symbolic reasoning system. Each position which is relevant to a verified position is considered by the symbolic reasoning system and expressed as a symbolic expression which contains the verified position. Thus the dependence of other positions on the verified position is worked out symbolically at compile time. After evaluation of the verified position, the symbolic position expressions can be evaluated. This method brings about some serious problems. When a position is represented by a symbolic expression, the principal part of the actual reasoning will be done by the evaluation at run time. If a verified position is related to several other positions then the expression of the verified position, or part of the expression, will appear in a large number of places in the run time code, and the evaluation of each of them will take much time. This will slow down the speed of the run time system. Also, when the effect of a verified position propagates through body instances, the expressions of positions of the body instances become longer and longer and the evaluation of them needs more and more time, since each position expression contains the expression of the position of the body instance from which the effect of the verified position comes. As many positions are expressed symbolically, the requirements on the storage space at both compile time and run time will be enlarged. Furthermore, this method needs a very powerful symbolic reasoning system. The proposed reasoning must be able to do everything symbolically that the cycle finder can do. It must be able to reason among not only 2-cycles but also large sized relation cycles. Thus it must be able to use not only the combination rules but also the creation rules. It must also be able to deal with a number of types of symbolic top level conditionals. It will be seen that because of the

disadvantages in time and space efficiency and because of the very great demands for a powerful run time inference system, this method is not viable. It was therefore felt not worth serious development.

7.2.2. Run Time Reasoning Method

In this method the major part of the work will be done at run time. The difference between this method and the symbolic reasoning method is that the compile time inference system only reasons about the relationships among body instances which are not dependent upon vision information, and reasons symbolically about the positions of body instances which are verified by vision directly. Thus the compile time reasoning is faster than in the symbolic reasoning method. The positions of body instances which are dependent upon vision information will be produced at run time by geometrical reasoning after the associated vision data is available and the relevant symbolic positions are evaluated. This method needs a powerful run time system which must be as capable as the compile time system in geometrical reasoning. This means that a powerful and usually expensive computing facility must be available at run time. Even if the computing facility is available, the run time geometrical reasoning will slow down the run time processing speed dramatically, especially when the positions of a number of body instances are dependent upon vision information. Thus this method is out of the question.

7.2.3. The Use of Teach Mode

Another method is suggested by D. Corner [COR84]. This method makes use of a modified reasoning system to establish the relations between a position which needs to be determined at run time and other positions. It is designed and implemented as an expansion of the RAPT system for introducing a teach mode in which positions of some body instances can be taught at run time.

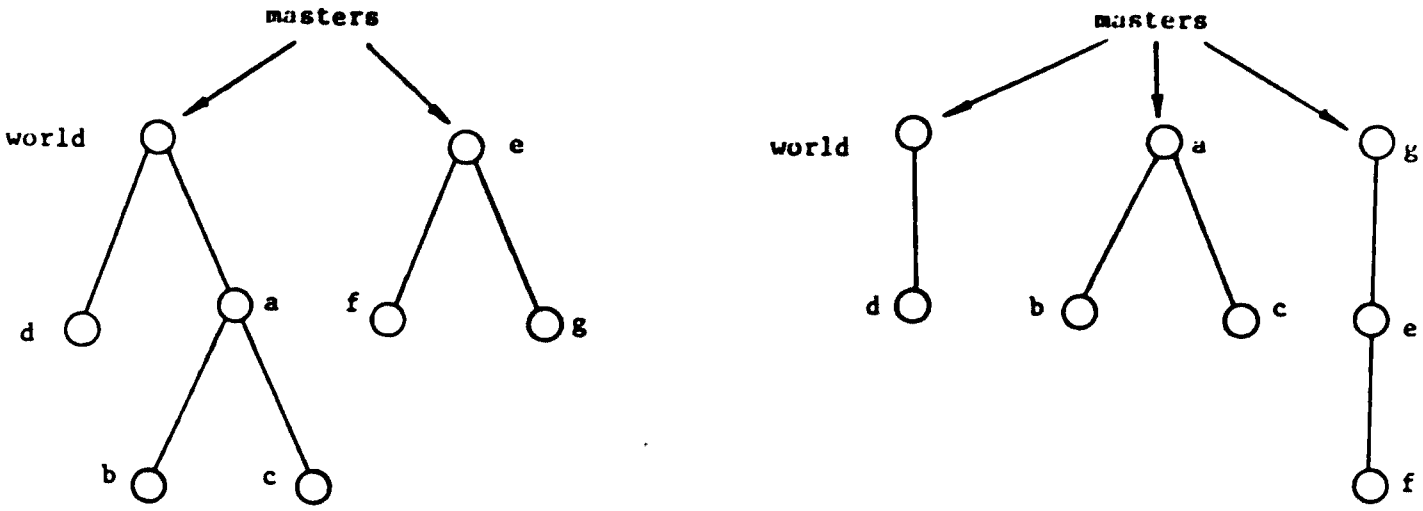
In order to understand Corner's system it is necessary to understand how the merging of two body instances which have a FIX relationship is carried out. One body instance (the child) is merged into another (the parent), and the position of the child is expressed relative to the position of the parent. According to the merging rules applied in the cycle finder, the body number of a parent is always smaller than or equal to those of body instances merged into it. A body instance which is a parent can itself be merged into another body instance. Therefore, after the merging, body instances are grouped into trees, and the most superior parent in each tree is the root node and is referred to as the master of other body instances in that tree. (Fig. 7.3 (a)). The positions of body instances which have been merged can be represented in terms of the coordinate system of the master body instance by repeated position multiplications. Therefore, if a body instance is a node of the tree whose root is the world (and since the body number of the world is 1 it must always be the root node of a tree) then its position in the world coordinate system can be determined.

In Corner's method a TEACH command creates a new situation in the assembly task. In the relation network there will be no direct

relations holding between the body instance whose position needs to be taught and the body instance for the same body in the previous situation. The reasoning process is divided into two phases. In the first phase, the reasoning system works in the same way as the cycle finder except that all the body instances whose positions will be taught at run time are marked, and these marked body instances are never merged into other body instances. Instead, other body instances which have a FIX relationship with them are merged into them. Thus the "taught" body instances are root nodes of individual trees. For example, Fig. 7.3 (b) shows the relationship network produced by a program similar to that which produces the relationship network shown in Fig. 7.3 (a). The only difference between these two programs is that in the program associated with Fig. 7.3 (b) the positions of body instances a and g is to be taught at run time. In the second phase, a symbolic FIX relation is inserted between the taught body instance and the body instance for the same body in the previous situation. At run time, when the taught position is known the symbolic FIX relation can be instantiated and all the positions in the cluster can be computed and expressed in terms of the world coordinate system.

The advantages of this method are that it makes full use of the capability of the current RAPT reasoning system to establish the relations between positions of body instances and information obtained at run time. If there are any symbolic expressions to be evaluated at run time then these need to be done only once. However, there would be several disadvantages in this method when it was used for dealing with vision information. Three main ones are listed as follows.

1) Firstly, the implementation of this method would need a significant modification on the current reasoning system and this is



(a) in the current cycle finder

(b) in Corner's TEACH mode

Fig. 7.3 Tree structures of body instances in inference systems

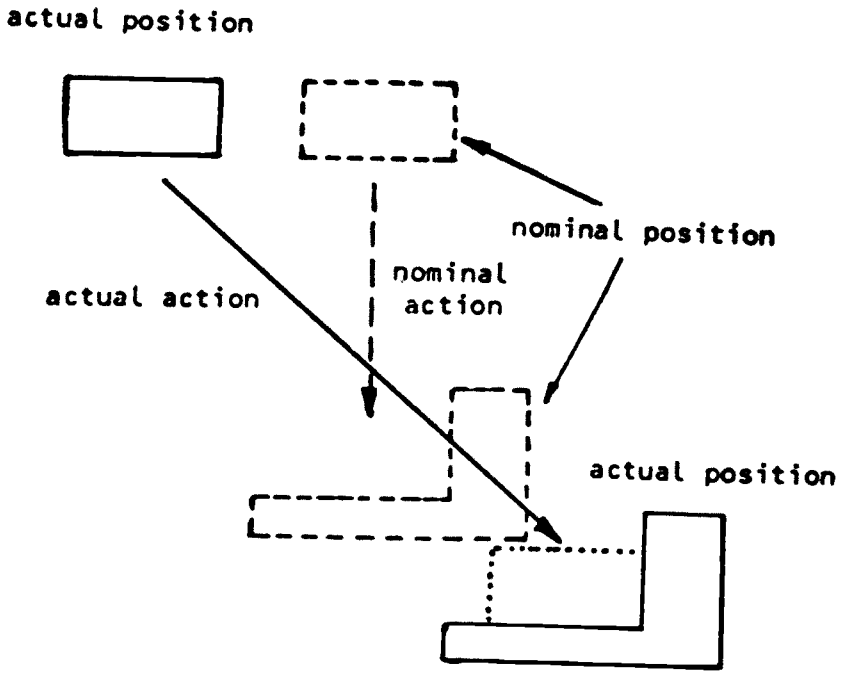


Fig. 7.4 The one step strategy

what the author wants to avoid.

2) Secondly, this method would influence the semantics of some RAPT statements in some circumstances. Some programs which are correct without vision commands would become incorrect when vision commands were added, if this method was used to handle vision information. For instance, consider the RAPT instructions to move object b1 from a known position to some distant place in relation to b2, whose position is to be verified. The following RAPT codes would be invalid since the verified position might not be identical to its nominal position so that the MOVE statement between situations i+2 and i+3 could not guarantee to bring about a specified FIX relation between b1 and b2 in situation i+3, although these codes would be a proper segment of a program without the vision command package.

```
remark now in sit i, bodies hand, b1 and b2 have been modelled;
    move/hand;
       fixed/hand, b1;  remark sit i+1, fixed is abbreviation
                                of a set of relations which can
                                fix the two bodies;
       tied/hand, b1;
    verify/b2;          remark abbreviation of a vision command
                                 package;
                        remark now in sit i+2;
    move/hand, perpto, f2 of hand, 55;
       fixed/b1, b2;    remark sit i+3;
```

7.2.4. The Method Adopted

The framework which was adopted by the verification vision system for handling vision information avoids all the disadvantages of the three methods which have been discussed above. It works efficiently at both compile time and run time, does not need to take much extra storage space, and does not change the semantics of RAPT statements. When vision commands are added into a correct RAPT program, no modification of the original program is needed. Furthermore, the implementation of the framework does not need significant changes in the current RAPT system.

The basic operational principle of the framework is that it decomposes the actual position of a body instance into two parts. One is the nominal position which results from the inference system and has no relation with the vision information. The other indicates the influence of the vision information on knowledge of the actual position of a body instance. This part is referred to as the modifying factor of the body instance and work on determining its value is done both at compile time (deciding what dependence it has on vision information) and at run time (evaluation).

The remainder of Chapter 7 is used to describe the outline of the framework and discuss its basic principles. Chapter 8 establishes and explains rules for making and simplifying the modifying factor array which is the heart of the framework. In discussions in these chapters the following two assumptions will be employed:

1. The nominal position of a body is assumed to be accurate unless

there is some evidence (e.g. vision data) to the contrary.

2. The movement of the robot arm is assumed to be accurate for each individual action.

Both these assumptions are reasonable. Firstly, if the nominal positions are not accurate then either their actual positions are immaterial, or the programmer should have included some evidence-gathering instructions in his program. Secondly, in present day robots the inaccuracies of movement are due to inaccuracies of their mechanical parts and imperfection in their feedback transducers.

7.3. Relations Between Vision Information and Body Positions

Positions of body instances can be classified into three sorts according to their relations with the vision information:

1) the body instances which are verified directly by vision commands and whose actual positions are determined by vision information.

2) the body instances which are not verified by vision commands but whose actual positions are influenced by vision information in some indirect way. Their actual positions are determined jointly by both the nominal positions and vision information.

3) the body instances whose positions have no relations with vision information at all. Their nominal positions are considered to be identical with the actual ones.

For uniformity, the actual positions of all the three sorts are decomposed into two parts so that the framework can deal with them in the same way.

7.3.1. Positions of Verified Body Instances

If the actual position of a verified body instance is to be determined completely by vision information, then it is treated in the following way. Suppose the nominal position of body b in situation i is represented by the matrix PNbi and its verified position in the same situation is PVbi. It can be considered that the body b makes a virtual movement from PNbi to PVbi. The movement can be represented by a matrix FMbi:

$$FMbi * PNbi = PVbi \tag{7.1}$$

and FMbi here is referred to as the prefix modifying factor of the nominal position of the body instance bi or simply the modifying factor.

It can be seen from (7.1) that the modifying factor FMbi is determined by the nominal position and the verified one:

$$FMbi = PVbi * PNbi^{-1} \tag{7.2}$$

The actual position is represented in the reasoning system as PNbi and FMbi.

7.3.2. Positions of Body Instances Affected by Vision Information

The position of a body instance can be influenced by vision information indirectly in several circumstances. Since the way in which TIES and subassemblies affect positions is fairly complex, discussion of this is reserved for a separate chapter (Chapter 8). In this chapter the relatively simple inference of actions is used to convey an idea of the principles of the system.

Chapter 3 shows how actions can be described in RAPT as purely translational (MOVE), purely rotational (TURN) and a mixture (general MOVE). From the syntax of action statements we can distinguish three classes: explicit action statements, implicit action statements, and general move statements. However, for the purposes of making the modifying factor array neither this classification nor the type of action are important. What is important is the way in which action statements and spatial relationship statements interact. Section 7.3.2.1 and section 7.3.2.2 therefore describe a new classification which depends upon such interactions, and discuss the corresponding effects on modifying factors of bodies being moved.

7.3.2.1. Unspecified Actions

In RAPT, a body can be moved to a position where no specified spatial relationships must be satisfied. The destination can either be specified by an explicit action statement which describes an absolute movement from an original position (see Example 1 below), or be deduced by the geometrical reasoning system from statements about later

situations (see Example 2). This kind of destination can be referred to as an unspecified position and the relevant action can be referred to as an unspecified action. It is necessary to distinguish between specified and unspecified actions. An unspecified action is ~~one which is~~ defined as one which precedes an unspecified position. Whether or not it is an explicit action is irrelevant.

Example 1:

```
... ...
move/b1, perpto, f1 of b1, 50;
turn/b1, about,f2 of b1, 45;
... ...
```

Example 2:

```
... ...
move/b1, perpto, f1 of b1;          remark This is an unspecified
                                          action;
move/b1, perpto, f2 of b1;          remark This is not an
   against/f1 of b1, f1 of a1;            unspecified action because of
   fits/f3 of b1, f5 of a1;               the special relations on the
   ... ...                                destination of the action;
```

In the case of an unspecified action there is a RAPT command to move a body by a certain amount after its position has been verified. Since the verified position may be different from the nominal one, the actual position of the body after some explicit actions may also be different from the corresponding nominal position. The following discussion will establish the relationship between a modifying factor of a body after an unspecified action and that before that action. Suppose the nominal

position of body b in situation (i+1), PNb(i+1), is produced by an unspecified action from PNbi

$$PNb(i+1) = PNbi * Tbi \tag{7.3}$$

where Tbi is a transformation representing an action.

Then the actual position of the body in situation (i+1) (the actual position is not a "verified position" since it has not been verified by vision commands, but it can be considered equivalent to a verified position in the discussion) is

$$\begin{aligned} PVb(i+1) &= PVbi * Tbi \\ &= FMbi * PNbi * Tbi \\ &= FMbi * PNb(i+1) \end{aligned} \tag{7.4}$$

From (7.1)

$$PVb(i+1) = FMb(i+1) * PNb(i+1) \tag{7.5}$$

Therefore in this case

$$FMb(i+1) = FMbi \tag{7.6}$$

and is referred to as the prefix modifying factor of the body instance PNb(i+1). Thus if the modifying factor of a body before an unspecified action is known then that of the body after the action is known, too.

7.3.2.2. Specified Actions

In RAPT, a body can also be moved to a position to satisfy certain relationships with respect to other bodies. These spatial relationships specify the destination of the body being moved in terms of the relative position of the body with respect to others. This kind of destination can be referred to as a specified position and the action which brings the body to this position can be referred to as a specified action. For example, the following segment of a program describes a specified action. This action brings the body b1 to a position to satisfy specified relationships holding between bodies b1 and a1.

```
... ...
 move/b1, perpto, f1 of b1;
 against/f1 of b1, f1 of a1;
 against/f2 of b1, f3 of a1;
... ...
```

Notice that the relevant relationship specifications may not follow the associated specified action statement directly. There may be some other action statements in between which do not change the position of the body concerned. In the following example,

```
... ...
          remark now in situation i;
move/b1, perpto, f1 of b1;
          remark situation i+1;
move/b2, parallel, f2 of b2;
   fits/f3 of b2, f3 of a2;
           remark situation i+2;
```

```
turn/b3, about, f1 of b3, 90;
        remark situation i+3;
   fits/f2 of b1, f2 of a1;
   against/f1 of b1, f1 of a1;
   ... ...
```

If b1 is neither TIED to b2 or b3 nor in the same subassembly with them then the MOVE statement which moves b1 is still a specified action statement though the relevant relationship specifications are in situation i+3. This is because the body b1 is not moved through situation i+1 to i+3, the spatial relationships specified in situation i+3 must be satisfied in situation i+1 by that action.

It is the existence of direct or indirect relationship specifications which determines whether an action is specified or not. Thus even if the action statement which is followed by a relationship specification is explicit the action is still specified, since the destination is a specified position.

In the case of a specified action a body is moved to a place which is specified by a set of relations holding between the body being moved (body a) and another body (body b). If body b has been verified then the destination of body a may be different from its nominal one since vision information may indicate that the actual position of body b to which body a refers deviates from its nominal one, and in order to satisfy the specified relations the destination needs to be adjusted. The following discussion will establish the relationships between a modifying factor of a body after a specified action and that of the reference body. Suppose the nominal position of body a and body b in

situation i are PNai and PNbi respectively and these positions are such that a set of specified relations hold between the bodies. Thus the difference in position of body a with respect to body b is

$$RPab = PNai * PNbi^{-1} = PVai * PVbi^{-1} \qquad (7.7)$$

So if the actual position of body b in situation i is

$$PVbi = FMbi * PNbi \qquad (7.8)$$

then the actual position of body a in situation i must be

$$\begin{aligned} PVai &= PNai * PNbi^{-1} * PVbi \\ &= PNai * PNbi^{-1} * FMbi * PNbi \end{aligned} \qquad (7.9)$$

if the specified relations are to hold. As for equation (7.5), equation (7.9) can be re-written as:

$$\begin{aligned} PVai &= PNai * PNbi^{-1} * FMbi * PNbi \\ &= PNai * PNbi^{-1} * FMbi * PNbi * PNai^{-1} * PNai \\ &= FMai * PNai \end{aligned} \qquad (7.10)$$

where $$FMai = PNai * PNbi^{-1} * FMbi * PNbi * PNai^{-1} \qquad (7.11)$$

So here we get an expression for the modifying factor of body a in situation i which depends upon the modifying factor of body b in situation i, and some constant transformations. If the modifying factor of the reference body and the nominal positions of both the reference body and the body being moved are known then the modifying factor of a body after a specified action can be determined.

7.3.3. Body Instances for which the Vision Information is Irrelevant

There are some body instances for which the vision information has no relevance. These bodies have neither been verified, nor moved to some place to satisfy specified relations with respect to bodies which have been verified. Nor are they members of TIES or subassemblies which have members that have been verified. In this case the nominal positions of these body instances are assumed to be accurate at run time, and the modifying factor of this kind of body instance is an identity matrix. Whether the position of a body instance is dependent upon vision information will be determined by the rules which will be discussed in Chapter 8.

7.3.4. Actual Positions and Actions

It can be seen from the discussion above that the actual position of a body instance can be decomposed into two parts: the nominal position and a modifying factor, and it is only the modifying factor that may be affected by the vision data. When a body instance is verified, its modifying factor is determined by the vision data and the nominal position of the body instance. It has also been shown above that if the actual position of a body instance (bi) is affected by that of a verified body instance and if the relative position of one body with respect to the other is known then the modifying factor of the body instance bi can be determined. As the nominal position for each body instance can be obtained from the current RAPT cycle finding system, and the

influence of vision information can be deduced by analysing the user's program, the modifying factor for every body instance can be determined. It follows then that when the verification vision system is combined with RAPT the inference system can deduce the nominal position of each body instance as usual at compile time while the framework can evaluate modifying factors at run time, and so get the actual positions of every body instance by matrix multiplication.

Now, while the actual positions have been determined, at run time what needs to be known is how the actions to be taken by the robot system have to be modified. The introduction of modifying factors will change the actions on bodies. The transition between two nominal positions can be referred to as a nominal action and the transition between two actual positions can be referred to as an actual action. Suppose body b is moved from nominal position PNbi to PNb(i+1) by a nominal action TNbi. TNbi can be expressed as:

$$PNbi * TNbi = PNb(i+1)$$
$$TNbi = PNbi^{-1} * PNb(i+1) \qquad (7.12)$$

On introducing the modifying factor, the positions of both the starting point and the destination of an action may be changed. Suppose body b is moved from actual position PVbi to PVb(i+1) by an actual action TAbi, then the actual action TAbi can be deduced as:

$$PVbi * TAbi = PVb(i+1)$$
$$TAbi = PVbi^{-1} * PVb(i+1)$$
$$= PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNb(i+1) \qquad (7.13)$$

It can seen from equation (7.13) that if FMbi = FMb(i+1) then the actual action will be the same as the nominal action.

7.3.5. The Modifying Factor as a Prefix and Postfix

In the discussion above, the modifying factor is used as a prefix, and left multiplies a nominal position. In fact, a modifying factor which represents the discrepancy between a nominal position and an actual one can also be represented as a postfix.

7.3.5.1. Use of Postfix

This section derives equations similar to (7.1) - (7.11) but with the modifying factor used as a postfix and referred to as GM instead of FM. Suppose that a body b in situation i makes a virtual movement from its nominal position PNbi to its virtual position PVbi and the movement is represented by a post multiplied matrix GMbi:

$$PNbi * GMbi = PVbi \qquad (7.1')$$

and GMbi here is referred to as the postfix modifying factor of the body instance PNbi.

It can be seen from (7.1') that

$$GMbi = PNbi^{-1} * PVbi \qquad (7.2')$$

In order to establish the relationship between the modifying factor of a body after an unspecified action and that before the action, let us suppose the nominal position of body b in situation (i+1), PNb(i+1), is produced by a fixed amount of unspecified movement from the nominal position PNbi:

$$PNb(i+1) = PNbi * Tbi \qquad (7.3')$$

then the actual position of the body instance is

$$\begin{aligned} PVb(i+1) &= PVbi * Tbi \\ &= PNbi * GMbi * Tbi \\ &= PNbi * Tbi * Tbi^{-1} * GMbi * Tbi \\ &= PNb(i+1) * Tbi^{-1} * GMbi * Tbi \\ &= PNb(i+1) * GMb(i+1) \end{aligned} \qquad (7.5')$$

$$\text{where} \quad \begin{aligned} GMb(i+1) &= Tbi^{-1} * GMbi * Tbi \\ &= PNb(i+1)^{-1} * PNbi * GMbi * PNbi^{-1} * PNb(i+1) \end{aligned} \qquad (7.6')$$

In order to determine the modifying factor of a body after a specified action let us consider a situation i in which body a has been moved by a specified action so that a set of specified relations hold between body a and body b. Suppose the nominal positions of body a and body b in situation i are PNai and PNbi respectively. The relative position of body a with respect to body b is

$$RPab = PNai * PNbi^{-1} = PVai * PVbi^{-1} \qquad (7.7')$$

If the actual position of body b is

$$PVbi = PNbi * GMbi \qquad (7.8')$$

then the actual position of body a is

$$\begin{aligned} PVai &= PNai * PNbi^{-1} * PVbi \\ &= PNai * PNbi^{-1} * PNbi * GMbi \\ &= PNai * GMbi \end{aligned} \qquad (7.10')$$

It can be seen that in this case the modifying factor of body a is the same as that of body b:

$$GMai = GMbi \qquad (7.11')$$

7.3.5.2. Relationships Between Postfix and Prefix Convention

Theoretically speaking, the prefix and postfix modifying factors are alternative. Either form can be used to represent the difference between a nominal position and an actual one, provided that different expressions are used to express the effect of a modifying factor on other body instances. In practice, the complexities of the equations of the two forms which express the effect are different. A prefix modifying factor represents a rotation and translation with respect to its nominal position. Thus the expression of the effect of the modifying factor over an unspecified position under this notation is simple. When

representing the effect of the modifying factor over a specified position, however, the expression is rather complex. These have been shown in equations (7.2) - (7.11). A postfix modifying factor, on the other hand, represents a rotation and translation with respect to the world coordinate system. Thus the expression of the effect of the modifying factor over a specified position is simple while that for an unspecified position is complex. These can be seen from equations (7.2') - (7.11').

As the framework has to deal with the effect of a modifying factor over both an unspecified position and a specified position, neither notation method is overwhelmingly better that the other. The author selects the prefix notation in the following discussions since it makes some rules of establishing and simplifying the effect of a modifying factor a bit simpler. The term modifying factor will mean the prefix one only in the remainder of the thesis. If the potential assembly task needs the RAPT system to deal with a large number of subassemblies and TIES then the postfix notation may be preferable since the subassembly and the TIE will bring many implied specified positions and using the postfix notation can simplify relevant expressions and reduce calculations at run time.

7.4. The Run Time Data Structure

At run time the system needs to access the nominal positions and modifying factors since the actual actions of the robot and other devices are determined from the actual positions which are derived from the nominal positions and the modifying factors. The modifying factors and the nominal positions are both stored at run time in arrays indexed by

body instances.

The elements of the modifying factor array have three possible forms.

1) The modifying factor will be an identity matrix symbol ("I") if the actual position of the corresponding body instance is independent of any vision information and can be assumed identical to its nominal position.

2) The modifying factor will be a position matrix if the corresponding body instance has been verified by vision commands. This matrix is the product of the verified position and the inverse of the corresponding nominal position, as expressed in equation (7.2). If the vision verification step has not been reached then the modifying factor is still a symbolic expression.

3) The modifying factor will be a pointer or a set of pointers if the position of the corresponding body instance is dependent upon another body instance which has been verified in the current situation or one of the preceding situations. Pointers point to modifying factors in the same modifying factor array. The modifying factors pointed to are themselves position matrices, pointers or pointer sets. This linkage is made at compile time according to the dependence of the actual position of the body instance upon verified positions. Rules for determining pointers or pointer sets will be discussed in Chapter 8. When evaluation occurs of a body instance actual position whose modifying factor is a pointer or a set of pointers, the run time system will interpret it by applying a set of rules and will produce an actual modifying factor for evaluating the actual

insert the following sentences in line 7 in p174:

Note that at compile time, the elements which will contain the position matrices at run time contain symbolic expressions of the verified positions rather than symbolic expressions of the discrepancies between the nominal positions and the verified ones. Detailed discussion can be found in Section 7.6.

change line 10 in p174 to follows:

The RAPT geometrical reasoning system, therefore, ...

position of the body instance. The rules for the evaluation of a pointer or a pointer set will also be discussed in detail in Chapter 8.

The pointers and identity matrix symbols are assigned at compile time, while the position matrices are assigned at run time after the vision data have been obtained and the corresponding symbolic positions have been evaluated.

The modifying factor array thus holds both vision information and knowledge about how it affects other body instances whose actual positions are dependent upon verified positions. The reasoning system, therefore, does not need to manipulate the symbolic form of the vision information and so does not need to be altered to cope with vision verification. This feature makes the compile time system work efficiently. Since the effect of a verified position over actual positions of other body instances is represented by pointers pointing among modifying factors, the symbolic expression of each verified position needs to be evaluated only once at run time. As pointers are established at compile time, the run time system only needs to do simple work such as tracing a pointer or multiplying a nominal position matrix with a modifying factor which is a position matrix. Thus the run time system can deal with vision information quickly. Furthermore, since the modifying factor array handles discrepancies between actual positions and nominal ones, nominal positions of body instances resulting directly from the cycle finder are available to the run time system in their original form. Thus, the run time system can work out nominal actions and discrepancies between nominal actions and actual ones separately if necessary. This increases the flexibility of the run time system in making use of vision information.

7.5. The Actual Action Control

In order to bring about desired states the run time system which controls actions of the robot must take account of discrepancies between nominal positions and actual ones of body instances in order to adjust planned movements of the robot. Knowing the modifying factor of each body instance, the actual action which moves a body from an actual position to the next can be calculated by equation (7.13) since nominal positions are already known. An actual action differs from its nominal one if the modifying factors for body instances before and after the action are different.

There are two possible ways to control the robot to move under the adjustment of modifying factors. One can be referred to as a one step control strategy and the other can be referred to as a two step control strategy.

7.5.1. One Step Control Strategy

If the nature of the route of the action which is specified by the user's program is not important and the user is only concerned about the destination of the body to be moved then the run time system can calculate the amount of the actual action by using equation (7.13) directly and order the robot to move accordingly in one step (Fig. 7.4). The effect of this method is the same as that of Corner's method. The advantage of this method is its simplicity and speed of the robot's

performance. It needs only three matrix multiplication operations to calculate the actual action. Since the action is accomplished in one step, it is faster than the two step method which will be discussed below.

7.5.2. Two Step Control Strategies

Sometimes, however, the nature of the planned route does matter. For instance, in order to avoid collision, a body must be moved along a route which maintains a specified relationship with certain body in a segment of its trajectory. In this case the one step control method is no longer suitable. In order to control the trajectory of movements of the robot, the expression of an actual action needs to be re-written in two parts, and the corresponding action of the robot is divided into two steps. The first part indicates the adjustment of the action which is caused by errors in positions of end points of the motion while the second part indicates the movement which maintains the constraints to the movement implied by the user's program. These two parts control the two steps of the action of the robot respecttively.

The decomposition of equation (7.13) can be done in several ways depending upon the nature of the nominal action that the user or system designer wants to keep in the second step of the actual action. When the user wants the movement in the second step to keep the same direction in the world coordinate system as the planned action, equation (7.13) can be re-written in the following way:

$$TAbi = PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNb(i+1)$$

$$= PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNbi * PNbi^{-1} * PNb(i+1)$$

$$= TCbi * TNbi \qquad (7.14)$$

Here $$TCbi = PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNbi \qquad (7.15)$$

It indicates the movement of the robot in the first step.

$$TNbi = PNbi^{-1} * PNb(i+1)$$

denotes the amount of movement in the second step of the action, and it is exactly the same as the nominal action. This decomposition is useful in cases such like that shown in Fig. 7.5 in which the discrepancies between actual destinations and nominal ones contain translations and a rotation about the axis which is parallel to the direction of the nominal action.

Sometimes it is necessary that the movement in the second step must be such that the moving object maintains the same relations with its destination as in the corresponding planned action. In this case the decomposition of the action expressed by equation (7.14) is no longer suitable and the action should be decomposed in another way.

Suppose the nominal positions of body b in situation i and (i+1) are PNbi and PNb(i+1), and their modifying factors are FMbi and FMb(i+1) respectively. The relative position of PNbi with respect to PNb(i+1) is $PNbi * PNb(i+1)^{-1}$. If the actual position of body b in situation (i+1) is:

$$PVb(i+1) = FMb(i+1) * PNb(i+1) \qquad (7.16)$$

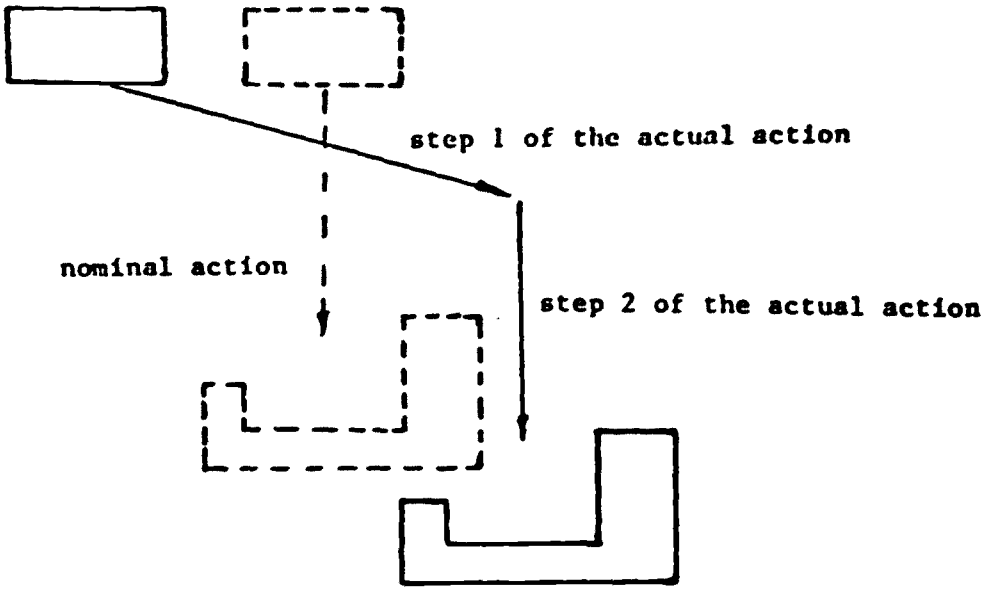


Fig. 7.5. Method 1 of the two step strategy which keeps the action in line with the world coordinate system

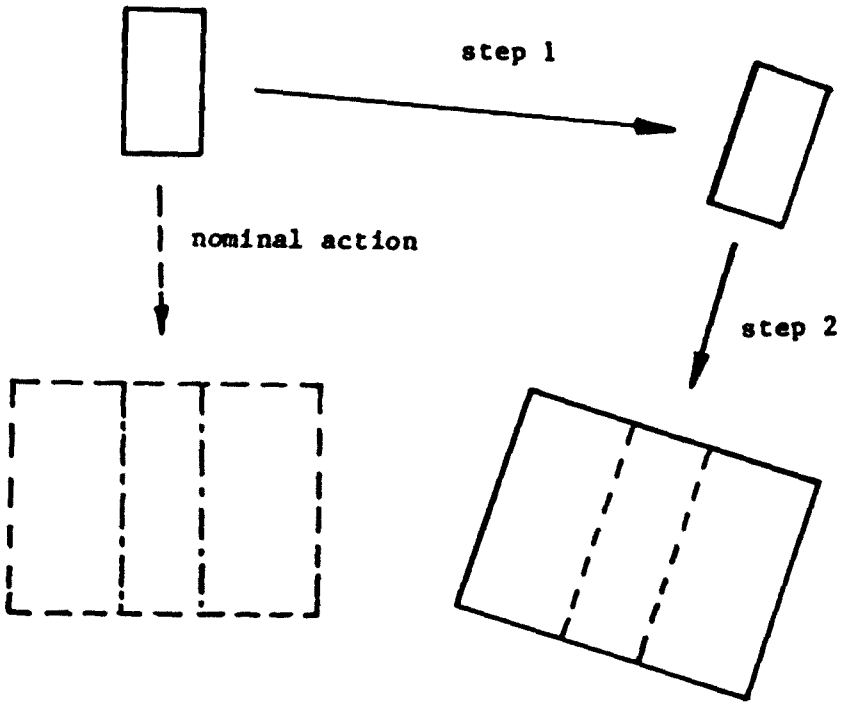


Fig. 7.6. Method 2 of the two step strategy which keeps the action in line with the destination coordinate system

and PNb' is the intermediate position which has the same relative position with respect to PVb(i+1) as PNbi has to PNb(i+1), then

$$PNb' = PNbi * PNb(i+1)^{-1} * FMb(i+1) * PNb(i+1) \qquad (7.17)$$

The first step of the movement Ta1 must bring the body to this position. The second step of the movement Ta2 which will move the body from this position to PVb(i+1) can be expressed as:

$$Ta2 = PNb(i+1)^{-1} * FMb(i+1)^{-1} * PNb(i+1) * PNbi^{-1} * \\ * FMb(i+1) * PNb(i+1) \qquad (7.18)$$

The whole action of body b from PVbi to PVb(i+1) is re-written as:

$$\begin{aligned} TA &= PNbi^{-1} * FMbi^{-1} * FMb(i+1) * PNb(i+1) \\ &= PNbi^{-1} * FMbi^{-1} * PNbi * PNb(i+1)^{-1} * FMb(i+1) * PNb(i+1) * \\ &\quad * PNb(i+1)^{-1} * FMb(i+1)^{-1} * PNb(i+1) * PNbi^{-1} * \\ &\quad * FMb(i+1) * PNb(i+1) \\ &= Ta1 * Ta2 \end{aligned} \qquad (7.19)$$

The usefulness of this method of controlling the action can be seen from the following example. Suppose the user wants to insert a shaft into a hole, and has written a program fragment:

```
... ...
move/shaft;
move/shaft, perpto, f1 of shaft, 20;
fixed/shaft, hole;
... ...
```

This program can be interpreted to mean that the user wants the route of the last movement of the robot to be such that the shaft moves along the axis of the hole, avoiding any collision between the shaft and the edge of the hole. However, if the actual position of the hole is different from its nominal one, then under the control strategy which is expressed either by equation (7.13) or by equation (7.14) the route of the last movement would be modified so that it may not be collinear with the axis of the hole and the action may not succeed. If the system uses the method specified by equation(7.19) to control the movement of the robot, i.e. the system asks the robot to adjust its position first and then move according to the nominal trajectory with respect to the hole, then the shaft will be moved along the axis of the hole in the second step of the movement (Fig. 7.6).

Alternative decompositions can also be made to the action specified by equation (7.13) if other requirements of the action control must be met. For example, the action can be decomposed into three or even more steps if needed. Because the modifying factor array handles vision information and its effects on actual positions of body instances separately from the nominal position, it is possible to make use of the variety of the control strategies.

The selection of the control strategy of the actual action is not a burden on the user. At the moment, this will be done by the designer of the run time system which controls the actual action of the robot. In the future, if an automatic task planner and a collision avoidance controller are added to the RAPT system then they can share the responsibility of choosing the control strategy. The verification vision

system, therefore, does not need to provide commands to the user to do this.

7.6. The Compile Time Work

The compile time modifying factor array is similar to that of the run time except that the symbolic expressions have not been evaluated. The framework needs some compile time work to be done in order to establish connections between vision information and dependent body instances. The final result of the compile time work is a properly assigned modifying factor array which is ready to be used by the run time part of the framework. The compile time work can be divided into three phases. They are the initiation phase, the reasoning phase and the simplification phase.

7.6.1. The Initiation Phase

In the initiation phase, the modifying factors of all the body instances in situation 1 are assigned an identity matrix symbol "I". This is because in the first situation no body has been either moved or verified, and therefore they can be assumed to be at their nominal positions. The modifying factors of the body instances of the world in all situations are also assigned an identity matrix symbol since the world represents the frame of the working station. It is the reference of the positions of other body instances and must always be considered to be at its nominal position.

7.6.2. The Reasoning Phase

In the reasoning phase, the modifying factors of body instances in the following situations are assigned either the symbolic expression of a verified position or a pointer or a set of pointers. This is done using the information provided by two of the tables formed by the RAPT system from the input program. One of the tables is the action table which records the object and the nature of each action. In this verification vision system the nature of the action can be vision verification. The other is a TIE table which records which bodies are TIED together or constitute a subassembly in each situation. This matter is discussed in more detail in Chapter 8. If the position of a body instance is verified then its modifying factor is assigned the symbolic expression of the verified position. Otherwise a pointer or a set of pointers will be assigned to the modifying factor. The assignment of a pointer or a set of pointers depends upon the state of the body instances indicated by the two table and corresponding rules. These pointers denote the connections between modifying factors. If a pointer points to a modifying factor whose content is a symbolic expression of a verified position then it represents the route along which the effect of the vision information propagates. If a pointer points to an identity matrix symbol then it indicates that the position of the corresponding body instance has no relation to any vision information. If a pointer points to another pointer or a set of pointer then it means the modifying factor depends upon others.

Sometimes a vision task may be specified incompletely, i.e. there are not enough vision commands in the corresponding COMBINE package. Thus the symbolic reasoning system cannot fix the body to be verified

with respect to the world. Instead, it produces a constrained symbolic relationship between the body and the world. In this case, the system reports the fact to the user and ignores the vision command (see Chapter 5), and therefore assigns a pointer to the modifying factor of the body to be verified which points to the modifying factor of the same body in the previous situation.

7.6.3. The Simplification Phase

In the simplification phase, each pointer or set of pointers in the modifying factor array will be checked to see whether it can be substituted by an identity matrix symbol or by another pointer so that the structure of the modifying factor array can be simplified. For example, if a pointer points to an identity matrix symbol then the modifying factor which contains the pointer can be replaced by an identity matrix symbol. The simplification rules will also be discussed in Chapter 8. Notice that at compile time the symbolic expressions in the modifying factor array are of the verified position PVbi rather than of the modifying factor FMbi. The real modifying factors will be obtained at run time when the symbolic positions are evaluated and multiplied by the inverses of corresponding nominal positions.

7.7. Modifying Factors in Symbolic Reasoning and Vision Commands

As well as influencing the symbolic reasoning, the introduction of modifying factors will also influence the way in which vision commands create the symbolic feature and control the camera. Modifying factors

indicate the discrepancies between nominal positions of body instances and their actual ones. This information can be used not only to adjust the action of the robot but also to adjust the reasoning of the symbolic expression of a verified position and provide the vision facilities a better estimation of the place where the feature to be verified can be found.

In a vision command package, the INVIOLATE statement is used to indicate the most reliable relationships holding between the body to be verified and a reference body. The modifying factor for the body instances of this reference body may not be an identity matrix. This means that it may not be at its nominal position, though the relationship mentioned in the INVIOLATE statement still holds. The modifying factors from previous vision verification enable the reasoning system to make better predictions for positions of bodies in subsequent vision verification. For example, if the vision system verifies the position of body A and subsequently verifies a body B which has an inviolate relationship with body A then the reasoning system should take account of what it has learnt about the actual position of body A in the prediction for body B.

As discussed in Chapter 5, if a reference feature does not belong to the world directly then the INVIOLATE statement will transform the position of the feature into its position in the world coordinate system and consider it as a world feature. Before the introduction of modifying factors the transformation can be done by a matrix multiplication

Pf * PNbi

where Pf is the position of the feature in the local coordinate system of the reference body and PNbi is the nominal position of the reference body in the situation directly preceding the vision verification. In order to enable the inviolate relation to represent the actual circumstances, the symbolic reasoning system needs to use the symbolic actual position of the reference body to take part in the symbolic reasoning, instead of the nominal one. Every time an INVIOLATE statement is met, the symbolic reasoning system will multiply the nominal position of the reference body by the corresponding modifying factor symbolically, provided that the modifying factor is not an identity matrix. Thus the transformation of the position of the reference feature from its local frame into the world coordinate system can be expressed as:

$$Pf * FMbi * PNbi \qquad (7.20)$$

At run time, the corresponding modifying factor will have already been evaluated in a previous situation before it is required and therefore the symbolic expression of the position of the body to be verified can be evaluated in the situation in which the vision verification takes place.

The run time vision facilities need to use the information provided by modifying factors of the body instances in order to estimate the position of the expected feature more accurately at run time. Actual positions of the camera to be used and the body instance to be verified are needed to take the places of their nominal counterparts. Since the vision facilities do not need to manipulate these actual positions at compile time, no extra work is necessarily to be done in the symbolic reasoning. The window suggester and the face generator will get access

to the modifying factor array directly at run time in order to multiply the nominal positions of the camera and the body to be verified by their modifying factors in the situation directly preceding the corresponding vision verification.

The data flow chart of the RAPT system with the verification vision is shown in Fig. 7.7.

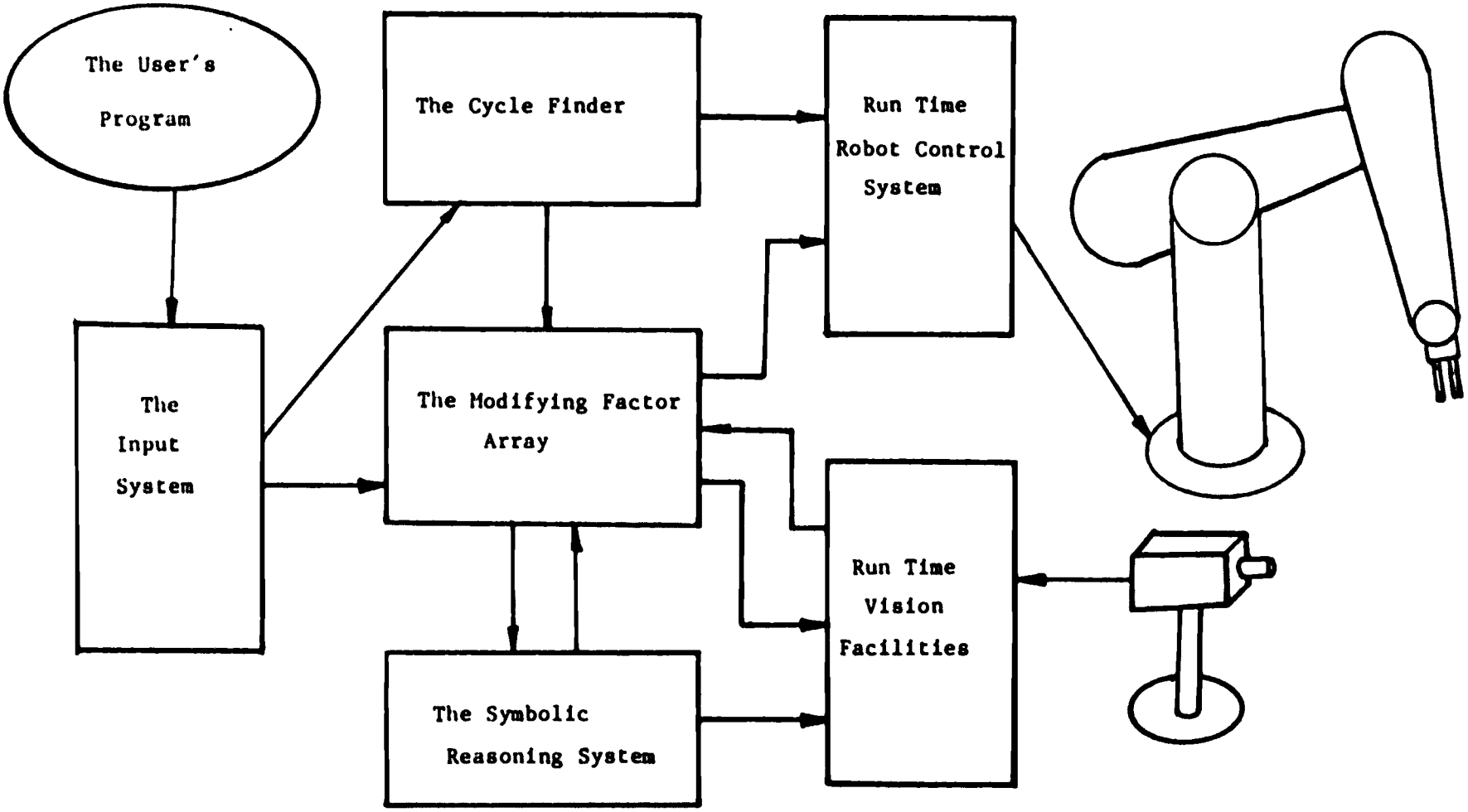


Fig. 7.7. Data flow chart of the verification vision system within RAPT

Chapter 8. Rules for Filling the Modifying Factor Array

Chapter 7 discussed the relationship between the vision information and actual body positions and described the framework that is used to handle the vision data. This involves the use of a modifying factor array. The present chapter discusses in more detail the rules which are used in assigning pointers in this array. During the reasoning phase (see Section 7.6.2) all elements of the modifying factor array are assigned pointers or sets of pointers except those which are initialized or those which refer to body instances which are verified directly by vision commands. The pointers represent the dependence of a modifying factor upon others, and this dependence is determined by the state of the associated body instances in the user's program, such as being moved, TIED and so on. Pointers for a particular situation can point to modifying factors in the same or previous situations. However, they can never point to the modifying factors in later situations. This restriction represents the fact that the actual position of a body instance can only be dependent upon the past and present status of the environment, and cannot be affected by any future events. The law of causality is ~~also~~ valid here.

In order to assign the pointers and then simplify the modifying factor array, a set of rules must be established. For convenience, the rules for making the pointers will be referred to as linking rules in the following discussion. These rules describe the connections between modifying factors in the situation directly following each corresponding action or vision command package.

8.1. Linking Rules for Actions

As discussed in Chapter 7, in order to make the modifying factor array the action statements are classified into two sorts: specified and unspecified action statements, depending upon the way in which action statements and spatial relationship statements interact. The rules for setting a pointer in an element of the modifying factor array distinguish these two sorts of action statement. The nature of pointers is also different depending upon the type of the destination of the corresponding body instance.

8.1.1. The Rule for Unspecified Actions

Suppose a body is moved by an unspecified action. The actual position of the body after the action is only determined by the actual position of the body before the action and the amount of the displacement. This case has been discussed in Section 7.3.2.1 in detail. It can be seen from equation (7.6) that the modifying factor of a body after an unspecified action is just the same as that before the action. Therefore the pointer for the modifying factor of the body instance after the action points to the modifying factor of the body instance before the action. This kind of pointer means that the corresponding modifying factor is equal to what is pointed at. An unmoved body can be considered as being moved by an unspecified action represented by an identity matrix. Its modifying factor is therefore also a pointer pointing to that of the same body in the previous situation.

8.1.2. The Rule for Specified Actions

Suppose a body is moved by a specified action. The actual position of the body after the action is determined by the specified relative position of the body being moved with respect to the reference body and the actual position of the reference body. This case has been discussed in detail in Section 7.3.2.2. The modifying factor after the action of the body being moved depends upon the modifying factor of the reference body and the nominal positions of both the reference body and the body being moved. Thus the pointer in the situation directly succeeding the action for the modifying factor of the body being moved points to that of the reference body after the action. The meaning of this kind of pointer, however, is different from that assigned by an unspecified action. This kind of pointer must be interpreted by equation (7.11). Since the pointer which is assigned by a specified action points to the modifying factor of another body while that which is assigned by an unspecified action points to the modifying factor of the same body in the previous situation, it is easy to distinguish them.

Sometimes the destination of a specified action refers to more than one body. If the modifying factors of the reference bodies are not identity matrices then whether the specified relationships can be realized in the real environment depends upon the types and positions of the features referred by the relationships and the modifying factors of the reference bodies. To judge this needs a complex calculation, and sometimes this is impossible since some modifying factors which are dependent upon vision information cannot be known until run time. In order to solve this problem, a criterion is adopted: If a specified action refers more than one reference body then the modifying factors of the

reference bodies b, c, ... must satisfy the condition that the actual ratio between these reference bodies must be the same as the nominal one:

$$FMbi * PNbi * (FMci * PNci)^{-1} = PNbi * PNci^{-1}$$

... ...

This condition covers the case that the modifying factors of all reference bodies are identity matrices. The proof of the sufficiency of this condition is given in Appendix III for the case in which two bodies are referred by the spatial relationships for convenience. The following discussion will also restricted to the case of two reference bodies.

The condition

$$FMbi * PNbi * (FMci * PNci)^{-1} = PNbi * PNci^{-1}$$

can be satisfied in two ways. In the first case both FMbi and FMci must be the identity matrices. Otherwise the modifying factors must satisfy the following equation

$$FMbi = PNbi * PNci^{-1} * FMci * PNci * PNbi^{-1}$$

This is the same as equation (7.11). Thus the condition can be examined by whether the modifying factor of a reference body points to that of the other. If the condition

$$FMbi * PNbi * (FMci * PNci)^{-1} = PNbi * PNci^{-1}$$

is not satisfied or cannot be examined in either way then the compile time facility will report the case to the user and then refer the modifying factor of the body being moved to that of the reference body in the first associated relationship specification. This discussion also stands for the case in which TIES and subassemblies are involved.

8.1.3. Summary

The following is the summary of the linking rules for actions:

A1. If a body is moved to an unspecified position or is unmoved then the pointer of the body instance points to its modifying factor in the previous situation.

A2. If a body is moved to a specified position then the pointer of the body instance after the action points to the modifying factor of the body instance of the body to which the relationships refer (reference body) after the action. If there are more than one reference body appearing in the relationships, then the modifying factors of the reference bodies must satisfy the condition

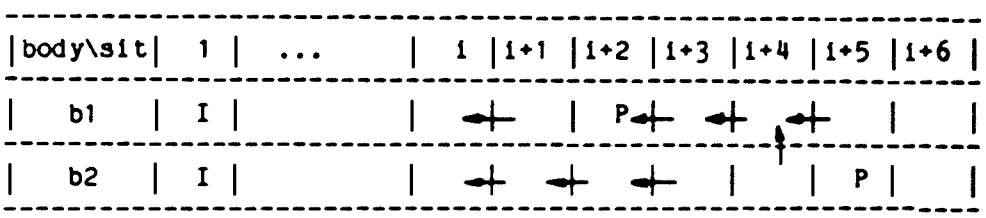
$FMbi * PNbi * (FMci * PNci)^{-1} = PNbi * PNci^{-1}$

otherwise an error is reported and the reference body in the first relationship specification is used in determining the pointer of the body to be moved.

Notice that the linking rules discussed are associated with individual bodies only. These bodies are neither members of a subassembly nor TIED to other bodies. The linking rules for bodies of TIES or subassemblies are discussed in following sections. The following example shows how the rules A1 and A2 work.

```
remark defined bodies b1 b2;
... ...
remark now in situation i;
```

```
move/b1,perpto,f1 of b1, 5;
     remark sit i+1, rule A1;
verify/b1;   remark abbreviation for a set of vision
                commands which verifies b1, sit i+2;

move/b2,parlel,f1 of b2;
     remark sit i+3, rule A1;
turn/b2,about, f2 of b2;
  fixed/b2,b1;
remark abbreviation for a set of relations which completely
       defines the position of b2 with b1, sit i+4, rule A2;

verify/b2;        remark sit i+5;

... ...
terapt;
```



8.2. Linking Rules for TIES

In RAPT, when bodies are TIED together their nominal positions maintain the same ratio before and after any actions. In practice the nominal ratio may not hold. For instance, suppose the manipulator of the

robot grasps a body at a place which differs from the specified position. The actual relative position of the body with respect to the manipulator, therefore, is different from the nominal one. However, if the body is TIED to the manipulator then the action of the manipulator is also applied to the body and so if two bodies are really TIED together then the actual positions of the bodies must keep the same ratio both before and after any actions. The changing of the modifying factor of one body must affect that of the other. The only exception happens when local vision commands (see Section 5.3.2) are used to check the actual relationships between the TIED bodies. In that case only the modifying factor of the body instance which is verified by local commands is subject to change.

8.2.1. The Effect of Unspecified Actions

The effect of the action on the modifying factor of the body being moved directly has been discussed in Chapter 7 and Section 8.1. The following will discuss the effect of the action on the modifying factor of the body which is TIED to the body being moved by an unspecified action. When a body is moved by an unspecified action, another body which is TIED to it must be moved by the same amount of unspecified action. Suppose that body b is TIED to body a and body a is moved by an unspecified action Tai between situation i and (i+1). The actual positions of the bodies before the action are:

$$PVai = FMai * PNai \qquad (8.1)$$

and

$$PVbi = FMbi * PNbi \tag{8.2}$$

respectively. The nominal position of the body a after the action is

$$PNa(i+1) = PNai * Tai \tag{8.3}$$

Since body b is TIED to body a, it keeps the same relative position with respect to body a. Thus

$$\begin{aligned} PNb(i+1) &= PNbi * PNai^{-1} * PNa(i+1) \\ &= PNbi * PNai^{-1} * PNai * Tai \\ &= PNbi * Tai \end{aligned} \tag{8.4}$$

According to the discussion in Section 8.1, the modifying factor of body a in situation (i+1) is

$$FMa(i+1) = FMai$$

and the actual position of body a in that situation is

$$PVa(i+1) = FMai * PNai * Tai \tag{8.5}$$

Since it is assumed that body b is actually TIED to body a in the real world, its actual position after the action must keep the same ratio with respect to body a as that before the action. Thus the actual position of body b in situation (i+1) is

$$\begin{aligned} PVb(i+1) &= PVbi * PVai^{-1} * PVa(i+1) \\ &= FMbi * PNbi * (FMai * PNai)^{-1} * FMai * PNai * Tai \\ &= FMbi * PNbi * Tai \end{aligned}$$

$$= FMbi * PNb(i+1)$$

$$= FMb(i+1) * PNb(i+1) \qquad (8.6)$$

and

$$FMb(i+1) = FMbi \qquad (8.7)$$

It can be seen that each member of a TIE keeps the same modifying factor before and after an unspecified action. A pointer which points to the modifying factor of the same body in the previous situation can be assigned to each member of a TIE no matter whether it is moved directly by an action statement or it is moved indirectly by the effect of the TIE.

8.2.2. The Effect of Specified Actions

Now let us consider the effect of a specified action statement on the modifying factor of the body being TIED to a body being moved. Suppose body b is TIED to body a and body a is moved by a specified action so that specified relationships between body a and body c are to be satisfied. Positions of these bodies before the action are:

$$PVai = FMai * PNai$$

$$PVbi = FMbi * PNbi$$

$$PVci = FMci * PNci$$

respectively. As discussed in Section 8.1, the actual position of body a after the action is

$$PVa(i+1) = PNa(i+1) * PNc(i+1)^{-1} * FMc(i+1) * PNc(i+1)$$

$$= FMa(i+1) * PNa(i+1) \tag{8.8}$$

where

$$FMa(i+1) = PNa(i+1) * PNc(i+1)^{-1} * FMc(i+1) * PNc(i+1) * * PNa(i+1)^{-1} \tag{8.9}$$

Exactly the same as the rule discussed in Section 8.1.3 and equation (7.11), a pointer which points to body c in situation (i+1) is assigned to the modifying factor of body a in situation (i+1).

Since body b is TIED to body a, it must keep the same relative position with respect to body a both before and after the action. Thus the actual position of body b in situation (i+1) is

$$\begin{aligned} PVb(i+1) &= PVbi * PVai^{-1} * PVa(i+1) \\ &= PVbi * PVai^{-1} * PVa(i+1) * PNb(i+1)^{-1} * PNb(i+1) \\ &= FMb(i+1) * PNb(i+1) \end{aligned} \tag{8.10}$$

where

$$\begin{aligned} FMb(i+1) &= PVbi * PVai^{-1} * PVa(i+1) * PNb(i+1)^{-1} \\ &= FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * * PNa(i+1) * PNb(i+1)^{-1} \end{aligned} \tag{8.11}$$

In order to respresent this expression, a pointer triple can be assigned as the modifying factor of body b in situation (i+1). The triple has the form:

[p1, p2, p3]

where p1 points to the modifying factor FMbi, p2 points to FMai and p3

points to FMa(i+1). At run time, the triple will be evaluated according to equation (8.11).

There are two bodies in a TIE and the relationships which specify the destination of an action can refer to either of them. Since the action applies equally to the two TIED bodies, in determining the modifying factor, we can consider that the body being moved directly is the one referred in the destination specification.

When both bodies in a TIE are referred by a specified action, the situation is a bit complex. The introduction of the modifying factor restricts the condition on which the spatial relationships are given in this case. When the spatial relationships refer both bodies in the TIE, it is uncertain whether the specified relationships can be realized in the real environment when the modifying factors are taken into account. To examine this in general needs a complex calculation and sometimes is impossible since some information is dependent upon vision. In the work discussed in this thesis, referring to both bodies in a TIE by spatial relationships is only allowed when the spatial relationships refer to the same reference body and the actual ratio between the bodies in a TIE is the same as planned:

$$FMai * PNai * (FMbi * PNbi)^{-1} = PNai * PNbi^{-1}$$

where a and b are the bodies in the TIE. This is a sufficient condition which is easy to examine at compile time. The proof of the sufficiency of this condition can be found in Appendix IV. The ratio between body a and body b maintains the same after they are TIED. Thus examining the condition in situation i can be done by examining the condition in the

situation in which the current effective TIE is declared. The method is the same as that discussed in Section 8.1.2.

If the reference body is in the TIE then the above rule is not applicable. If the rule were used, it would lead to an unsolvable pointer circle. A relationship between a body being moved and a body which is already TIED to it is used to describe the relationships between the two bodies rather than to specify the destination of a movement. Note that in RAPT, two bodies can be TIED together without necessarily giving any relationship specifications between them. When bodies are tied together, the action on one must be copied to the other body in the TIE. Therefore a relationship specification between either of the bodies in the TIE and another body serves to define the destination of both bodies. If an action statement is followed by any relationship specifications which hold between the body being moved or the body TIED to it and a body which is not TIED with it then the action statement is considered as a specified action statement. Otherwise it must be tackled as an unspecified action statement and the rule described in Section 8.2.1 must be applied.

8.2.3. The Effect of Vision Commands

Vision command packages are classified into two classes: global ones and local ones. A local package is produced when the reference body in the INVIOLATE statement is TIED to the body being verified or belongs to the same subassembly as it. This has been discussed in Section 5.3.2. The difference between these two classes is that a global vision command package updates the actual positions of all bodies in the TIE or

subassembly by the use of vision information while a local one only updates the actual position of the body being verified.

8.2.3.1. Global Vision Command Package

The linking rule for global vision commands is quite similar to that for specified actions. Suppose body b has been TIED to body a which is verified by a global vision command package between situation i and (i+1). The modifying factor of body a in situation (i+1) is determined by its nominal position and vision information as discussed in Section 7.3.1. At compile time the symbolic equation of the verified position is assigned to the corresponding element of the modifying factor array. The actual modifying factor will be obtained at run time by the use of equation (7.2). The actual position of body b in situation (i+1) is

$$\begin{aligned} PVb(i+1) &= FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * PNa(i+1) \\ &= (FMbi * PNbi) * (FMai * PNai)^{-1} * \\ &\quad * (FMa(i+1) * PNa(i+1)) * PNb(i+1)^{-1} * PNb(i+1) \\ &= FMb(i+1) * PNb(i+1) \end{aligned} \tag{8.12}$$

where

$$\begin{aligned} FMb(i+1) &= FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * \\ &\quad * PNa(i+1) * PNb(i+1)^{-1} \end{aligned} \tag{8.13}$$

Equation (8.13) is the same as equation (8.11). Thus a pointer triple can be assigned as the modifying factor of body b in situation (i+1). The pointer triple is the same as that discussed in Section 8.2.2.

8.2.3.2. Local Vision Command Package

Local vision commands update the actual position of the body being verified only. The body which is TIED to it is assumed not moved and therefore keeps the same modifying factor as in the previous situation. Thus the modifying factor of the body being verified is assigned the corresponding symbolic expression while that of the body being TIED to it is assigned a pointer which points to its own modifying factor in the previous situation. Note that if a vision command involves a body which is TIED to the world then it will be treated as a local vision command.

8.2.4. Termination of the Effect of a TIE on Linking Rules

The effect of the mutual influence between modifying factors of members of a TIE does not necessarily disappear immediately after the revocation of the TIE. It is more natural to keep this effect for a period until certain conditions are met. For example, suppose the manipulator is TIED to a block and moves the block to a place. The manipulator is then UNTIED from the block and moves away waiting for the vision system to verify the position of the block. If the user does not ask the vision system to verify the new position of the manipulator then the unspecified action which brings the manipulator to the new position is assumed to be carried out accurately and the relative position of the manipulator with respect to the block can be worked out. Thus it is reasonable to use the vision information obtained by verifying the position of the block to update the system's knowledge about the position of the manipulator as well. On the other hand, if, in the above example, the manipulator moves to a place in order to satisfy specified

relationships with respect to another body after it is UNTIED from the block then its new position will be restricted by the actual position of that other body and the verified position of the block cannot be used to update the actual position of the manipulator.

In order to make full use of vision information to update the knowledge of the robot system about the environment, the compile time part of the framework assumes that the effect of a TIE over linking rules remains after the revocation of the TIE until one of the following two circumstances are encountered. The first circumstance is that one body in the revoked TIE is moved by a specified action. Since the destination of the body being moved is determined by its nominal position and the actual position of the reference body, a pointer which points to the modifying factor of the reference body is assigned to the modifying factor of the body being moved. The other body in the revoked TIE has not been moved. Hence it keeps the same position as in the previous situation, and a pointer which points to the modifying factor of the same body in the previous situation is assigned as its modifying factor. Thus the relationships between the actual positions of the two bodies which were established by the TIE no longer exists and the further change of the actual position of one body cannot be used to deduce the change of the actual position of the other.

The second circumstance is that the two bodies in the revoked TIE have each been verified. When the user asks the vision system to verify actual positions of the two bodies individually it means that the relationship between the bodies which have been UNTIED is in doubt. In this case the vision information provides more reliable evidence to show where the bodies are than that implied by a revoked TIE statement. Thus

the effect of the mutual influence between the actual positions of the bodies implied by the former TIE does not need to last.

8.2.5. Tree Structure of the Super TIE

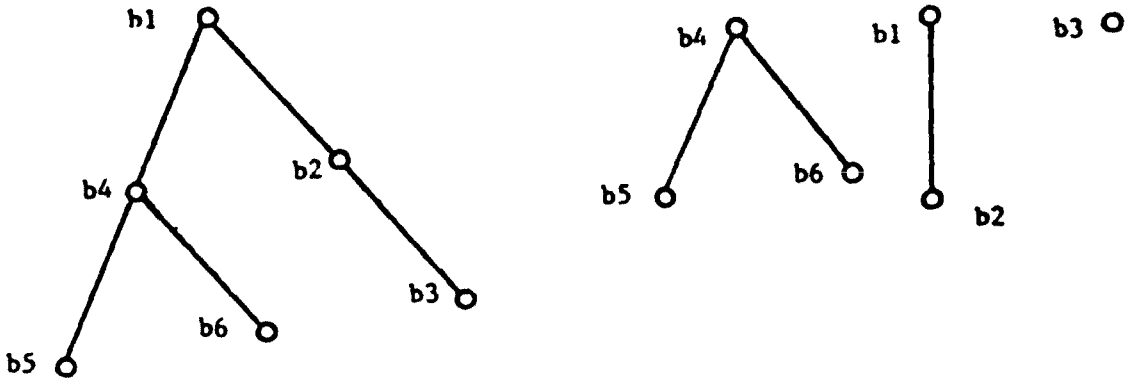
Suppose n bodies b1, ..., bn are TIED together by n-1 TIE statements. The action of one body in this group must be applied to every other. For convenience, this group of TIED bodies is referred to as a super TIE. The graph of a super TIE can be represented by a tree. For example, the following codes:

```
TIED/b1, b2;
TIED/b3, b2;
TIED/b1, b4;
TIED/b4, b5;
TIED/b4, b6;
```

define a super TIE. Its graph is shown in Fig. 8.1(a). After the declaration

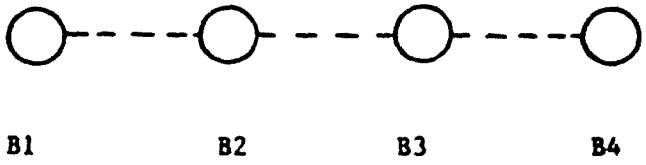
```
UNTIED/b1, b4;
UNTIED/b2, b3;
```

the former super TIE is divided into three pieces. They are shown in Fig. 8.1(b). For convenience, the pieces resulting from the revoking of a super TIE are referred to as sub-super TIE. A sub-super TIE may be an individual body, an ordinary TIE or a group of bodies which are TIED together.

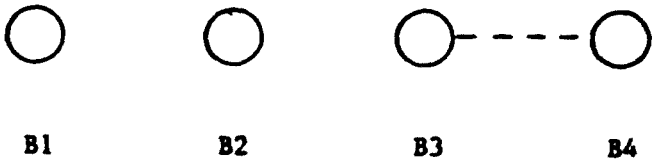


(a) before being broken (b) after being broken

Fig. 8.1. Tree structures of the super TIE
"bi" represents a body instance



(a) before a body in B2 has been moved by a specified action



(b) after a body in B2 has been moved by a specified action

Fig. 8.2. The effect of the specified action on linking rules in a broken super TIE. "Bi" represents a sub-super TIE, dashed lines represent broken TIEs

The discussions in Section 8.2.1, 8.2.2 and 8.2.3 can be applied directly to a super TIE structure. The discussion in Section 8.2.4 (termination of the effect of a TIE) needs to be expanded a bit when being applied to a super TIE.

8.2.5.1. Specified Actions

Suppose a super TIE B has been broken into some sub-super TIES B1, ..., Bm. If a body in Bk is moved by a specified action then the effect of the former super TIE on the linking rules between Bk and other sub-super TIES terminates. This is because the specified action specifies new actual positions of bodies in Bk with respect to a new reference body. The relationships between bodies in Bk and those in other sub-super TIES which are implied by a revoked TIE statement may no longer exist.

The relationships between bodies in other sub-super TIES may still be assumed to remain if necessary as there is no indication either to support this assumption or to conflict with it. In order to implement the rule more easily, the framework terminates the influence between other sub-super TIES if the propagation of this influence passes through Bk.

Fig. 8.2 shows an example of the operation of the rule. In this example, a super TIE has been broken into four sub-super TIES B1, B2, B3 and B4. The influence between actual positions of bodies in sub-super TIES B1, B2 and B3 terminates after a body in B2 is moved by a specified

action. However, the influence between B3 and B4 continues since this effect does not propagate through B2.

8.2.5.2. Vision Commands

Suppose a super TIE B has been broken into some sub-super TIES B1, ..., Bm. If a body bj1 in Bj has been verified by global vision command then the effect of the former super TIE on the linking rules still holds. If a second body bj2 in Bj is verified by global vision commands then the verification of bj2 does not change the effect of the former super TIE on the linking rules either. The reason is that both the bodies belong to the same sub-super TIE which is still valid. The second verification has the same effect as the first one. The vision information obtained from these verifications does not show whether the relative position of a body in this sub-super TIE with respect to a body in another sub-super TIE is changed or not. Thus it is reasonable to assume it still holds.

If a body in Bj is verified by global vision commands after some bodies in another sub-super TIE Bk have been verified by global vision commands then the effect of the former super TIE on the linking rules between Bj and other sub-super TIES terminates. In this case the verification vision system provides enough information about new actual positions of bodies in these two sub-super TIES. This information indicates the new ratios between these two groups of bodies which is more reliable than that implied by the revoked TIE statement. Thus the effect of that revoked TIE must cease. However, the vision information gives no hint about the relative positions of bodies in other sub-super TIES with

respect to either Bj and Bk. In order to avoid contradiction, the framework must terminate the mutual influence between other sub-super TIE which propagates via Bj.

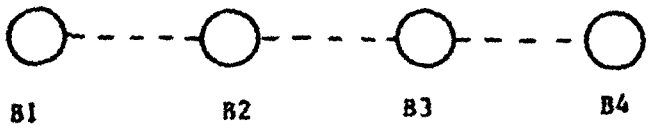
Fig. 8.3 shows an example of this rule. A super TIE has been broken into four sub-super TIES B1, B2, B3 and B4. The influence between actual positions of bodies in sub-super TIES B1, B2 and B3 terminates after a body in both B2 and B3 has been verified by global vision commands in this sequence. The influence between B3 and B4 continues.

8.2.6. Summary of the Linking Rules for TIES

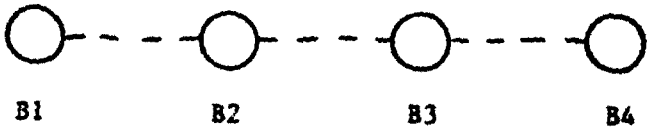
The linking rules for TIE statements can be summarized as follows:

T1. In a super TIE B, if body bj is moved in situation i by a specified action which brings about some relationships between body bj and body C which is not a component of the super TIE, then the pointer of body bj will point to the modifying factor for body C in situation i while the modifying factors of other bodies in the super TIE are pointer triples. The triples have the form [p1,p2,p3] where p1 points to the modifying factor for the same body in situation (i-1), p2 to the modifying factor for bj in situation (i-1) and p3 points to the modifying factor for bj in situation i. If the reference body C is also a component of the super TIE then rule (T4) is applied.

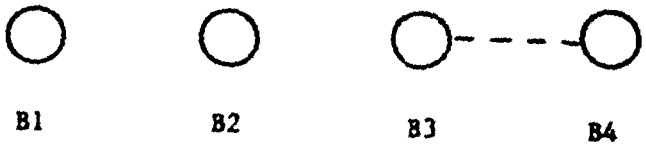
T2. In a super TIE B, if body bj is verified by a set of global vision commands in situation i, then its modifying factor will be assigned a symbolic position expression "P" while that of



(a) before any body has been verified



(b) after a body in B2 has been verified



(c) after a body in B3 has been verified

Fig. 8.3 The effect of vision commands on linking rules in a broken super TIE

other bodies in the super TIE are pointer triples. The triples have the same form and contents as in rule (T1).

T3. In a super TIE B, if body bj is verified by a set of local vision commands in situation i, then its modifying factor will be assigned a symbolic position expression "P" while that of other bodies in the super TIE will refer to their modifying factor in situation (i-1). Note that the influence of a super TIE over the local/global decision will continue after the TIE is broken until the termination.

T4. In a super TIE B, if none of the conditions mentioned in (T1)-(T3) are met, the modifying factors for the bodies in this situation will point to their modifying factors in the previous situation.

T5. After a super TIE B has been broken into sub-super TIES B1, ..., Bm, if no bodies in any of the B1, ..., Bm are either verified by global vision commands or moved by a specified action then rule (T4) is applied.

T6. After a super TIE B has been broken into sub-super TIES B1, ..., Bm, if a body bj in sub-super TIE Bj is moved by a specified action which brings about some relationships between body bj and body C then rule (T1) is applied to the new super TIE Bj while rule (T4) applied to other new super TIES, and the effect of the former super TIE on the linking rules which is related to Bj is terminated hereafter.

T7. After a super TIE B has been broken into sub-super TIES B1, ..., Bm, if no bodies in any of the B1, ..., Bm have been either verified or moved by a specified action since then and if a body bj in Bj is to be verified by global vision commands then rule (T2) is applied to all the components of the original super TIE.

T8. After a super TIE B has been broken into sub-super TIES B1, ..., Bm, if some bodies in Bj have been verified by global vision commands and another body also in Bj is to be verified by global vision commands then rule (T2) is applied to all the components of the old super TIE. If some bodies in Bk have been verified by global vision commands and a body in Bj is to be verified by global vision commands then rule (T2) is applied to Bj while rule (T4) is applied to all other components of the old super TIE, and the effect of the former super TIE on the linking rules which is related to Bj is terminated hereafter.

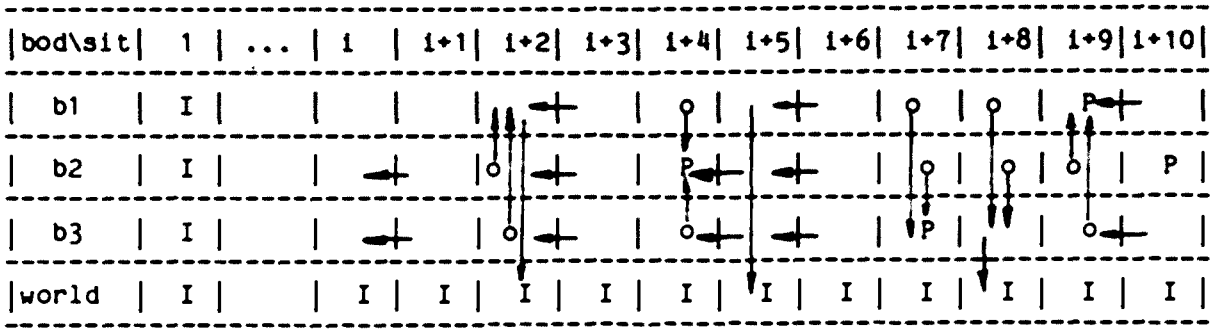
T9. For a super TIE which has been TIED to the world, global vision commands which attempt to verify the position of any component of the super TIE will be dealt with as local vision commands and rule (T3) is applied.

The following example shows the use of these linking rules.

```
remark bodies b1 b2 b3 have been defined;
... ...
remark now in situation i;

move/b1,parlel, f1 of b1;      remark sit i+1, rule A2;
   fixed/b1, b2;

tied/b1,b2;
tied/b1,b3;

move/b1,perpto, f1 of b1;      remark sit i+2, rule T1;
```

```
    fixed/b1, world;

untied/b1,b2;

  move/b1,parlel,f6 of b1,30;     remark sit i+3, rule T5;

  verify/b2;                      remark sit i+4, rule T7;

untied/b1,b3;

  move/b1;      remark sit i+5, rule T6;
    fixed/b1, b3;

tied/b1,b2;
tied/b1,b3;

  move/b1,perpto,f1 of b1, 50;     remark sit i+6, rule T4;

  verify/b3;         remark sit i+7, rule T2;

  move/b3;      remark sit i+8, rule T1;
    fixed/b3, world;

untied/b1,b3;
untied/b1,b2;

  verify/b1;      remark sit i+9, rule T7;

  verify/b2;      remark sit i+10, rule T8;
```

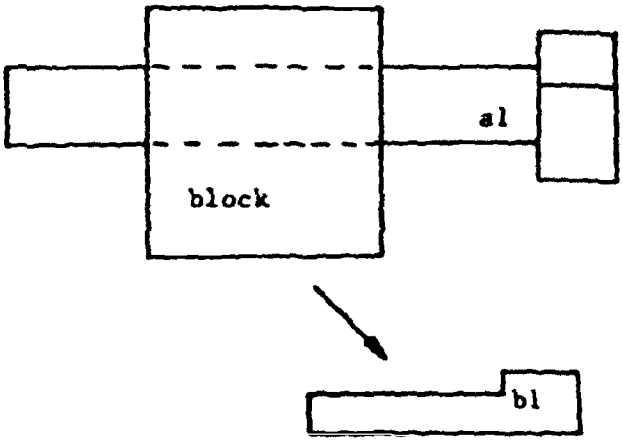
```
... ...
terapt;
```



** o--> represents the pointer triple. A triple pointing from ai to bi contains 3 pointers. One points to a(i-1), one to b(i-1) and one to bi.

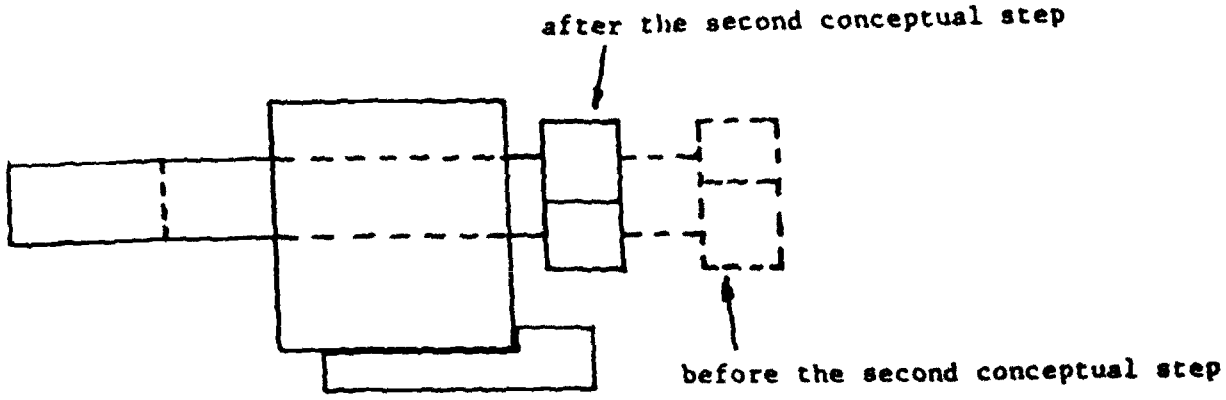
8.3. Linking Rules for Subassemblies

The features of a body in a subassembly keep specified relationships with those of other bodies of the same subassembly. The maintainence of specified relationships between components of a subassembly after an action (including global vision verification) can be considered as to be performed in the following way. At first, the components of the subassembly keep the same relative positions with respect to each other before and after an action, just like the situation discussed for the TIE. Then they possibly make actions along and about the remaining axes of freedom. For instance, in Fig. 8.4 when the block of the subassembly is moved, the shaft may move to any position provided that it still fits the hole of the block. The new position of the shaft can be considered as being obtained by translating and rotating the shaft along and about its axis from the position in which the shaft has the same relative position with respect to the block as before the action.



(a) before a movement

```
subass/perp;
    fits/shaft of a1, hole of block;
tersub;
        ... ...

move/block;
    against/f1 of block, f1 of b1;
    against/f2 of block, f2 of b1;
        ... ...
```



(b) after a movement

Fig. 8.4 Positions of bodies in a subassembly affected by an action

The nature and the amount of the second conceptual step are restricted by the structure of the subassembly, modifying factors of members of the subassembly and the reference bodies, and the relationship statements which specify the destination of the action of the subassembly. To determine the modifying factors of members of a subassembly in the general case is very complex since there are many degrees of freedom existing in the relationship between the members, and sometimes it is impossible since some information is not available at compile time. Thus in this thesis only two cases will be dealt with.

The first case is that the subassembly is moved by an unspecified action. There is no relationship statement given which is relevant to any member of the subassembly after the action. The second case is that the subassembly is moved by a specified action and the relationship statements are only relevant to the member which is moved directly by the associated action statement. In these two cases no information is given about the change of the relative positions among the members of the subassembly so that they can be considered as unchanged, the same as in the case of the TIE. Thus the influence between modifying factors of bodies of a subassembly is the same as that between modifying factors of bodies in a super TIE. When two bodies of a subassembly are TIED together, the TIE does not affect the influence between modifying factors of these two bodies since the effect of the TIE on the influence coincides with the effect implied by the subassembly.

Bodies in a subassembly can also be TIED to other bodies, or even to other subassemblies. The result of this connection is referred to as a super subassembly. For example, the following segment of a program

defines a super subassembly the structure of which is shown in Fig. 8.5.

```
subass/perm;
   against/f1 of b1, f1 of b2;
   fits/f2 of b1, f2 of b3;
tersub;

subass/temp;
   parax/f2 of b5, f3 of b6;
   fits/f1 of b5, f1 of b4;
   against/f3 of b5, f2 of b7;
tersub;
   ... ...
tied/b1, b5;
tied/b4, a1;
 ... ...
```

Obviously, the influence between modifying factors of bodies of a super subassembly is also the same as that between modifying factors of bodies in a super TIE.

In a super subassembly, a permanent subassembly behaves like a permanent sub-super TIE while a temporary subassembly behaves just like an ordinary sub-super TIE. The termination of a temporary subassembly is treated by the framework the same as the revocation of a set of TIES. Thus all the linking rules for the TIE can be applied to the subassembly except one special case.

Bodies of a subassembly can move with respect to each other provided

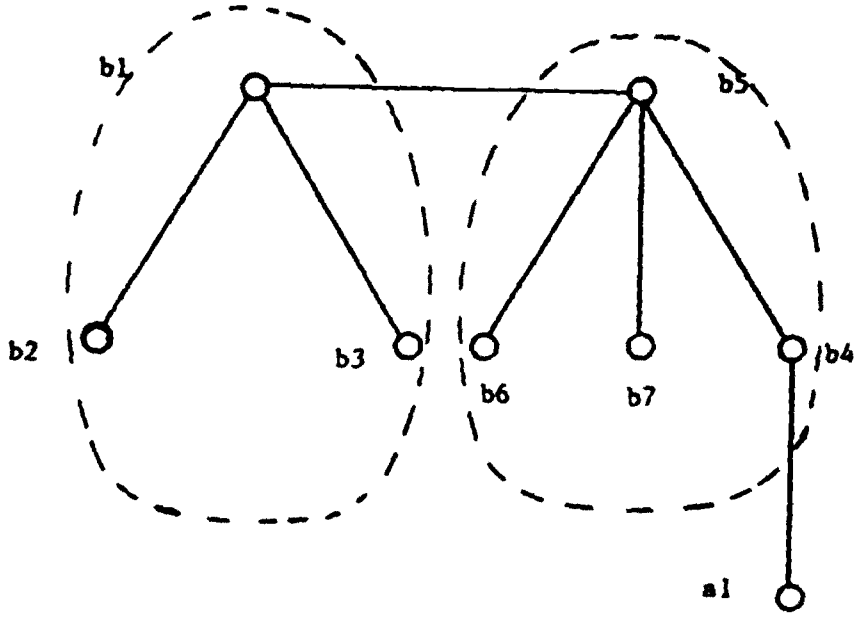


Fig. 8.5. The tree structure of a super subassembly

(b1, b2, b3) is a subassembly,

(b4, b5, b6, b7) is another subassembly

that the specified relationships hold. Thus when a body of a subassembly has been TIED to the world, other bodies can still be moved by either an unspecified action or a specified action under the constraint of the specified relationships. The linking rule for an unspecified action in this case is simple. According to the discussion in Section 8.2.1, when an unspecified action is applied the modifying factor of every body in the super subassembly needs to be the same as that in the previous situation. So does the world. Since the modifying factor of the world before the action is an identity matrix, its modifying factor after the action is still an identity matrix. It is just what it should be. Thus rule (T4) which is suitable for a super subassembly which does not contain the world can still be used in this case.

However, when a specified action is applied to a body of a super subassembly which contains the world, the corresponding rule which can be used to deal with the case in which the super subassembly does not contain the world is no longer suitable. If rule (T1) was used and the modifying factor of the reference body and that of the body being moved were not identity matrices then the actual position of the world might be changed from its nominal one as its modifying factor might not be an identity matrix. This would mean that the world made a virtual motion and would conflict with the definition of the world. If in this case rule (T1) was applied to every body of the super subassembly excluding the world then the relationships between the world and other bodies of the super subassembly which have been TIED would be violated. In order to avoid both cases described above, rule (T4) needs to be applied. In this case the specified relationships between the reference body and the body being moved by the specified action may not be guaranteed if the modifying factors of the two bodies are not the same.

The linking rules for the subassembly can be summarized as follows:

S1. Linking rules (T1)-(T9) are applied to a super subassembly except when the condition mentioned in (S2) is met.

S2. Once a component of a super subassembly has been TIED together to the world and another component is to be moved by a specified action, rule (T4) is applied to the super subassembly.

8.4. The Position of the Camera

A camera which is defined by a general camera specification statement (see Section 5.6) consists of a camera body and a specified focal length. The camera body is an ordinary RAPT body. This means that the general camera can be operated by the RAPT system, being moved, TIED and so on, as any other body. In most cases the position of a camera body can be modified by the modifying factor array according to the linking rules. For example, the position of a general camera can be verified by another camera. When a general camera is moved so that it is fixed to another body, the position of the camera will be modified by the modifying factor of that body.

However, there is one case in which general cameras differ from other bodies. This happens when the user wants to use a general camera to verify the relative position of the camera with respect to another body. Consider the following example in which the user has installed a general camera on the hand of the robot.

... ...

```
fixed/camera,hand;
tied/camera,hand;

move/hand;
   fixed/hand,b1;
   tied/hand,b1;

move/b1;
   fixed/b1,world;

   untied/hand,b1;
move/hand,perpto,f1 of hand, 50;

combine;
   look/f2 of b1,camera;
     ...  ...
tercom;
  ...  ...
```

According to the linking rules listed in Section 8.2, the position of the camera is also subject to modification, since it was TIED, indirectly, to the body to be verified and this TIE, although having been broken, is still effective in determining the linking rules to be applied. However, the verification system has been designed to verify the absolute positions of the bodies in the world. It is obvious that when a camera is used to verify the position of another body, its own position must be certain and should not be changed by the verification. In order to solve this problem, the following rule is applied:

R1. If the result of a global vision command package will affect the position of the camera to be used according to the linking rules listed in Section 8.2, then the vision command package will be dealt with as a local one. This means that only the position of the body to be verified is subject to modification.

In fact, this rule is natural. Cameras are usually well installed so that their positions are less likely to be disturbed than other bodies. Recalling the assumption that the robot moves accurately over a small distance, the example given above will be appropriate when the user thinks that the object has been placed in an unstable position and has possibly moved after it has been let go.

8.5. Simplification Rules

After the linking rules have been applied the modifying factor array is full of pointers or pointer triples. All modifying factors are either pointers or pointer triples except those of body instances of the world, those of body instances in the initial situation and those of body instances which are to be verified. Among these pointers some can be evaluated at compile time. Usually, the majority of body instances in an assembly program have no relationships with vision information. If the modifying factor of this kind of body instance can be found and replaced by the identity matrix symbol at compile time then the structure of the modifying factor array will be simplified and the evaluation of tne array at run time will be speeded up. Furthermore, the evaluation equation of a pointer triple is rather complex. It is also desirable if a pointer triple can be replaced by a pointer. In order to

perform these desired replacements some simplification rules are applied in the simplification phase.

8.5.1. The Simplification of a Pointer Triple

Pointer triples are assigned as modifying factors of body instances by linking rules associated with the TIE and the subassembly. A pointer triple in the modifying factor array represents equation (8.11):

$$FMb(i+1) = FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * * PNa(i+1) * PNb(i+1)^{-1}$$

Here the super TIE effect over the linking rules (see Section 8.2, 8.3) between body a and body b is still valid and body a is either moved by a specified action or verified by global vision commands between situation i and (i+1). A pointer triple can be replaced by a pointer if one of the following two conditions is met.

The first condition is

$$FMa(i+1) = FMai \tag{8.14}$$

In this case, equation (8.11) can be re-written as

$$\begin{aligned} FMb(i+1) &= FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * \\ &\quad * PNa(i+1) * PNb(i+1)^{-1} \\ &= FMbi * PNbi * PNai^{-1} * PNa(i+1) * PNb(i+1)^{-1} \end{aligned} \tag{8.15}$$

Since the relative position of body b with respect to body a is assumed to be TIED between situation i and (i+1), the nominal position of body b in situation (i+1) is

$$PNb(i+1) = PNbi * PNai^{-1} * PNa(i+1) \tag{8.16}$$

Thus

$$PNbi * PNai^{-1} * PNa(i+1) * PNb(i+1)^{-1} = I \tag{8.17}$$

and

$$FMb(i+1) = FMbi \tag{8.18}$$

The pointer triple can therefore be replaced by a pointer which points to FMbi. In the triple [p1, p2, p3] p1 points to FMbi. Thus p1 is assigned as the modifying factor instead of the triple.

The second condition is

$$FMbi = FMai = I \tag{8.19}$$

In this case equation (8.11) can be re-written as

$$\begin{aligned} FMb(i+1) &= FMbi * PNbi * (FMai * PNai)^{-1} * FMa(i+1) * \\ &\quad * PNa(i+1) * PNb(i+1)^{-1} \\ &= PNbi * PNai^{-1} * FMa(i+1) * PNa(i+1) * \\ &\quad * PNb(i+1)^{-1} \end{aligned} \tag{8.20}$$

It can be seen from equation (8.16) that

$$PNb(i+1) * PNa(i+1)^{-1} = PNbi * PNai^{-1} \qquad (8.21)$$

Equation (8.20) can therefore be re-written as

$$FMb(i+1) = PNb(i+1) * PNa(i+1)^{-1} * FMa(i+1) * \\ * PNa(i+1) * PNb(i+1)^{-1} \qquad (8.22)$$

This equation has the same form as equations (7.11) and (8.9). Thus it can be represented by pointer p3 in the triple which points to FMa(i+1).

8.5.2. The Simplification of a Pointer

A pointer can be replaced by an identity matrix symbol if it points to such a symbol. This replacement is obvious for a pointer which points to the modifying factor of the same body in the previous situation. As discussed in Section 8.1, this kind of pointer means the modifying factor equals what the pointer points to. Thus this kind of pointer can be substituted by any thing it points to.

If a pointer points to the modifying factor of another body in the same situation then the pointer needs to be explained by equation (7.11)

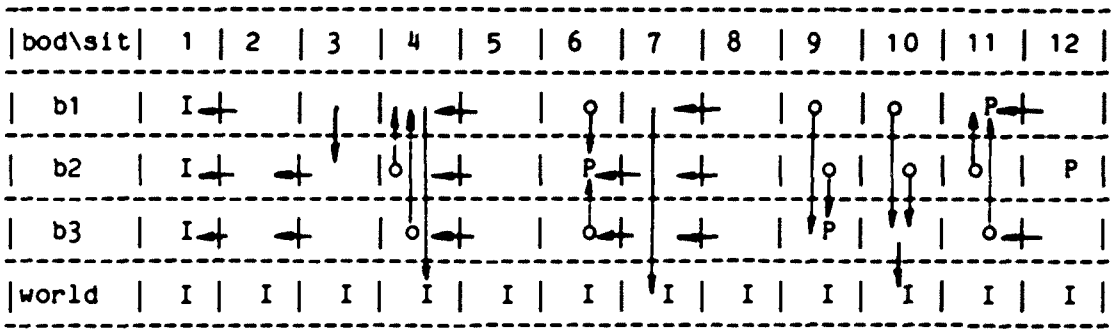
$$FMai = PNai * PNbi^{-1} * FMbi * PNbi * PNai^{-1}$$

It is easy to see that when FMbi is an identity matrix FMai is also an identity matrix.

The simplification rules can be summarized as follows.

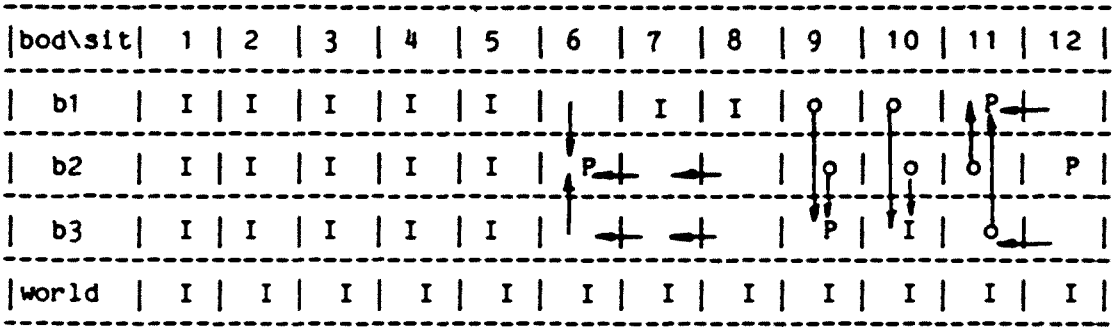
1. If a pointer points to an identity matrix symbol then this pointer is replaced by the symbol.
2. If, in a pointer triple [p1, p2, p3], p1 and p2 point to identity matrix symbols then the pointer triple is replaced by pointer p3.
3. In a pointer triple [p1, p2, p3] if p2 and p3 point to identity matrix symbols or p3 points to p2 then the pointer triple can be replaced by pointer p1.

The following example shows how the rules work. Suppose a modifying factor array before simplification is as follows:



** o--> represents the pointer triple. A triple pointing from ai to bi contains 3 pointers. One points to a(i-1), one to b(i-1) and one to bi.

After the simplification rules have been applied, the array is as follows:


bod\sit
1
2
3
4
5
6
7
8
9
10
11
12
b1
b2
b3
world
I
P

Chapter 9. Implementation and Test

This chapter describes the implementation of the current work and some tests which have been done. All the necessary parts of the system (i.e. vision command input facility (Chapter 5), the symbolic reasoning system (Chapter 6), the framework for handling vision data (Chapter 7 and 8), the window suggester (Section 5.7.1) and the face generator (Section 5.7.3)) have been implemented on the DEC-10 system in Wonder-POP. They are used in conjunction with a normal RAPT system. They have been tested with simulated data and work successfully. The implementation and test procedure are described below.

9.1. Implementation of the Reasoning System

When a RAPT program containing vision commands is read in, the input system parses the RAPT program and transforms the information in the program into RAPT internal data structures. The internal data structures for vision commands are in the form of tables. The information expressed by each type of vision command is stored in a special table indexed by the sequence number of that vision command. The entry for each line of each table includes the situation in which a vision command appears, the body and/or the feature the vision command is concerned with, and a special field for the information which is associated with that type of vision command. At this stage the syntax of vision commands is checked, as well as other things like whether the body mentioned in the INVIOLATE statement is the same as that mentioned in the LOOK statements in the current COMBINE package. The data structures are then passed to the RAPT reasoning system. At first the reasoning system

uses the spatial relationships specified outwith the vision commands to work out the nominal positions of bodies in each situation. These nominal positions are held in the normal way as RAPT body data structures. Then the symbolic reasoning facility is called to deal with the relationships described by the vision commands. This will result in the creation of symbolic expressions which include some variables. The variables represent the positions of the symbolic features created by the LOOK statements, and are indexed by sequence numbers. The symbolic reasoning process also generates the run time commands necessary to operate the cameras. These commands are stored in a list in the same order as that of the corresponding LOOK statements. Each command indicates the situation in which the command is active, the body and the feature to be verified and their nominal positions, the camera to be used and its position and focal length. The framework for handling vision information is called in order to establish the modifying factor array using the linking rules (see Chapter 8). This is the end of the compile time reasoning.

9.2. Implementation of a Run Time Program

Currently, there is a run time program implemented on the DEC-10. This program runs on simulated data and does not have access to a real robot since there is no working robot available in the department at present. The correctness of the symbolic reasoning has been demonstrated with test programs by using this system. A run time system which could be used to control a real robot will be discussed in Chapter 10.

This run time program steps through each camera command generated in the symbolic reasoning. For each command, it uses the model of the camera and the nominal position of the body being verified and its modifying factor (cf Section 7.7) to determine the nominal position of the image of the edge feature to be verified. It then calls the window suggester to obtain a window within which the image of the edge feature should appear. Then simulated data is typed in from the terminal and the face generator is called to generate the face feature implied by the vision data and the model of the camera in use, and the corresponding variable is then instantiated by the position of the face feature. When all the vision data relevant to the camera commands in a situation have been obtained the associated symbolic position equation is evaluated and the result, together with the nominal position of the object being verified, is displayed. Meanwhile, the modifying factor of the body instance being verified is instantiated by a position matrix which indicates the discrepancy between the nominal position and the verified one.

This apparatus has been tested for consistency by typing in data which was the same as the nominal data predicted by the system. Consistency was confirmed by the evaluated result of the symbolic position being the same as the nominal position of the body. This checked the correctness of the implementation of the symbolic reasoning facility, the window suggester and the face generator. Appendix V contains a RAPT program which was used in testing, a sequence of testing operations and the resulting modifying factor array.

9.3. Simulation with ROBMOD

A program has also been written to allow the results of the symbolic reasoning to be displayed. Since there was no working robot available in the department it was decided to simulate the use of verification vision on the ROBMOD system. ROBMOD is a solid modelling system designed and implemented by S. Cameron [CAM84]. It runs under the UNIX operating system and allows solid models of objects to be constructed and their positions in the "world" to be specified. The scene can be viewed from any point, and displayed on a graphics screen. The RAPT system has a subroutine which allows it to produce a file of ROBMOD commands to display the objects in the assembly in each situation. The subroutine has been adapted so that the ROBMOD commands take account of the modifying factors.

After all the simulated vision data had been given to the face generator, the framework instantiated all the elements of the modifying factor array. By the use of the instantiated modifying factors and the nominal positions, the actual position (simulated) of each body in each situation could be generated. The actual positions were then transformed into ROBMOD commands and transferred to the ROBMOD system situation by situation, and ROBMOD displayed each situation on a graphics terminal. The situations displayed show the verified positions of bodies, and the positions which have been modified as a result of the vision data. Observing the position of each body and relationships between bodies in the simulation and comparing them with the RAPT program demonstrated the correctness of the implementation of the framework. Some situations of the simulation of the RAPT program included in Appendix V are shown in figures in that Appendix.

9.4. Implementation of the Edge Finder

The edge finder (Section 5.7.2) has been implemented on a vision system (the Vision Box) [MAT80] in Intel 8080 assembly language. It has been tested with actual scenes and also works successfully. The test procedure is as follows.

The Vision Box takes a picture from a real scene by using the operating system IMAGE [FIS82]. Then the coordinates of three vertices of a window within which the image of an edge feature exists is given to the edge finder which then searches within the window. In order to show the result of the edge finding process, a line is drawn on the image between the end points of the image of the edge feature found by the edge finder. The processing time varies from a tenth of a second to about a second, depending upon the size of the window and the quality of the picture. Figure 9.1 shows an example of the result of the edge finding process.

9.5. Refining Positions of Objects Using Vision Information

An experiment has also been done in order to test the incorporation of the edge finder with all the other components of the verification vision system, and to estimate the accuracy of the result of vision verification. In this experiment a vision station was set up which included a table with a turntable on it, a body on the turntable whose position was to be verified, and a table top TV camera (see Fig. 9.2).

Fig. 9.1. The result of an edge finding process

Fig. 9.2. The vision station used in the experiment

A RAPT program was written to describe this situation. This program was processed by the RAPT system to produce the symbolic position equation of the body to be verified and to suggest the windows to be used by the edge finder. Since there was no hardware link between the Vision Box containing the edge finder and the DEC-10 system containing the other parts of the verification vision system and the RAPT system, the interaction between the edge finder and other parts of the system was done through a human. The human read the coordinates of vertices of a window from a DEC-10 terminal and typed them into the Vision Box, and then called the edge finder. He then read the coordinates of the end points of the image of the edge found by the edge finder from the Vision Box and typed them into the vision system on the DEC-10. After all the required vision data had been typed into the vision system, the symbolic position equation of the body to be verified was evaluated and the verified position displayed.

The experiment was successful in refining positions of the body when it was placed at different positions within the tolerance range. The accuracy of the refined positions is of the same order as that of the calibration of the camera and the placed positions. This experiment demonstrated that vision verification used in this way is useful in updating the robot system's knowledge about the environment. The accuracy of the refined position based upon vision information is mainly dependent upon the calibration of the camera and the resolution of the image. In this experiment, although the accuracy of the calibration is not very high, the accuracy of the refined position is acceptable. More details of the experiment can be found in Appendix VI.

Because of the lack of equipment and time, the system has not been

linked with a real robot in order to control it with real vision data. There will be, of course, some problems in ensuring the correctness of match between the model features and their images, and in obtaining accurate vision information in a complex environment. This problem will be discussed further in Chapter 10. Nevertheless, the concepts and the implementation of the concepts of combining vision verification with an object level robot programming language which have been discussed in this thesis have been shown to be correct.

Chapter 10. Conclusions and Suggestions for Future Work

This thesis has described the achievement of the research on the integration of sensory information into a robot programming language for automatic assembly. The research work contains three major parts. The first is the development of a method to specify a sensory task in an assembly environment in which some prior expectation of the environment is available. The second is the implementation of a symbolic reasoning system which reasons about spatial relationships implied by sensory information off-line before the sensing operation has taken place. The third is the development of a technique to make full use of the sensory information in order to update the robot system's knowledge about the environment. A complete system has been written which uses verification vision information and works successfully in conjunction with the normal RAPT system. In this chapter, the generality and originality of the work and some suggestions of the future work are to be discussed.

10.1. The Generality of the Framework

In the body of the thesis, emphasis has been placed on vision verification. In fact the framework that has been developed has much generality since it is based on the idea of obtaining information about relationships however they may be formed. Generally speaking, in RAPT, when we use a sensor to detect a feature of an object, we create a new relationship between that object and the sensor, and, since the absolute position of the sensor in the world is usually known when the sensor is used, we in effect create a new relationship between that object and the world. The nature of the relationship depends entirely on the type of

the sensor. For example, suppose we use a touch sensor to detect a plane feature p1 of a body B. If we touch a point on p1, we create a relationship AGSP between the world and the body, the "AGAINST" relationship between a sphere feature of the world (a vertex is considered as a sphere with radius 0 in RAPT) and the plane p1. If we detect two points on p1, then we create an AGCP, an against relationship between an edge feature of the world and the plane p1. The position of the edge feature of the world is determined by the positions of the touch sensor when it touches the two points on the plane p1. If we detect three points on the plane p1, then we create a new relationship AGPP between the world and the plane p1 of the body B. The position of the plane is determined by the positions of the touch sensor when it touches the three points on the plane p1. Since sensor data only becomes available at run time, we can only obtain symbolic relationships at compile time. When the symbolic relationships are sufficient to "fix" the object symbolically, we can use a suitable reasoning system to deduce the symbolic position of the object, and then at run time when the corresponding sensor data becomes available we can evaluate the symbolic position expression. In order to introduce a new kind of sensor, we need only to provide a set of new commands to specify how to use the new sensor, and a new set of symbolic reasoning rules which are capable of dealing with the new relationships created by the new sensor. The framework handles new kinds of sensor information in the same way as it does vision data.

Some authors (e.g. [ROS77]) have suggested that vision sensors have poor precision and therefore it is better to use both vision and contact sensors. Vision sensors can be used for coarse sensing while contact sensors can be used for fine resolution. For example, in fitting a shaft in a hole, the information provided by vision can be used to guide

the shaft to a position at which a compliance operation can be executed, and after the operation the robot system's knowledge about the positions of the hole and the shaft can be updated by the more accurate information obtained from this compliance operation. In this case, the generality of the framework to handle both kinds of sensory information is very important.

10.2. Suggestions for Future Work

Although the verification vision system described in this thesis is a complete one, there are some improvements which will make the system more powerful and more convenient to use. These improvements have not been done for two reasons. The first is that although some modification would improve the practicality of the system, they are not theoretically significant. The second reason is that the current state of RAPT (in particular, the RAPT modelling system) makes some improvements impossible. In this section some important potential improvements are listed, and the possible ways of making these improvements are discussed. However, the list is not comprehensive, and the approaches discussed here are tentative rather than fully worked out.

10.2.1. Selecting Suitable Features

When describing a vision task, the programmer needs to select features to be verified carefully so that top level conditionals (see Chapter 6) will not result in special cases but instead the symbolic reasoning system will always apply the general case. Sometimes it is

not easy for a human user to judge whether the features chosen are appropriate or not. This is especially the case when different cameras are used in different LOOK statements in a vision command package. In order to make it easier for the user, the capability of the symbolic reasoning system can be expanded so that the programmer can provide more candidate features to be verified and leave the system to select suitable ones automatically. The extensions needed to do this are discussed below.

Currently the symbolic reasoning system can only recognize some of the top level conditions which cause special cases (such as ROTYLIN) and can then select proper features from any extra candidates that the programmer has provided (see Chapter 6). These recognizable conditions can be examined by investigating relationships between some non-symbolic features so that the symbolic reasoning system can determine the nature of the reasoning at compile time and decide whether to keep the reasoning result or discard it. The current system cannot determine the nature of the reasoning result at compile time for those top level conditions which need to know parameters of symbolic features since the symbolic features are vision information dependent.

However, although the system cannot exactly know the nature of a vision dependent top level conditional, it is possible to estimate whether the result may be out of the general case or not by making use of the information given by the programmer. When the programmer specifies a vision task, he tells the vision system the positions of the cameras to be used, the nominal position of the object to be verified, and the maximum range of the difference between the nominal position and the actual one. The system should be able to use this information to

determine whether the result of a top level conditional might possibly be a special case when the object was at a special position within the range of the specified tolerance. If the answer is "yes" and the programmer has provided some provisional candidate features then the system could examine other combinations of the features until one was found in which no special result could possibly happen. This combination of features would then be chosen as the features to be verified. In this case the programmer would need only to consider whether a selected feature was visible to a selected camera.

10.2.2. Using Complete Models

The current RAPT modelling system is an incomplete one and the user needs only to define the features to be used in the associated RAPT program. Although this modelling system is simple, it does not provide complete information about the environment. Furthermore, the modelling system is only able to describe the surface features of an object and has no knowledge about volume occupancy. The lack of complete information limits the further development of the RAPT system toward higher automatic programming. It also restricts the capability of the verification vision system.

In order to meet the requirement of collision detection and avoidance [CAM82] and the requirement of the automatic planning of an assembly task [KOU82], it is intended that a solid modelling system like NONAME [PEN83] or ROBMOD [CAM84] be used in RAPT. The future verification vision system, therefore, should also take advantage of complete models, and this will enable various improvements to be made. These are

discussed below.

10.2.2.1. Automatically Selecting Features to be Verified

By the use of solid models, the vision system would be able to select features completely automatically. From the solid models the vision system could know how many edge features the object to be verified would have. Using an algorithm similar to the hidden line removal algorithm used in computer graphics (e.g. [FOL82]) the system could decide which feature is visible to a certain camera. Then the system could reason about what combination of features and cameras could avoid causing special cases of top level symbolic conditionals when the object is located at any position within the expected range. Thus the programmer would not need to worry about any details of a vision task. What he would need to do would be to tell the vision system which object would be verified in which situation, and what the position tolerance would be.

Implementation of the ideas talked about above needs much work done on intelligent modules. In the short term, the user can be helped by showing him the scene from the expected viewpoint so that he can select the cameras to be used and the features to be verified more easily.

10.2.2.2. Image-Feature Matching

The information provided by solid models can also be used to detect errors in vision data. It is known that errors may arise when vision

data is obtained and transfered, and these errors may cause the system to malfunction. Some of these errors may come from imperfection of vision hardware and low resolution of the camera. However, the most important error is caused by a mismatch between an object feature and an image. This kind of error is likely to happen especially when there are some similar features near the expected one. Unfortunately, it is commonly the case in vision tasks where from some point the images of some edges are parallel and are very near to each other. Unlike errors caused by imperfection and low resolution of the image which introduce small inaccuracies in the result of the vision verification, errors caused by mismatches between features and images bring significant misunderstanding of the world into the system. Thus this kind of error should be avoided utterly.

The method used in the current vision system to prevent mismatch is to introduce the TOLERANCE statement. It is expected that features to be verified and the cameras to be used are so selected that within the window suggested for the image of the anticipated feature, no images of any other features may appear when the position of the object to be verified is located within the range limited by corresponding TOLERANCE statement. However, this method may not work in some cases when the range of the expected position of the object is too large or there are similar features very near to the expected one.

By the use of solid modelling, an image-feature matching sub-system could be designed to overcome this problem. From solid models, the vision system could know whether there are any similar features whose images may appear in the window when the object wanders under the constraint of the specified tolerance. If the answer was "no" then there

would be no posibility of mismatch for the expected image. If the answer was "yes" then the image-feature matching sub-system would be invoked at run time. The sub-system would make a hypothesis of each possible match of the image and then verify it by investigating expected features at the positions suggested by the hypothesis. If evidences supported one hypothesis and rejected others then the hypothesis being supported would be adopted and the position of the image of the expected feature determined. It would be impossible that more than one hypothesis would be supported simultaneously unless the images of the edge features upon which the hypotheses were made were collinear. However, it would be possible that every hypothesis would be rejected because of the imperfection of the image. In this case, the hypothesis which was supported by most evidence would be considered as the correct one.

10.2.3. Combining Searching and Recognition with Verification

Usually, positions of objects in an assembly task described in an object level language like RAPT are known during operation. There may be some disturbances to these objects in the real world so that the planned position may not be exactly the same as the actual ones. The task of the verification vision system is to detect discrepancies between planned positions and actual ones. Since usually the discrepancies are not too large, the planned positions tell the vision system approximately where the objects may exist. On the basis of this information the verification vision system can refine the position of the object to be verified. However, in some assembly tasks, actual positions of some objects may be far from their actual ones or even unknown

in some cases. For example, when some objects are fed by a cheap belt conveyor the range in which the desired object may be positioned can be very large, much larger than that a verification vision system can tolerate. Thus the capability of the verification vision system is not enough for dealing with this kind of circumstance. A vision system which can search for and locate a desired object in a large range is needed. Furthermore, if different kinds of workpieces are delivered by the same belt conveyor, or different kinds of workpieces are stored in a bin, then the vision system to be used must be able to distinguish them.

By using the information provided by a solid modelling system, a vision system which would be able to search for and recognize objects in a large range could be combined with a verification vision system. In order to deal with three dimensional objects in the real world rather than their silhouette image, three dimensional vision techniques must be used. Some candidates of the method to be used might be a syntactic approach which has been discussed by Luh and Yam [LUH81], or a hypothesis-verification approach described by Fisher [FIS83]. When the position of an object which might be far from the planned place was to be found, the vision system might be asked to search and locate it first. If the accuracy of the position determined in this step was not high enough then the vision system could determine inviolate relationships between the object and the world by comparing the approximate position with possible stable states of the objects, and select features to be verified and cameras which could produce a more accurate result automatically, and then invoke the verification function of the vision system.

This powerful vision system would be easily connected with the RAPT

system via the framework discussed in Chapter 7. At compile time, the corresponding modifying factor would be assigned a symbol to indicate that a position would be assigned at run time. If the planned position could not be determined then it could be assigned arbitrarily. For example, it could be assigned an identity matrix. Since the modifying factor indicates the difference between a planned position and the actual one, if the vision system located the object correctly and accurately, it would not matter what the planned object position was.

10.2.4. More Types of Inviolate Relationships

At the moment, there is only one type of relationship that can be used in an INVIOLATE statement. It is enough to demonstrate the basic idea and show the usefulness of the inviolate relationship in describing a vision task. However, as a practical system, more types of inviolate relationships may be desired in order specify a vision task conveniently. The types of new inviolate relationships which are worth being considered are FITS and ROT. These two types of relationships have been discussed in Chapter 3. They might be the second most common types of inviolate relationships encountered in a vision task. FITS has two degrees of freedom: a translational one and a rotational one while ROT has one degree of rotational freedom. The approach to the introduction of these new types of inviolate relationships would be the same as that of AGPP. It would only be necessary to add some new functions into the symbolic reasoning system so that symbolic relationship cycles which would contain these types of relationships could be inferred. As both types of relationships contain one degree of rotational freedom, most results of reasoning would be ambiguous. There would usually be two

possible solutions which could satisfy a relationship cycle. In order to resolve the ambiguity, a method which would be similar to that used in the current system for dealing with the ambiguity in reasoning among AGPP and AGPE (see Chapter 6) could be adopted: the possible results would be compared with the planned position and the one with less discrepancy would be considered as the correct one.

There would be one difficulty in introducing these new types of inviolate relationships. There would be more conditions which would lead to special cases of top level symbolic conditionals (cf [POP81]) so that it would be more difficult for a human user to select features to be verified and cameras to be used properly. Thus it would be better to implement this work together with those suggested in Sections 10.2.1 and 10.2.2.

10.2.5. Run Time System for Robot Control

A run time system needs to be written which connects a vision system directly to the robot operating system in order to instantiate the modifying factor array by using vision data and then to control a real robot using the output of the RAPT system and the instantiated modifying factor array. The run time system will work in a way similar to the run time program described in Sections 9.2 and 9.3 except that it will communicate with the vision system directly and control the robot rather than a modelling system. During execution, the run time system commands robot operations situation by situation by using nominal positions of the robot arm in each situation and its modifying factor except in situations in which vision commands need to be run. In situations in

which vision tasks are to be performed, the run time system sends vision commands to operate the vision system, and then receives the vision data from the vision system in order to evaluate symbolic position expressions and to instantiate relevant modifying factors. To this end, interfaces between the host machine on which the run time system resides and the robot and the vision system need to be developed.

The reasoning facility which is needed in the run time system is much less than that of the compile time reasoning system (the cycle finder). The run time system only needs the code particularly involved in evaluating the top level functions contained in the symbolic position expressions and does not need the control of the reasoning which is a major part of the compile time reasoning system.

10.3. Originality

Reasoning about spatial relationships is the main characteristic of the RAPT system. The research work described in this thesis relied heavily on this characteristic. The concept of verification vision for programmable assembly was suggested by R. C. Bolles [BOL77]. The use of sensory information in robot programming languages has already been reported. The main originality of the research work discussed in this thesis lies in the way in which the use of the sensory information is incorporated with a high level robot command language intelligently.

As described in Chapter 2, some work on using vision data in robot programming languages has already been reported, such as VAL [UNI79, UNI80] and AL [GOL77]. However, these systems combine vision facilities

with end-effector level languages only and the techniques used are of the first generation [LOU81] in that they use a binary picture and only provide a very restricted form of mapping from the two dimensional image to the location of a three dimensional object. Vision is only used to locate some individual objects or to check whether an operation has been fulfilled or not, rather than to update the knowledge of the robot system about the environment. There has been no report on combining vision with an object level robot programming language. Some object level languages have the ability to use some touch sensor information. For example, in AUTOPASS [WIL79] and LAMA [LOZ76] force or torque sensor information is used to provide special threshold values in order to constrain or terminate some actions, or to detect whether a specified action terminates at a correct position or not. In both the systems, sensory information is used as a condition in a decision tree and no further explanation of the information is made.

In contrast to these systems, the work described in this thesis allows the use of sensory data in a general way. It exploits the expectation about spatial relationships in order to make the best use of the sensory data without having to place restrictions upon the relative positions of objects and sensors. A method is provided of specifying vision tasks and reasoning about the vision data symbolically off-line. It allows partial information about positions to be combined with sensor information in a general way. When the vision data becomes available at run time, the result of the symbolic reasoning is evaluated and is used to improve the system's knowledge about the environment. This method uses grey level pictures and produces a three dimensional interpretation rather than a two dimensional one. The framework which is used by the system for handling vision data makes it possible to use the vision

information to adapt the operation of the robot in a variety of flexible ways. The method developed to handle vision data has some generality and can also deal with some other kinds of sensory information, such as that from tactile sensors. This has already been discussed in Section 10.1. All of these enhance the capability of normal RAPT which has so far had no facilities to manipulate sensory information.

Bolles's work concentrated mainly on achieving sufficient confidence of correct correspondence between object features and their image, adequate precision of location, and sufficiently low cost of achieving required confidence and precision in order to refine the position of the object to be verified. In contrast to this, the work described in this thesis combines verification vision with a high level robot programming language which reasons about spatial relationships. It concentrates on making intelligent use of vision information to update the robot system's knowledge about the environment rather than to refine the position of the object being verified only.

10.4. Significance

This thesis tackled two problems. The first was how to enhance a high level robot programming language so that it can express vision commands to locate workpieces of an assembly. The second was how to find a way of making full use of sensory information to update the knowledge of the robot system about its environment. The introduction of verification vision into RAPT as described in this thesis has successfully solved these two problems. This is the first instance of model-based vision being used in an object level robot programming language to deal with positional errors in the real environment.

References

[AGI77] G. J. Agin

Servoing with Visual Feedback

SRI International Technical Note 149, 1977.

[AGI80] G. J. Agin

Computer Vison Systems for Industrial Inspection and Assembly

Computer Vol. 13, No. 5, 1980, pp11-20.

[AMB75a] A. P. Ambler et al.

A Versatile System for Computer Controlled Assembly

Artificial Intelligence 6. 1975, pp129-156.

[AMB75b] A. P. Ambler, R. J. Popplestone

Inferring the Positions of Bodies From Specified Spatial Relationships

Artificial Intelligence 6. 1975. pp157-174.

[AMB82] A. P. Ambler, R. J. Beattie, D. F. Corner

RAPT1 User's Manual

Dept. of Artificial Intelligence, Univ. of Edinburgh, 1982.

[AMB83] Ambler,A.P., Corner, D.F. & Popplestone,R.J.

Reasoning About the Spatial Relationships Derived from a

RAPT Program for Describing Assembly by Robot

IJCAI-83, Karlsruhe, West Germany. 1983, pp842-844.

[BAL82] D. A. Ballard, C. M. Brown

Computer Vision

Prentice-Hall Inc. 1982.

[BAR76] H. G. Barrow, R. M. Burstall

Subgraph Isomorphism Matching Relational Structures and Maximal Cliques

Information Processing Lett. Vol. 4, No. 4, pp83-84, 1976

[BAR78] H. G. Barrow, J. M. Tenenbaum

Recovering Intrinsic Scene Characteristics From Images

Computer Vision Systems (edited by A. R. Hanson, E. M. Riseman), Academic Press, 1978, pp3-26.

[BAR81] H. G. Barrow, J. M. Tenenbaum

Computational Vision

Proceedings of the IEEE. Vol. 69, No. 5, 1981 pp572-595.

[BAU74] B. G. Baumgart

Geometric Modeling for Computer Vision

Stanford AI Lab. Memo. AIM-249, Computer Science Department, Stanford University, 1974.

[BAU81] E. W. Baumann

Model-Based Vision and the MCL language

The 1981 International Conference on Cybernetics and Society,

Oct. 1981, Atlanta, Georgia, USA.

[BEA82] R. Beattie

Edge Detection for Semantically Based Early Visual Processing

Proc. of 1982 European Conference on Artificial Intelligence, Orsay, France, 1982, pp190-196.

[BEJ79] A. K. Bejczy

Manipulator Control Automation Using Smart Sensors

Electro/79 Convention, New York NY. April 24-26, 1979. p16.

[BIN82] T. O. Binford

Survey of Model-Based Image Analysis System

The International Journal of Robotics Research, Vol. 1, No. 1, 1982, pp18-64.

[BOL75] R. C. Bolles

Verification Vision Within a Programmable Assembly System

Stanford AI. Lab Memo AIM-275 1975.

[BOL77] R.C. Bolles

Verification Vision for Programmable Assembly

5th IJCAI. 1977

[BOL80] R. C. Bolles

Robust Feature Matching Through Maximal Cliques

SRI International Tech. Notes 212, 1980.

[BOL81a] R. C. Bolles

Three-Dimensional Locating of Industrial Parts

SRI International Tech. Note 234, 1981.

[BOL81b] R. C. Bolles

An Overview of Applications of Image Understanding to Industrial Automation

SRI International Tech. Notes 242, 1981

[BOL82] R. C. Bolles, R. A. Cain

Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method

Robotics Research Vol. 1, No. 3, 1982, pp57-82.

[BRO81] R.A. Brooks

Symbolic Reasoning Among 3-D Models and 2-D Images

Artificial Intelligence Vol. 17. 1981

[BRO82] R.A. Brooks

Symbolic Error Analysis and Robot Planning

Robotics Research. Vol. 1, No. 4, 1982, pp29-68.

[BUR77] R. M. Burstall et al.

Programming in POP-2

Edinburgh University Press. 1977.

[CAM82] S. Cameron

The Clash Detection Problem

DAI Working Paper 126, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1982.

[CAM84] S. Cameron

Modelling Solids in Motion

Ph.D Thesis, Department of Artificial Intelligence, University of Edinburgh, 1984.

[CAN83] J. F. Canny

Finding Edges and Lines in Images

MIT AI Lab. Technical Report 720, 1983.

[CLO82] W. F. Clocksin

Visually Guided Robot Arc-Welding of Thin Sheet Steel Pressing

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France, pp225-230.

[COR84] D. F. Corner

A Simple Addition to the RAPT Inference System to Handle Taught Positions

DAI Working Paper, University of Edinburgh, 1984.

[CRO82] A. J. Cronshaw

Automatic Chocolate Decoration by Robot Vision

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France, pp249-257.

[DAR75] J. A. Darringer, M. W. Blasgen

MAPLE: A High Level Language for Research in Mechanical Assembly

IBM Research Report RC-5606, 1975.

[DAV75] L. S. Davis

A Survey of Edge Detection Techniques

Computer Graphics and Image Processing 4, 1975, pp248-270.

[DAV78] L. S. Davis, A. Rosenfeld

Hi erarchical Relaxation for Waveform Parsing

Computer Vision Systems (edited by A. R. Hanson, E. M. Riseman), Academic Press, 1978, pp101-109.

[DUD73] R.O. Duda, P.E. Hart

Pattern Classification and Scene Analysis

John Wiley & Sons, Inc. 1973

[ENG80] J. F. Engelberger

Robotics in Practice

IFS Publications Ltd. 1980.

[ERN61] H. A. Ernst

A Computer-Operated Mechanical Hand

Sc.D. Thesis, MIT, 1961.

[FAL80] D. Falek & M. Parent

An Evalutionary Language for an Intelligent Robot

The Industrial Robot, Vol. 7 No.3. pp168-171 (1980)

[FAN82] O. D. Fangeras, et al.

Toward a Flexible Vision System

Proc. of 12th International Symposium on Industrial Roboots, June, 1982. Paris, France.

[FEL71] J. Feldman et al.

The Use of Vision and Manipulation to Solve the Instant Insanity Puzzle

Proc. of 2nd IJCAI. London, England, 1971, pp395-364.

[FIN75] R. Finbel et al.

An Overview of AL: a Programming System for Automation

Proc. of the 4th IJCAI, Tabilisi, Georgia, USSR, 1975, pp758-765

[FIS82] R. B. Fisher

IMAGE User's Guide

Department of Artificial Intelligence, University of Edinburgh, 1982.

[FIS83] R. B. Fisher

Using Surfaces and Object Models to Recognize Partially Obscured Objects

Proc. of the 8th IJCAI, Karlsruhe, West Germany, 1983, pp989-995.

[FLE82] J. Fleck, et al.

The Development and Diffusion of Industrial Robots

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France.

[FOL82] J. D. Foley, A. ven Dam

Fundamentals of Interactive Computer Graphics

Addison-Wesley Publishing Company, 1982.

[FOR69] A. R. Forrest

Co-ordinates, Transformations and Visualisation Techniques

Cambridge Computer-Aided Design Group, 1969.

[GES83] C. C. Geschke

A System for Programming and Controlling Sensory-Based Robot Manipulators

IEEE Trans. on Pattern Analysis & Machine Intelligence, Vol. PAMI-5, No. 1, 1983, pp1-7.

[GOL77] R. Goldman

Recent Work with AL System

Proc. of the 5th IJCAI, MIT, USA. 1977, pp733-735.

[GRE70] R. L. Gregory

The Intelligent Eye

Weidenfeld & Nicolson Ltd. 1970.

[HAN78] A. R. Hanson, E. M. Riseman (ed)

Computer Vision Systems

Academic Press, New York, 1978, ppXV-XXVII.

[HAR80] L. D. Harmon

Touch-Sensing Technology

Dept. of Biomedical Engineering, Case Western Reserve University

Cleveland, Ohio 44106

[HAR81] J. F. Harris

An Overview of Robot Vision R and D in UK and Abroad

Image Analysis Group, University of Oxford, 1981.

[HIR82] G. Hirzinger

Force Feedback Problems in Robotics

Second IASTED DAVOS International Symposia, Davos, March 2-5, 1982.

[HOG83] D. Hogg

Model-Based Vision: a Program to See a Walking Person

Image and Vision Computing Vol. 1 No. 1, Feb. 1983, pp5-20.

[HOR73] B. K. P. Horn

The Binford-Horn Lind Finder

MIT. AI Memo. 285, 1973.

[HOR75] B. K. P. Horn

Obtaining Shape from Shading Information

The Psychology of Computer Vision, pp115-155, (ed. by Winston), 1975.

[INO74] H. Inoue

Force Feedback in Precise Assembly Tasks

MIT. AI Lab. Memo. 308, 1974.

[JAP82]

The Specifications and Applications of Industrial Robots in Japan

Japan Industrial Robot Association, 1982.

[KAN77] Takeo Kanade

Model Representations and Control Structures in Image Understanding

Proc. of the 5th IJCAI, MIT, USA. 1977, pp1074-1082.

[KAS77] H. Kasdan

Pattern Recognition in Quality Assurance

Industrial Research, August, 1977, pp49-52.

[KEL81] R. B. Kelley

Recent Progress in Bin Picking

Combined Seminar & Poster Session on Pattern Recognition for Robots, University College, London. March 1981.

[KEM81] K. G. Kempf

Robot Command Languages and Artificial Intelligence

AI Group, Electronics Research Laboratory, Hirst Research Center, General Electric Company Ltd. Wembly, Middlesex, England, 1981.

[KEM83] K. G. Kempf

Artificial Intelligence Applications in Robotics - a Tutorial

IJCAI-83 Tutorial on Artificial Intelligence, Karlsruhe, West Germany, 1983.

[KLE78] J. de Kleer, G. J. Sussman

Propagation of Constraints Applied to Circuit Synthesis

Memo AIM 485, MIT, Cambridge 1978.

[KOU81] A. Koutsou

A Survey of Model-Based Robot Programming Languages

DAI. Working Paper No. 108, University of Edinburgh, Dec. 1981.

[KOU82] A. Koutsou

Thesis Proposal: Planning Parts Mating Operations

DAI Working Paper 114, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1982.

[LAT79] J. C. Latombe

Une Analuse Structuree d'Outils de Programmation pour la Robotique Industrielle

Proc. of the International Seminar on Programming Methods and Languages for Industrial Robots. IRIA, France, June, 1979

[LAT81] J. C. Latombe, E. Mazer

LM: A High-Level Programming Language for Controlling Assembly Robots

Tripartite Robotics Seminar, Stuttgart, W. Germany, Feb. 1981.

[KEM83a] D. R. Kemp et al.

A Sensory Gripper for Handling Textiles

Proc. of 13th International Symposium on Industrial Robots, 1983. Chicago, USA.

[LIE77] L. I. Lieberman, M. A. Wesley

AUTOPASS: An Automatic Programing System for Computer Controlled Mechanical Assembly

IBM Journal of Research and Development, Vol. 21, No. 4, 1977,

pp321-323.

[LIN73] P. M. Lin

A Survey of Applications of Symbolic Network Functions

IEEE Trans. on Circuit Theory Vol. CT-20 No.6, 1973, pp732-737

[LOU81] C. Loughlin

Robot Vision Survey

Robotics Research Unit, Dept. of Electronic Eng. University of Hull, 1981.

[LOZ76] T. Lozano-Perez

The Design of a Mechnical Assembly System

MIT AI. Lab. Technical Report 397. Dec. 1976.

[LOZ82] T. Lozano-Perez

Robot Programming

MIT AI. Lab. AI Memo. 698, Dec. 1982.

[LUH81] J. Y. S. Luh et al.

3-D Vision for Robotic Systems.

Proc. of the 1st International Conference on Robot Vision and Sensory Controls, April 1-3, 1981, Stratford upon Avon, UK.

[MAT80]

RGB-256 Manual

Matrox Electronic Systems Ltd., 5800 Andover Ave., T.M.R. Qug., H4T 1H4, Canada, 1980.

[MCG78] D. McGhie, J. W. Hill

Vision-Controlled Subassembly Station

AI. Center, SRI International, Menlo Park, California 94025. Oct. 1978.

[NIT76] D. Nitzan, A. E. Brain, R. O. Duda

The Measurement and Use of Registered Reflectance and Range Data in Scene Analysis

SRI International AI Center Technical Note 128. Menlo Park, California 94025. 1976

[NIT79] D. Nitzan

Robotic Sensors in Programming Automation

SRI International Technical Note 183. Menlo Park, California 94025. 1979.

[PAU77] R. P. Paul

WAVE: A Model-Based Language for Manipulator Control

The Industrial Robot, Vol. 4, No. 1, 1977, pp10-17

[PAU81] R. P. Paul

Robot Manipulators: Mathematics, Programming and Control

MIT Press, 1981.

[PEN83] A. de Pennington, M. S. Bloor, M. Balila

Geometric Modelling: A Contribution Towards Intelligent Robots

Proc. of 13th ISIR/Robots-7, Chicago, 1983, pp35-54.

[PER78] W. A. Perkins

A Model-Based Vision for Industrial Parts

IEEE Trans. on Computers Vol C-27, No. 2 Feb. 1978.

[POP77] R. J. Popplestone, A. P. Ambler

Forming Body Models From Range Data

DAI. Research Paper, No.46, University of Edinburgh, 1977

[POP78] R. J. Popplestone, A. P. Ambler, I. Bellos

RAPT: A Language for Describing Assemblies

DAI. Research Paper, No.79, University of Edinburgh, 1978

[POP79] R. J. Popplestone

Specifying Manipulation in Terms of Spatial Relationships

DAI. Research Paper. No.117, University of Edinburgh, 1979

[POP80] R. J. Popplestone, A. P. Ambler, I. M. Bellos

An Interpreter for a Language for Describing Assemblies

DAI. Research Paper. No.125, University of Edinburgh, 1980

[POP81] R. J. Popplestone, A.P. Ambler

A Language for Specifying Robot Manipulations

DAI. Research Paper. No.161, University of Edinburgh, 1981

[PUG82] A. Pugh

Second Generation Robotics

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France.

[RAE81] R. Rae

Wonder-POP: A Brief Guide to the Main Features of the New POP-2 Implementation

DAI. University of Edinburgh. 1981.

[ROS69] A. Rosenfeld

Picture Processing by Computer

Computing Surveys, Vol. 1, No. 3, Sept. 1969, pp147-176.

[ROS77] C. A. Rosen, D. Nitzan

Use of Sensors in Programmable Automation

Computer. Vol. 10, No. 12, Dec. 1977, p12-23.

[SAR81] P. Saraga, B. M. Jones

Parallel Projection Optics in Simple Assembly

Proc. of the 1st International Conference on Robot Vision and Sensory Controls, April 1-3, 1981, Stratford upon Avon, UK, pp99-112.

[TAY76] R. H. Taylor

A Synthesis of Manipulator Control Program from Task-Level Specifications

Stanford AI Lab. Memo. AIM-282, 1976.

[TAY82a] P. M. Taylor et al.

Closed Loop Control of an Industrial Robot Using Visual Feedback From a Sensory Gripper

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France.

[TAY82b] G. E. Taylor et al.

Vision Applied to the Orientation of Embroidered Motifs in the Textile Industry

Proc. of 2nd International Conference on Robot Vision and Sensory Controls, Stuttgart, West Germany, Oct. 1982.

[TEN77] J. M. Tenenbaum, H. G. Barrow

Experiments in Interpretation-Guided segmentation

Artificial Intelligence Vol. 8, No. 3, 1977, pp241-274.

[UNI79] Unimation, Inc.

User's Guide to VAL: A Robot Programming and Control System

Unimation, Inc., Danbury, Connecticut, Feb. 1979.

[UNI80] Unimation, Inc.

Unimation Vision Manual

Unimation, Inc., Danbury, Connecticut

[VIL82] P. Villers

Present Industrial Use of Vision Sensor of Robot Guidance

Proc. of 12th International Symposium on Industrial Robots, June, 1982. Paris, France.

[WAN76] S. S. M. Wang, P. M. Will

Sensors for Computer Controlled Mechanical Assembly

Computer Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, 1976.

[WIL75] P. M. Will, D. Grossman

An Experimental System for Computer Controlled Mechanical Assembly

IEEE Trans. on Computers Vol. C-24, No. 9, 1975, pp879-888.

[WIL79] P. M. Will

Very High Level Languages for Robots

Proc. of the International Seminar on Programming Methods & Languages for Industrial Robots, IRIA, Rocquencourt, France. June, 1979.

[WON79] Wong

Computational Structures for Extracting Edge Features from Digital Images for Real Time Control Applications

Ph.D Thesis, Col. Tech. 1979.

[YIN83] B. Yin

A Framework for Handling Vision Data in an Object Level Robot Language --- RAPT

Proc. of the 8th IJCAI, Karlsruhe, West Germany, 1983, pp814-820.

[ZUC78] W. Zucker

Vertical and Horizontal Processes in Low Level Vision

Computer Vision Systems (edited by A. R. Hanson, E. M. Riseman), Academic Press, 1978, pp187-198.

Appendix I

Tables of Reasoning Rules in the RAPT Reasoning System

Degrees of Freedom of the Relations in the Cycle Finder

Relations		Feature Types		Degrees of freedom	
				Spatial	Rotational
AGPP	Against	plane	plane	2	1
FITS	Fits	shaft	hole	1	1
AGPC	Against	plane	shaft	2	2
AGPS	Against	plane	sphere	2	3
AGSS	Against	sphere	sphere	0	3
PARAX	Parallel	plane	plane	3	1
LIN	Linear	shaft	shaft	1	0
LINLIN	Planar	plane	plane	2	0
ROT	Rotation	shaft	shaft	0	1
FIX	Fixed	any	any	0	0

Table 1. Combination Rules

Relation 1	Relation 2	General Case	Special Cases
LIN	LIN	FIX	LIN
LINLIN	LINLIN	LIN	LINLIN
LINLIN	LIN	FIX	LIN
ROT	ROT	FIX	ROT
ROT	LINLIN	FIX	
ROT	LIN	FIX	
FITS	FITS	FIX	FITS,LIN
FITS	ROT	FIX	ROT
FITS	LINLIN	FIX	LIN
FITS	LIN	FIX	LIN
AGPP	AGPP	LIN	AGPP
AGPP	FITS	FIX	LIN,ROT
AGPP	ROT	FIX	ROT
AGPP	LINLIN	LIN	LINLIN
AGPP	LIN	FIX	LIN
AGPP	AGPC	LIN (2)	AGPP,ROT+LIN
PARAX	PARAX	-	PARAX
PARAX	AGPP	LINLIN	AGPP
PARAX	FITS	LIN	FITS
PARAX	ROT	FIX	ROT
PARAX	LINLIN	LINLIN	
PARAX	LIN	LIN	
AGSS	AGSS	-	ROT,AGSS
AGSS	PARAX	-	ROT
AGSS	AGPP	-	ROT
AGSS	FITS	-	ROT,FIX
AGSS	ROT	FIX (2)	ROT,FIX
AGSS	LINLIN	-	FIX
AGSS	LIN	FIX (2)	FIX
AGPS	AGPS	-	
AGPS	AGSS	-	ROT
AGPS	PARAX	-	AGPP
AGPS	AGPP	-	AGPP
AGPS	FITS	-	ROT,LIN
AGPS	ROT	FIX (2)	ROT,FIX
AGPS	LINLIN	LIN (2)	LINLIN
AGPS	LIN	FIX	LIN
AGPC	AGPC	-	AGPP
AGPC	AGPS	-	AGPP
AGPC	AGSS	-	ROT
AGPC	PARAX	LINLIN (2)	ROT,FIX
AGPC	FITS	FIX (2)	FITS,ROT,LIN
AGPC	ROT	FIX (2)	ROT,FIX
AGPC	LINLIN	LIN	LINLIN
AGPC	LIN	FIX	LIN

This table shows how pairs of relations (relation 1 and relation 2) can be combined to produce a new relation. The resulting relation is

given for the general case and also for some special cases where the resulting relations are more restricted. The special cases arise from particular geometrical relationships between features involved. In this table, "-" means the result is not included in the ten standard relations. "(2)" means the result is not unique.

Table 2. Creation Rules

Relation 1	Relation 2	General Relation	Special Cases
LIN	LIN	LINLIN	LIN
LINLIN	LINLIN	PARAX*	LINLIN
LINLIN	LIN	PARAX*	LINLIN
ROT	ROT	-	ROT
ROT	LINLIN	PARAX*	AGPP
ROT	LIN	PARAX*	FITS
FITS	FITS	-	FITS,PARAX*
FITS	ROT	-	FITS,AGPC,PARAX*
FITS	LINLIN	-	PARAX
FITS	LIN	-	FITS,PARAX*
AGPP	AGPP	-	AGPP
AGPP	FITS	-	PARAX,AGPC
AGPP	ROT	-	AGPP
AGPP	LINLIN	PARAX	AGPP
AGPP	LIN	PARAX	AGPP
PARAX	PARAX	-	PARAX
PARAX	FITS	-	PARAX
PARAX	ROT	-	PARAX
PARAX	LINLIN	PARAX	
PARAX	LIN	PARAX	
AGSS	AGSS	-	AGSS
AGSS	AGPP	-	AGPS
AGSS	ROT	-	AGSS
AGPS	AGSS	-	AGPS
AGPS	AGPP	-	AGPS
AGPS	ROT	-	AGPS
AGPS	LINLIN	-	AGPS
AGPS	LIN	-	AGPS
AGPC	AGSS	-	AGPS
AGPC	AGPP	-	AGPC
AGPC	FITS	-	AGPC
AGPC	ROT	-	AGPC
AGPC	LIN	-	AGPC

Table 2 shows a chained pair of relations (relation 1 and relation

2) may be replaced by a new relation. The resulting relation is given for the general case and also for some special cases where the resulting relations are more restricted. The special cases arise from particular geometrical relationships between features involved. Since the RAPT reasoning system only applies Table 2 when it can obtain a useful result, not all possible pairs of relations need to be included. In this table, the relations marked with "*" are not the true combinations for the input relations, because the true answers are not expressible in our limited set of relations. A relation with additional degrees of freedom is used instead.

Appendix II

Detailed Analysis of Combination Rules Likely to be Used in the Symbolic Reasoning

The following table is an extract from a document produced by Tamio Arai (personal communication). It shows all possible special cases in combinations of relations under different conditions. The entries are relevant to pairs of relationships occurring in the symbolic reasoning described in the thesis.

R1	R2	RR	Condition
AGPP	AGPC	AGPP	x_a_par
		ROTYLIN	x_b_par
		LIN(2)	x_b_perp
		LIN(2)	general
LIN	AGPC	LIN	x_a_perp
		FIX	x_a_par
		FIX	general
ROTYLIN	AGPC	ROTYLIN	x-a-par
		ROTYLIN	x_b_collinear & is_0_a12
		ROT	x_b_collinear
		LIN(2)	is_0_a12 x_b_par
		LIN(2)	is_0_a12 & x_a_perp & r_xa1_p1a2
		LIN	is_0_a12
		function	x_b_par
		FIX(2)	x_b_perp
		FIX(2)	general
ROT	AGPC	ROT	x_a_par
		ROT	x_b_collinear
		FIX	x_a_perp
		FIX(2)	general

NOTE: This table has not been completely implemented in the current version of RAPT, because the current cycle finder is restricted

to standard relations and non-ambiguous result.

R1: the first relation

R2: the second relation

RR: the resultant relation

The order of the features in the relations between the two bodies is different from that in TABLE 1 in [POP81]. Here if we refer to the bodies involved in the relationships as body A and body B, then the first feature in the relationship always belongs to body A and the second to body B. For example, the combination of AGPP_AGPC means two planes belong to the body A and one plane and a edge belong to the body B.

"(2)" means there are two possible ways to create the new feature, i.e., there is ambiguity.

Condition: the condition under which the rewrite rule can be used

x_a_par: X-axes of features in body A are parallel

x_b_par: X-axes of features in body B are parallel

x_a_perp: X-axes of features in body A are perpendicular

x_b_perp: X-axes of features in body B are perpendicular

is_0_a12: the angle between x-vector of feature 2 of body A and y-vector of feature 1 of body A is 90 degree

r_xa1_p1a2: a complex function defined in Wonder-POP code.

Appendix III

A Sufficient Condition for Correct Modifying Factor Determination when There Is More Than One Reference Body

When a body a is moved by an action statement to a specified position to satisfy a set of spatial relationships between it and two reference bodies b and c, a sufficient condition under which the specified relationships can be realized in the real environment when the modifying factors are taken into account is

$$PVbi * PVci^{-1} = PNbi * PNci^{-1}$$

where situation i is the situation following the specified action. This can be proved as follows.

According to equation (7.11), the modifying factor of body a should be

$$FMai = PNai * PNbi^{-1} * PVbi * PNai^{-1}$$

in order to realize the relationship between body a and body b. For the same reason, the modifying factor of body a should also be

$$FMai' = PNai * PNci^{-1} * PVci * PNai^{-1}$$

in order to realize the relationship between body a and body c.

When

$$PVbi * PVci^{-1} = PNbi * PNci^{-1}$$

we have

$$PNci^{-1} * PVci = PNbi^{-1} * PVbi$$

Thus

$$\begin{aligned} FMai' &= PNai * PNci^{-1} * PVci * PNai^{-1} \\ &= PNai * PNbi^{-1} * PVbi * PNai^{-1} \\ &= FMai \end{aligned}$$

Therefore, the modifying factor of body a which guarantees the realization of the relationship between body a and body b will also guarantee the realization of the relationship between body a and body c.

Appendix IV

A Sufficient Condition for Correct Modifying Factor Determination when There Is More Than One Reference Body in a TIE

When the spatial relationships associated with a specified action refer to both body a and body b in a TIE, a sufficient condition under which the specified relationships can be realized in the real environment when the modifying factors are taken into account is

$$PVbi * PVai^{-1} = PNbi * PNai^{-1}$$

where situation i is the situation preceding the specified action. This can be proved as follows.

Since there is a relationship to hold between the reference body c and both body a and body b, the modifying factors of both body a and body b after the specified action can be determined individually by equation (7.11):

$$FMa(i+1) = PNa(i+1) * PNc(i+1)^{-1} * PVc(i+1) * PNa(i+1)^{-1}$$
$$FMb(i+1) = PNb(i+1) * PNc(i+1)^{-1} * PVc(i+1) * PNb(i+1)^{-1}$$

On the other hand, since body a and body b are TIED together, from equation (8.11) the modifying factor of body b should also be

$$FMb(i+1)' = PVbi * PVai^{-1} * PVa(i+1) * PNb(i+1)^{-1}$$

$$= PVbi * PVai^{-1} * FMa(i+1) * PNa(i+1) * PNb(i+1)^{-1}$$

$$= PVbi * PVai^{-1} * PNa(i+1) * PNc(i+1)^{-1} * PVc(i+1) *$$

$$* PNa(i+1)^{-1} * PNa(i+1) * PNb(i+1)^{-1}$$

$$= PVbi * PVai^{-1} * PNa(i+1) * PNc(i+1)^{-1} * PVc(i+1) *$$

$$* PNb(i+1)^{-1}$$

If

$$PVbi * PVai^{-1} = PNbi * PNai^{-1}$$

then

$$FMb(i+1)' = PNbi * PNai^{-1} * PNa(i+1) * PNc(i+1)^{-1} * PVc(i+1) *$$

$$* PNb(i+1)^{-1}$$

Since body a and body b are TIED together during the action,

$$PNbi * PNai^{-1} = PNb(i+1) * PNa(i+1)^{-1}$$

Thus

$$FMb(i+1)' = PNb(i+1) * PNa(i+1)^{-1} * PNa(i+1) * PNc(i+1)^{-1} *$$

$$* PVc(i+1) * PNb(i+1)^{-1}$$

$$= PNb(i+1) * PNc(i+1)^{-1} * PVc(i+1) * PNb(i+1)^{-1}$$

$$= FMb(i+1)$$

Therefore, the modifying factor of body b determined by equation (8.11) will guarantee the realization of the relationships between body b and the reference body c when the relationships between body a and the reference body hold.

Appendix V

An Example of Vision System Testing

1. A RAPT Program Used in the Test

The following is a RAPT program used in testing the vision command input system, the symbolic reasoning facility and the framework. The coordinates are measured in millimeters and the angles measured in degrees. For simplicity, cameras used in this program are all of simple type (see Section 5.6). There are four bodies defined in this program. They are the world, b1, b2 and b3, and their body sequence numbers are 1, 2, 3 and 4 respectively. Bodies b1, b2 and b3 are shown in Fig. A5.1.

```
body/b1;                    remark begins to define a body named "b1";
  p1=point/0,0,0;
  p2=point/20,0,0;
  p3=point/20,-20,0;
  p4=point/20,-20,30;
  p5=point/20,0,30;
  p6=point/0,0,30;
  p7=point/0,-20,0;
  p8=point/10,-10,0;
  p9=point/10,-10,-25;
```

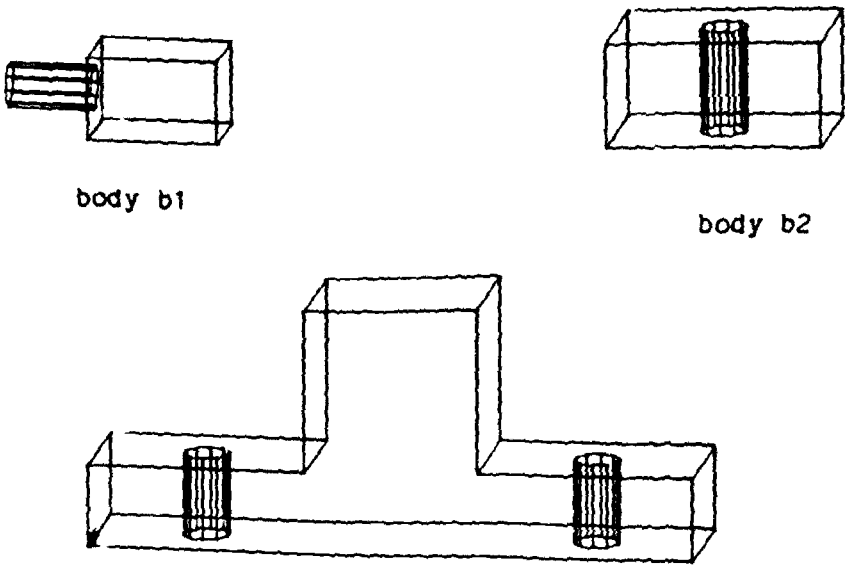


Fig. A5.1. Wireframes of the bodies used in the example testing program

```
      f1=face/p1,p2,p5,ylarge;
      f2=face/p1,p2,p3,zsmall;
      f3=face/p1,p6,p7,xsmall;
      f4=face/p4,p5,p6,zlarge;
      f5=face/p3,p2,p4,xlarge;
      f6=face/horiz,25,zsmall;
  l1=line/p2,p3;
  l2=line/p1,p6;
  l3=line/p8,p9;
      e1=edge/l1,ysmall;
      e2=edge/l2,zlarge;
  s1=shaft/axis,l3,radius,5,zsmall;
terbod;

body/b2;                remark begins to define a body named "b2";
  p1=point/0,0,0;
  p2=point/25,0,0;
  p3=point/25,-30,0;
  p4=point/0,0,50;
  p5=point/25,0,50;
  p6=point/25,-30,50;
  p7=point/25,-15,25;
  p8=point/0,-15,25;
      f1=face/p1,p2,p5,ylarge;
      f2=face/p4,p5,p6,zlarge;
      f3=face/p1,p2,p3,zsmall;
      f4=face/p2,p3,p6,xlarge;
      f5=face/p1,p8,p4,xsmall;
  l1=line/p2,p5;
```

```
  l2=line/p2,p3;
  l3=line/p1,p4;
  l4=line/p7,p8;
      e1=edge/l1,zlarge;
      e2=edge/l2,ylarge;
      e3=edge/l3,zlarge;
  h1=hole/axis,l4,radius,5,xlarge;
terbod;

body/b3;                remark begins to define a body named "b3";
  p1=point/0,0,0;
  p2=point/20,0,0;
  p3=point/20,-15,25;
  p4=point/20,0,50;
  p5=point/60,0,50;
  p6=point/60,-30,50;
  p7=point/60,-30,90;
  p8=point/60,0,90;
  p9=point/20,-30,90;
  p10=point/0,0,140;
  p11=point/20,-30,140;
  p12=point/0,-30,140;
  p13=point/20,-15,115;
  p14=point/0,-15,115;
  p15=point/0,-15,25;
      f1=face/p1,p2,p10,ylarge;
      f2=face/p7,p5,p6,xlarge;
      f3=face/p4,p5,p6,zsmall;
      f4=face/p2,p3,p4,xlarge;
```

```
      f5=face/p7,p8,p9,zlarge;
      f6=face/p11,p13,p9,xlarge;
      f7=face/p1,p10,p12,xsmall;
      f8=face/p10,p11,p12,zlarge;
  l1=line/p1,p10;
  l2=line/p7,p8;
  l3=line/p3,p15;
  l4=line/p13,p14;
      e1=edge/l1,zlarge;
      e2=edge/l2,ylarge;
  h1=hole/axis,l3,radius,5,xlarge;
  h2=hole/axis,l4,radius,5,xlarge;
terbod;

remarks the following statements define some features of the world;

  pw1=point/0,0,0;
  pw2=point/0,0,200;
  pw3=point/0,-150,0;
  pw4=point/200,0,200;
  pw5=point/200,-150,200;
  pw6=point/200,-150,0;
  pox=point/560,-20,50;
  poy=point/260,50,120;
  po3=point/360,-50,-20;
      f1=face/pw1,pw2,pw3,xlarge;
      f2=face/pw1,pw3,pw6,zlarge;
      f3=face/pw5,pw6,pw3,ylarge;
      f4=face/pw2,pw4,pw5,zsmall;
```

```
      f5=face/pw6,pw4,pw5,xlarge;
  lw1=line/pw2,pw1;
  lw2=line/pw4,pw5;
      e1=edge/lw1,zlarge;
      e2=edge/lw2,ylarge;

remark all bodies & features are defined;

agent/b1;        remark this is the body generating the movement;

remark sit 1;

against/f7 of b3, f1 of world;
against/f1 of b3, f3 of world;
against/f8 of b3, f2 of world;
   remark these relations serve to fix b3 with respect to the world;

against/f5 of b2, f1 of world;
against/f3 of b2, f2 of world;
aligned/e3 of b2, e1 of world;
   remark these relations serve to fix b2 with respect to the world;

aligned/e2 of b1, e2 of world;
parallel/f3 of b1, f1 of world;
coplanar/f4 of b1, f3 of world;
   remark these relations serve to fix b1 with respect to the world;

cam1=camera/position,pox,theta,0,phi,75,psi,0,focus,75;
cam2=camera/position,poy,theta,90,phi,0,psi,0,focus,85;
```

```
cam3=camera/position,po3,theta,0,phi,10,psi,0,focus,185;
        remark define cameras;
setcamera/cam1;
        remark sets the default camera to be cam1;

        po4=point/220,50,45;
        po5=point/350,-5,25;
        po6=point/250,5,5;
        po7=point/350,0,110;

cam4=camera/position,po4,theta,-90,phi,3,psi,0,focus,185;
cam5=camera/position,po5,theta,0,phi,90,psi,0,focus,185;
cam6=camera/position,po6,theta,0,phi,90,psi,0,focus,185;
cam7=camera/position,po7,theta,0,phi,90,psi,0,focus,125;

tolerance/b3,trans,5;
tolerance/b2,trans,15;
        remark global tolerances;

        REMARK END OF DEFINITIONS AND START OF ASSEMBLY TASK DESCRIPTION;
        remark now define some movements and vision tasks;
move/b1;
        remark sit 2;
combine;
        remark sit 3, verification of position of b3;
        remark two INVIOLATE statements and one redundant
                LOOK statement in this COMBINE package;
   inviolate/against,f1 of world,f7 of b3;
   inviolate/against,f2 of world,f8 of b3;
```

```
   tolerance/trans,5;    remark local tolerance;
   look/e1 of b3;        remark uses default camera;
   look/e2 of b3;
tercom;

move/b1, perpto, f6 of b1, 200;
        remark sit 4, here b1 is moved to a particular position
                with respect to b3, hence the need to verify b3 above;
   fits/s1 of b1, h2 of b3;
   against/f2 of b1, f6 of b3;
   parallel/e1 of b1, e1 of b3;

      tied/b1,b3;        remark defines a TIE, i.e. b1 grabs b3;

move/b3,perpto, f3 of b3, 150;
        remark sit 5;
turn/b3, about,h2 of b3;
        remark sit 6;
   against/f8 of b3, f4 of world;

move/b3;
        remark sit 7;

      untied/b1, b3;     remark b1 lets go of b3;

move/b1,perpto, f4 of b1,180;
        remark sit 8;
move/b1, perpto, f5 of b1,50;
        remark sit 9;
```

```
combine;
        remark sit 10, verification of position of b1;
   tolerance/trans,5;
   inviolate/against,f5 of world,f2 of b1;
   look/e1 of b1,cam5;
   look/e2 of b1,cam4;
tercom;

combine;
        remark sit 11, verification of position of b2;
   tolerance/trans,12;
   inviolate/against,f1 of world,f5 of b2;
   look/e2 of b2,cam6;
   look/e1 of b2,cam6;
tercom;

move/b1, perpto, f2 of b1, 175;
        remark sit 12, b1 now in particular relationship with b2;
   fits/s1 of b1, h1 of b2;
   against/f2 of b1,f4 of b2;
   parax/e1 of b1, e2 of b2;

      tied/b1,b2;
        remark the following code is mainly to demonstrate the
        effect of the modifying factor array over a super TIE.
move/b2;
        remark sit 13;
   aligned/h1 of b2, h2 of b3;
   against/f5 of b2, f5 of world;
```

```
   parallel/e1 of b2, e1 of b3;

combine;
        remark sit 14, verification of position of b3;
   inviolate/against,f1 of world,f7 of b3;
   look/e1 of b3,cam7;
   look/e2 of b3,cam7;
tercom;

move/b2;
        remark sit 15;
   aligned/h1 of b2, h2 of b3;
   against/f5 of b2, f6 of b3;
   parallel/e1 of b2, e1 of b3;

      tied/b2,b3;        remark b1, b2 and b3 form a super TIE;
move/b3;
        remark sit 16;
   coplanar/f1 of b3, f3 of world;
   against/f3 of b3, f2 of world;
   against/f7 of b3, f1 of world;
terapt;
```

2. The Modifying Factor Array of the Test Program

The following are the modifying factor arrays produced by the framework for the RAPT program listed in Section 1. The first array is that before simplification while the second is that after simplification. In

both arrays the sequence number of the world is 1, that of body b1 is 2, that of body b2 is 3 while that of body b3 is 4.

The Modifying Factor Array Constructed by the Linking Rules

B\S	1	2	3	4	5	6
1	I	I	I	I	I	I
2	I	2,1	2,2	4,4	2,4	[4,6]
3	I	3,1	3,2	3,3	3,4	3,5
4	I	4,1	P	4,3	4,4	1,6

B\S	7	8	9	10	11	12
1	I	I	I	I	I	I
2	2,6	2,7	2,8	P	2,10	3,12
3	3,6	3,7	3,8	3,9	P	3,11
4	4,6	4,7	4,8	[2,10]	4,10	4,11

B\S	13	14	15	16
1	I	I	I	I
2	[3,13]	2,13	[3,15]	[4,16]
3	1,13	3,13	4,15	[4,16]
4	4,12	P	4,14	1,16

* NOTE: 1) A number pair B,S represents a pointer pointing to the modifying factor of body B in situation S.
2) A list [B,S] of a body A in situation S represents a pointer triple [p1,p2,p3] in which p1 is a pointer A,(S-1), p2 a pointer B,(S-1) while p3 a pointer B,S. For example, the list [4,6] of body number 2 in situation 6 represents a pointer triple [p1,p2,p3] in which p1 is a pointer (2,5), p2 a pointer (4,5) while p3 a pointer (4,6).
3) I is an identity matrix symbol.
4) P is a symbolic position expression.

The Modifying Factor Array After Simplification

B\S	1	2	3	4	5	6
1	I	I	I	I	I	I
2	I	I	I	4,4	2,4	[4,6]
3	I	I	I	I	I	I
4	I	I	P	4,3	4,4	I

B\S	7	8	9	10	11	12
1	I	I	I	I	I	I
2	2,6	2,7	2,8	P	2,10	3,12
3	I	I	I	I	P	3,11
4	I	I	I	[2,10]	4,10	4,11

B\S	13	14	15	16
1	I	I	I	I
2	[3,13]	2,13	[3,15]	[4,16]
3	I	I	4,15	[4,16]
4	4,12	P	4,14	I

3. Testing the Run Time Program with Simulated Data

The following is a record of an operation sequence of testing the run time program using simulated vision data. In order to show the consistency of the implementation of the symbolic reasoning facility, the

window suggester and the face generator, some of the simulated data has been made the same as that predicted by the system. Some are not: this is to make the verified positions (simulated) different from the nominal ones so that the modifications to the nominal positions can be shown by the ROBMOD simulation.

*** *** *** *** *** ***

The camera cam1 is working in situation 3

The edge to be verified may appear between

(-17.28073, 7.71058) and (-18.4667, 27.44932);

The window is

p1: (-19;10); p2: (-15;10); p3: (-20;25)

p1: -17 8
p2: -19 25

comment simulated data typed in by the author, in what follows everything typed by the author has been underlined;

The camera cam1 is working in situation 3

The edge to be verified may appear between

(-15.52914, 20.09619) and (-20.18789, 20.09619);

The window is

p1: (-16;18); p2: (-16;22); p3: (-19;18)

p1: -15 19
p2: -20 21

In situation 3 the nominal position of body b3 (4) is:

1.0 0.0 0.0
0.0 -1.0 0.0
0.0 0.0 -1.0
0.0 -150.0 140.0

The real position is:

1.0 0.0 0.0
0.0 -1.0 0.0
0.0 0.0 -1.0
0.0 -147.632 140.0

The camera cam5 is working in situation 10

The edge to be verified may appear between

(0, 12.33334) and (-24.66667, 12.33334);

The window is

p1: (-8;6); p2: (-8;19); p3: (-17;6)

p1: 0 12.33334
p2: -24.7 12.33334

comment the data in this situation is the same as that predicted by the system;

The camera cam4 is working in situation 10

The edge to be verified may appear between

(-33.05263, 56.50052) and (-34.00656, -43.74868);

The window is

p1: (-55;53); p2: (-11;52); p3: (-56;-40)

p1: -33.05263 56.50052
p2: -34.00656 -43.74868

In situation 10 the nominal position of body b1 (2) is:

0.0 0.0 -1.0
0.0 1.0 0.0
1.0 0.0 0.0
200.0 -5.0 35.0

The real position is:

```
 0.0 0.0 -1.0
 0.0 1.0 0.0
 1.0 0.0 0.0
 200.0 -5.000019 35.0
```

The camera cam6 is working in situation 11

The edge to be verified may appear between

(-4.111111, 4.111114) and (20.55556, 4.111114);

The window is

p1: (6;-6); p2: (6;14); p3: (11;-6)

p1: -4 3
p2: 20 5

The camera cam6 is working in situation 11

The edge to be verified may appear between

(-4.111111, 4.111114) and (-4.111111, -37.0);

The window is

p1: (-14;-8); p2: (6;-8); p3: (-14;-25)

p1: -3 4
p2: -5 -36

In situation 11 the nominal position of body b2 (3) is:

```
 1.0 0.0 0.0
 0.0 1.0 0.0
 0.0 0.0 1.0
 0.0 0.0 0.0
```

The real position is:

```
 1.0 0.0 0.0
 0.0 0.9965458 -0.08304549
 0.0 0.08304549 0.9965458
 0.0 1.290586 1.255067
```

The camera cam7 is working in situation 14

The edge to be verified may appear between

(35.71429, 7.142859) and (-14.28571, 7.142859);

The window is

p1: (32;5); p2: (32;9); p3: (-11;5)

p1: 34 6
p2: -14 8

The camera cam7 is working in situation 14

The edge to be verified may appear between

(4.310345, 21.55173) and (4.310345, 8.62069);

The window is

p1: (2;17); p2: (6;17); p3: (2;13)

p1: 3 21
p2: 5 8

In situation 14 the nominal position of body b3 (4) is:

1.0 0.0 0.0
0.0 0.0 1.0
0.0 -1.0 0.0
0.0 100.0 90.0

The real position is:

1.0 0.0 0.0
0.0 -0.04163054 0.9991331
0.0 -0.9991331 -0.04163054
0.0 97.92412 93.31351

4. The ROBMOD Command File Produced by the Run Time Program

The following is an extract from the ROBMOD command file for simulating the RAPT program in Section 1 of this Appendix. This file is generated by the run time program using the simulated vision data typed in in the test procedure shown in Section 3. The first four statements of the ROBMOD program indicate the files in which the wireframes of the bodies are stored. The position of the wireframe of body b1 in a situation is specified by parameters b1x, b1y and b1z while its orientation specified by parameters b1t1, b1t2 and b1t3. So are those of bodies b2 and b3. The statement

"scene = ... "

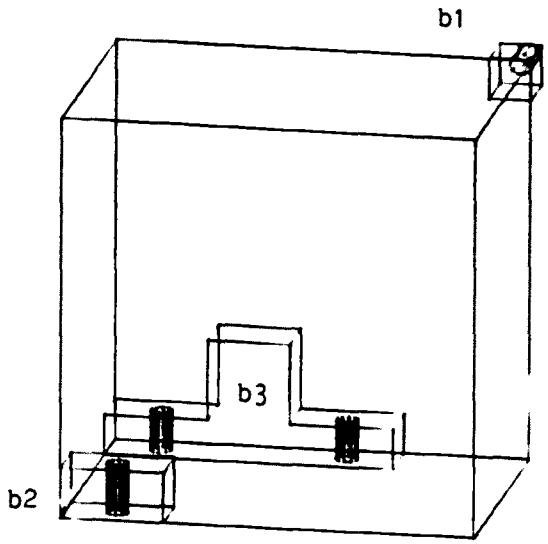
commands ROBMOD to draw a scene on a graphics terminal. The name "wworld" was used instead of "world" because it was a reserved word in an early version of ROBMOD.

```
wworld = rbody "boxa3"
b1 = rbody "block1"
b2 = rbody "block2"
b3 = rbody "block3"
b1x = 200.0000
b1y = -180.0000
b1z = 200.0000
b1t1 =  90.0000
b1t2 =  90.0000
b1t3 =  90.0000
 rb1 = b1 rotz b1t1 roty b1t2 rotz b1t3 to b1x b1y b1z
```

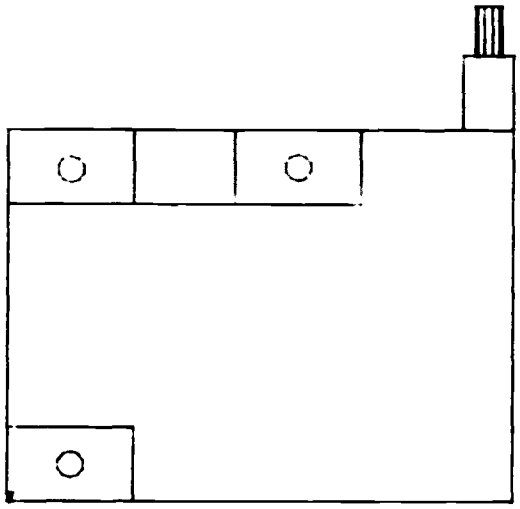
```
b2x =   0.0000
b2y =   0.0000
b2z =   0.0000
b2t1 =   0.0000
b2t2 =   0.0000
b2t3 =   0.0000
rb2 = b2 rotz b2t1 roty b2t2 rotz b2t3 to b2x b2y b2z
b3x =   0.0000
b3y = -150.0000
b3z = 140.0000
b3t1 = -180.0000
b3t2 = 180.0000
b3t3 =   0.0000
rb3 = b3 rotz b3t1 roty b3t2 rotz b3t3 to b3x b3y b3z
scene =  rb3 @ rb2 @ rb1 @ wworld
wire scene
 write 0.2 0.01 "situation 1"
value 0.0
shell "sleep 5"
   ... ...
b3y = -147.6320
wire scene
 write 0.2 0.01 "situation 3"
 write 0.3 0.98 "body b3 is verified now"
value 0.0
shell "sleep 5"
   ... ...
```

5. Some Scenes Produced by ROBMOD

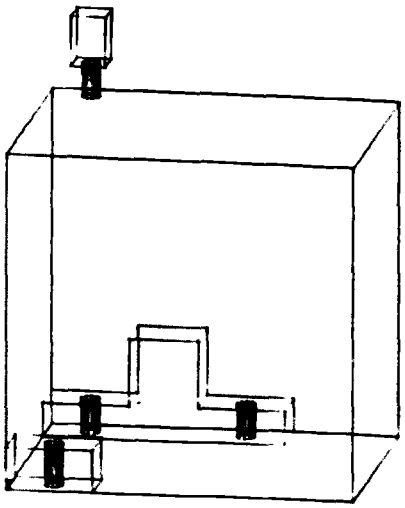
The following are some scenes of the simulation of the RAPT program shown in Section 1. Each situation of the program is shown from two different viewpoints so that a perspective projection and a projection which is perpendicular to the Y-Z plane of the world can be observed. The big box in the following pictures represents a notional world but has no physical significance.



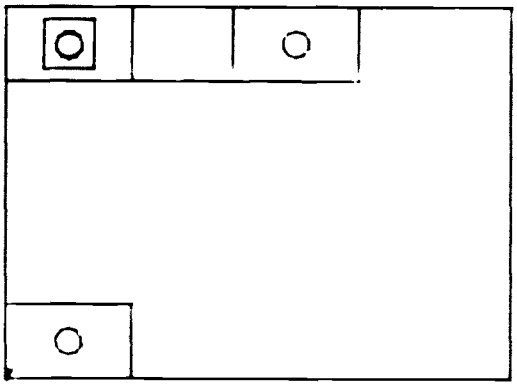
situation 1



situation 1



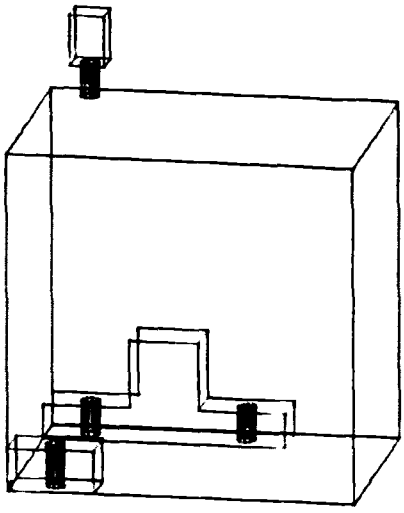
situation 2



situation 2

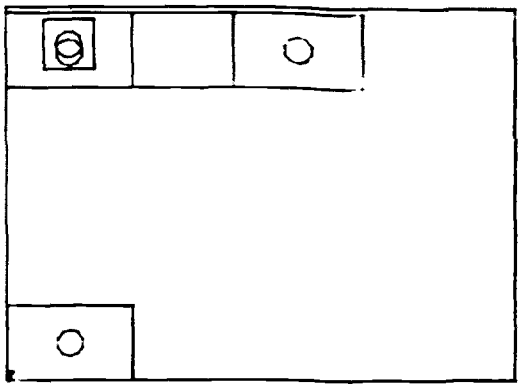
Fig. A5.2. Some scenes of the ROBMOD simulation of a RAPT program

body b3 is verified



situation 3

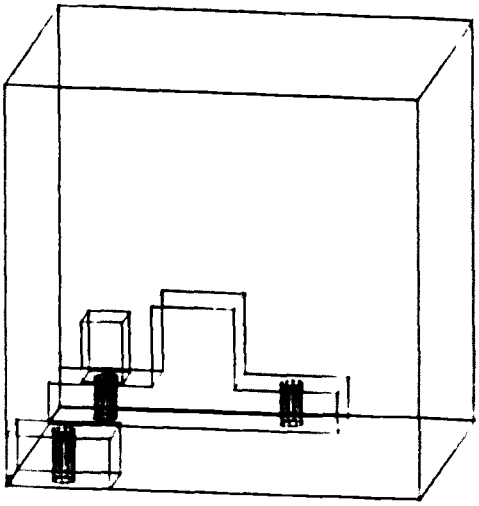
body b3 is verified



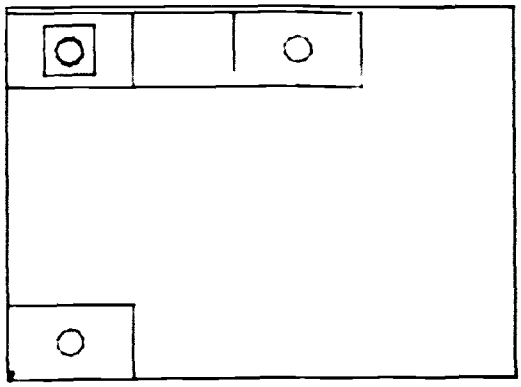
situation 3

Note the actual position of b3 is now displayed and is different from the nominal one displayed in situation 2 but the inviolate relationship has been preserved

Note also b1 is in an inappropriate position with respect to b3



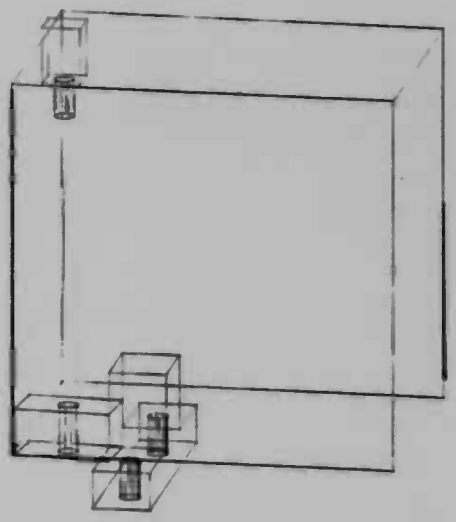
situation 4



situation 4

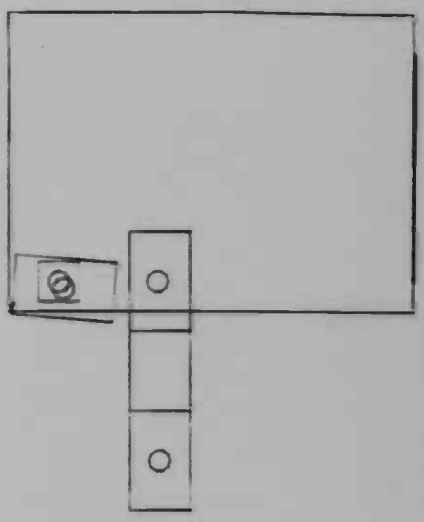
Note the position of b1 has been adjusted to take account of the actual position of b3

body b2 is verified



situation 11

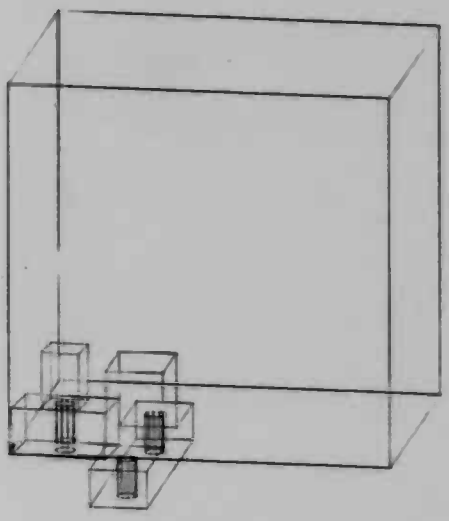
body b2 is verified



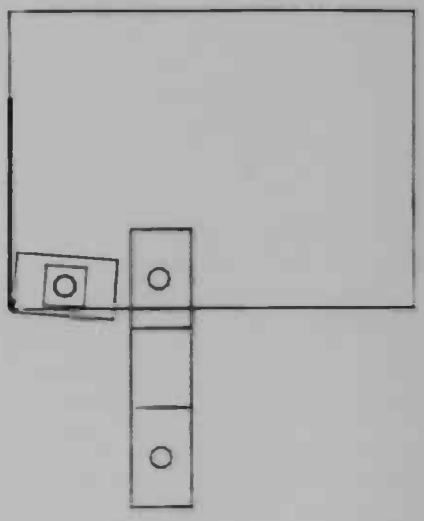
situation 11

Note the actual position of b2 and the

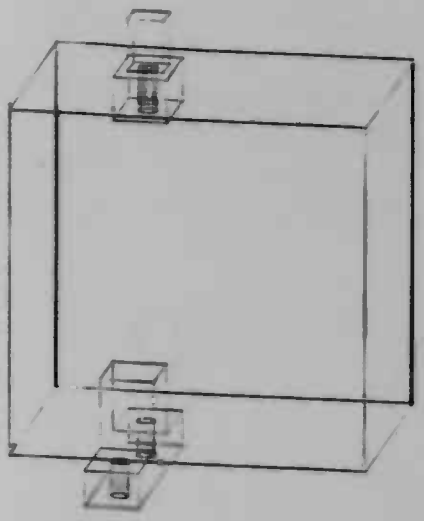
relative position between b1 and b2



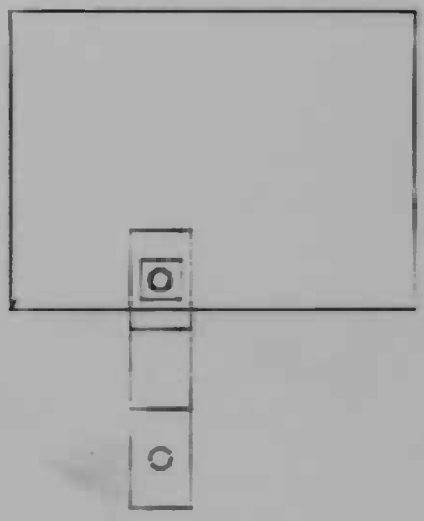
situation 12



situation 12

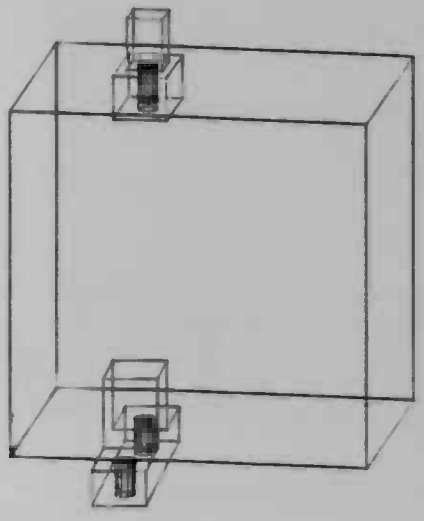


situation 13



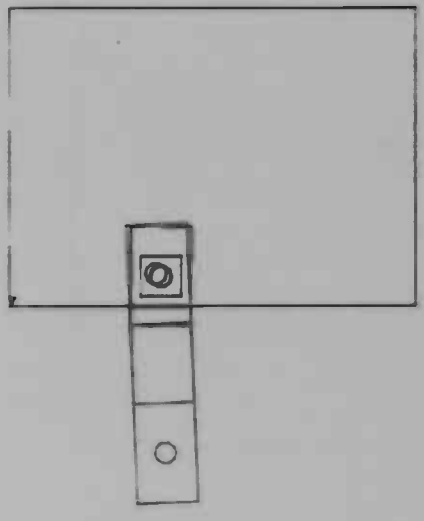
situation 13

body b3 is verified



situation 14

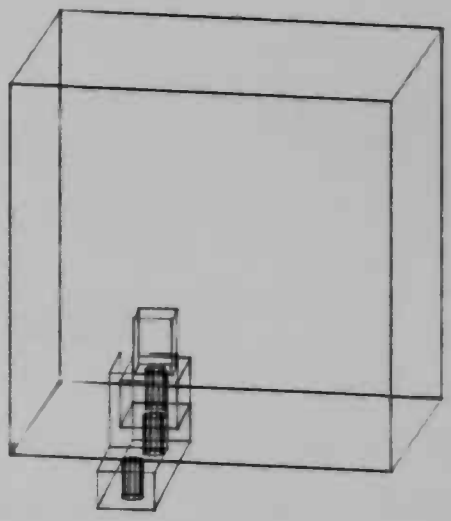
body b3 is verified



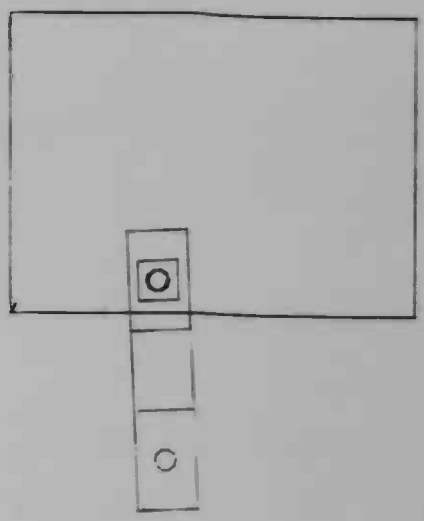
situation 14

Here b3 has to be verified because

the user wants to put b2 down on it



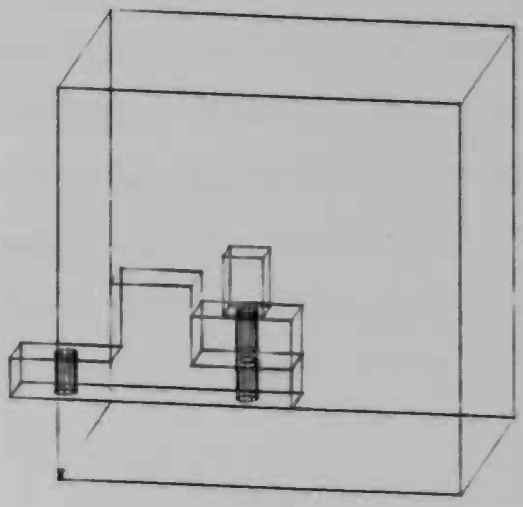
situation 15



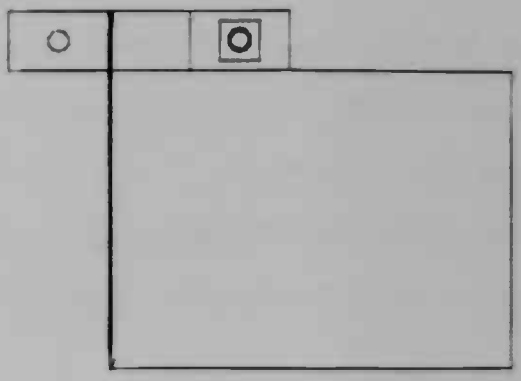
situation 15

b2 is successfully placed in relation

to the actual position of b3



situation 16



situation 16

The super TIE (b1, b2 and b3) moves together

Appendix VI

The Vision Experiment

1. Vision Station

A vision station was set up in order to test the feasibility of using vision to verify the position of the object in a robot environment. The vision station consisted of a plane board, a TV camera connected with a digitizer and a micro-computer (the Vision Box), and a turntable. The camera was supported by a tripod which together with the turntable, was fixed on the top of the horizontal plane board. The object whose position was to be verified was placed on the top of the turntable which could rotate about a vertical axis. The vision station was illuminated by the conventional indoor light sources and no special lighting arrangement was used. The TV camera was connected with the Vision Box and the picture was digitized and stored in the 256x256 image memory of the Vision Box with 256 degrees of grey level. The edge finder (Section 5.7.2) was loaded in the main memory of the Vision Box. The vision station together with the body to be used is shown in Fig. A6.1.

Fig. A6.1. The vision station used in the experiment

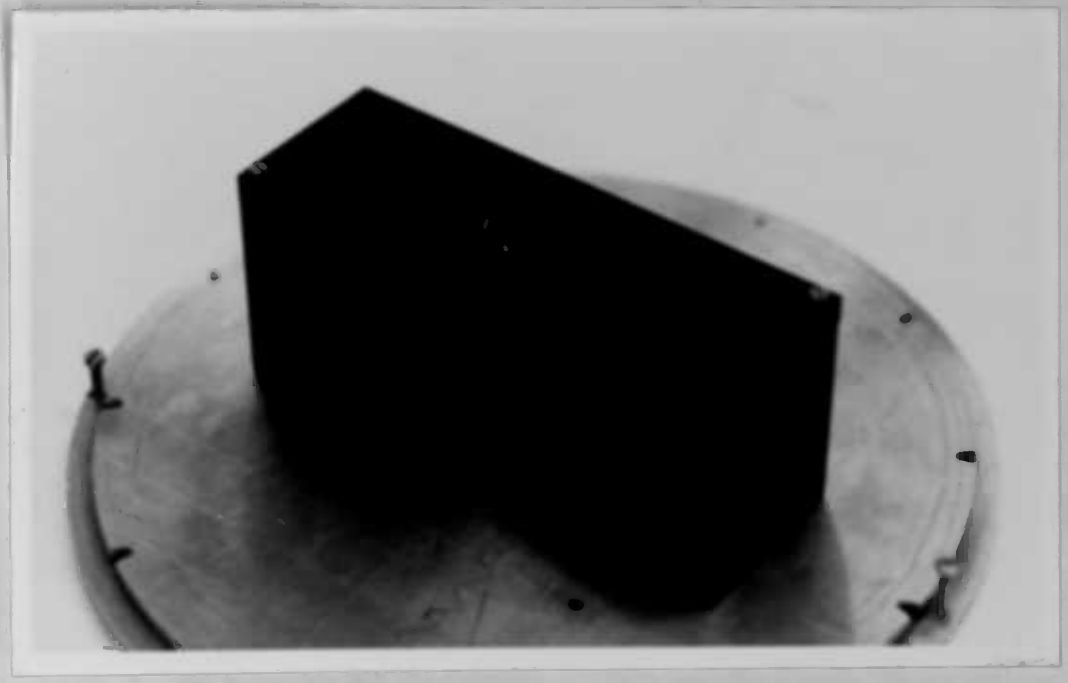


Fig. A6.2. The body used in the experiment

2. Calibration of the TV Camera

The position and the orientation of the camera was measured using triangulation. The accuracy of the position is about 2 mm while that of the orientation is within 1 degree. The non-linearity of the camera in the imaging process was measured but not compensated. This is about 10% in both horizontal and vertical directions.

3. The Testing Program

The following is the program used in the test with some statements deleted which are not relevant to this test. It first specifies the model of the camera and that of the body to be verified and its nominal position, and then specifies the vision task. The body used in this test was the outer case of the gripper being developed in this department. It is shown in Fig. A6.2.

```
body/box;        remark defines the body to be verified;
   p0 = point/0,0,0;
   p1 = point/0,27.5,-81.5;
   p2 = point/72,27.5,-81.5;
   p3 = point/72,27.5,81.5;
   p4 = point/72,-27.5,81.5;
   p5 = point/0,-27.5,81.5;
   p6 = point/0,-27.5,-81.5;
   p7 = point/72,-27.5,-81.5;
   p8 = point/0,27.5,81.5;
   p9 = point/72,0,0;
```

```
        f1 = face/p1,p2,p3,ylarge;
        f5 = face/p1,p5,p6,xsmall;
    l1 = line/p1,p2;
    l2 = line/p4,p7;
    l5 = line/p0,p9;
       e1 = edge/l1,xlarge;
       e2 = edge/l2,zsmall;
       e5 = edge/l5,xlarge;
terbod;

          remark the following defines some world features;
    wp0 = point/0,0,0;
    wp1 = point/0,27.5,-81.5;
    wp2 = point/72,27.5,-81.5;
    wp3 = point/72,27.5,81.5;
    wp4 = point/72,-27.5,81.5;
    wp5 = point/0,-27.5,81.5;
    wp6 = point/0,-27.5,-81.5;
    wp7 = point/72,-27.5,-81.5;
    wp8 = point/0,27.5,81.5;
    wp9 = point/72,0,0;
       wf1 = face/wp1,wp2,wp3,ylarge;
       wf5 = face/wp1,wp5,wp6,xlarge;
  l5 = line/wp0,wp9;
     e5 = edge/l5,xlarge;

pcam1 = point/503,595.0,159.4;
   cam1=camera/position,pcam1,theta,-105,phi,37.2,psi,0,focus,33.1;
        remark defines the camera;
```

```
setcamera/cam1;

remark Now in situation 1;

  coplanar/f1 of box, wf1;
  aligned/e5 of box, e5 of world;
  against/f5 of box, wf5;
        remark these relations define the nominal position of the body;

  combine;
        inviolate/against,wf5,f5 of box;
        look/e1 of box;     remark verifies a vertical edge of the body;
        look/e2 of box;     remark verifies a horizontal edge of the body;
  tercom;

terapt;
```

4. The Test Procedure

The body to be verified was placed on the top of the turntable with an accuracy of about 1 mm in position and 2 degrees in orientation. The run time program which was described in Section 9.2 was called to suggest the window to the edge finder on the Vision Box, and to receive and process the vision information which was obtained by the edge finder. The communication between the edge finder (on the Vision Box) and the run time program (on the DEC-10) was done via a human. Every time the body was placed at a new position or the turntable was rotated, the run

time program was called and the vision data obtained from the Vision Box was typed in. The following is the record of a segment of the test procedure:

```
The camera cam1 is working in situation 2

The edge may appear between

 (224.299,79.51659)  and  (51.14755,45.06556);

The window is

p1: (209;85);  p2: (211;68);  p3: (63;56)

p1:  209 78
p2:  65 46

The camera cam1 is working in situation 2

The edge may appear between

 (36.00486,173.0825)  and  (30.64297,88.95519);

The window is

p1: (20;161);  p2: (49;158);  p3: (16;103)

p1:  39 158
p2:  35 100

In situation 2 the nominal position of body box (2) is:

 1.0 0.0 0.0
 0.0 1.0 0.0
 0.0 0.0 1.0
 0.0 0.0 0.0

The real position is:

 1.0 0.0 0.0
 0.0 0.9995763 -0.02910785
 0.0 0.02910785 0.9995763
 0.0 -0.1896315 0.1235619
```

5. The Result of the Test

The result of the test is contained in the following table. The table also shows the comparison of the setting positions of the body and its verified positions obtained by using vision information. The position is measured in millimetres while the orientation measured in degrees. Since there is an INVIOLATE statement specified in the vision task, the position of the body has only three degrees of freedom: two translational and one rotational. In all cases the nominal position of the body was the identity matrix.

	setting position			verified position			comparison		
	Y	Z	theta	Y	Z	theta	Y	Z	theta
1	0	0	0	-0.19	0.12	-1.6	-0.19	0.12	-1.6
2	0	0	-15	-1.12	-2.16	-17.3	-1.12	-2.16	-2.3
3	0	0	15	-1.29	3.98	12.8	-1.29	3.98	-2.2
4	-10	0	0	-11.65	3.69	-0.1	-1.65	3.69	-0.1
5	10	0	0	7.16	0.09	-0.1	-2.84	0.09	-0.1
6	0	-15	0	-3.77	-19.39	-0.3	-3.77	-4.39	-0.3
7	0	10	0	-1.46	9.35	-0.2	-1.46	-0.65	-0.2
8	-10	-10	0	-13.02	-9.815	-0.2	-3.02	0.185	-0.2

6. A Brief Analysis of the Accuracy

The accuracy of the verified positions is mainly dependent upon the calibration of the camera and the resolution of the image. As mentioned

in Section 2 in this Appendix, the accuracy of the position of the camera is about 2 mm while that of the orientation is within 1 degree. The non-linearity of the camera imaging process is about 10%. Each pixel in the image corresponds to about 1 mm distance on a plane which is perpendicular to the X-axis of the camera and placed at the same distance as that between the camera and the body to be verified. The accuracy of setting the positions is about 1 mm and that of the orientation is within 2 degrees. It can be seen from the result that the accuracy of the verified position is of the same order as that of the calibration.

Because of the lack of proper equipment, it is difficult to increase the accuracy of both the calibration and the setting. Under these conditions the result obtained could be considered as satisfactory. With better equipment and therefore better calibration, higher accuracy of the result of the vision verification achieved this way could be expected. Calibration is relatively infrequent compared with the use of vision information. Making allowance for the temporary nature of the vision station, this experiment has demonstrated the usefulness of the vision verification in robot assembly.