Multi-Robot Workcell with Vision for Integrated Circuit Assembly

Christian Michaud

B Eng Universite Laval Québec, 1983

Department of Electrical Engineering

McGill University

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

M Eng . McGill University, 1986

July 28, 1986

Abstract

This thesis describes the software tools developed to perform multi-robot assembly of Integrated Circuit (IC) with vision The tools developed include the development of a multi-robot workcell, the design of a database, and the development of image processing algorithms to detect corners and rectangles. All the facilities developed are integrated in order to experiment with robotic IC assembly

Résumé

Cette thèse décrit un système multi-robot utilisant la visionique pour assembler des circuits intégrés Le développement d'un environnement multi-robot, la conception d'une banque de données, et la mise au point d'algorithmes de traitement d'image pour la détection de coins et de rectangles, sont décrits Les résultats des expériences d'assemblages sont discutés

Acknowledgements

The author would like to thank all the members of the Computer Vision and Robotics Laboratory for their constant support during the course of this work I would like to especially thank my supervisor, Dr. Malowany, who provided me with the required guidance when necessary I would like also to thanks colleagues such as Paul Freedman, Bruno Blais, Kamal Gupta, Wade Hong, Gregory Carayannis, Barıs Demir, Abdol-Reza Mansouri, Iskender Paylan and Mike Parker Their professionnal advice as well as their perpetual sense of humour (and countless coffee sessions) made this research more productive and much more interesting

I would like to acknowledge my immediate family, to whom this thesis is dedicated, for the continued support they have provided me through these years of study FCAR financial support is also gratefully acknowledged.

Contents

List of Figures

Chapter 1 Introduction

Since the "Industrial Revolution companies have sought greater efficiency in their production The first major step was the introduction of hard automation machines which could produce the same product in vast quantities Then Numerically Controlled (NC) machines became available to precisely execute cutting instructions according to a program punched on paper tape [Pressman, Williams 1977] These NC machines brought a new dimension to the industrial process, one machine could be used to produce slighty different products Although these products must be of the same category e g boat propellers, the flexibility gained through programming saved both time and money Since then, NC machine programming has become even more sophisticated with the integration of Computer Aided Design and Computer Aided Manufacturing (CAD/CAM) The most recent step in the industrial evolution is the introduction of robots in the production plant

At first, robots were used for pick and place operations As technology improved, the use of robots became more pervasive Robots are now used for many different tasks such as welding, palletizing and assembly Nowadays, the use of multi-robot workcells permit more complex tasks to be performed faster However, a robot performing complex tasks requires sophisticated sensory feedback The most acclaimed type of sensory feedback is *vision feedback* [Levine 1985] but it is the most complex to use

Robotics research in the Computer Vision and Robotics Laboratory (CVaRL) of McGill University is mainly oriented towards the inspection and repair of hybrid integrated

circuits and printed circuits boards in a dynamic 3-D environment using a distributed set of sensing elements (eg vision), manipulating elements (eg robot, X-Y stage), and "knowledge" elements (eg data base, expert system) The intent is to develop the hardware and the software tools needed in a robotics workcell to visually inspect and repair certain types of circuit defects

This thesis project addresses the Integrated Circuit (IC) assembly task and includes the development of a multi-robot workcell the design of a database and the development of image processing algorithms

1.1 System Overview

The McGill University Computer Vision and Robotics Laboratory incorporates three VAX minicomputers, four Sun workstations, and a wide variety of peripheral equipment, as shown in Figure 1.1 A VAX 11/780 running VAX/VMS and Eunice, a Unix emulator, is linked with two VAX 11/750 minicomputers running UNIX 4 2BSD, by a 10 MHz Ethernet local area network The VAX 11/780 is used for the Artificial Intelligence oriented programs which control the overall workcell One of the two VAX 11/750s is used to control robots and peripheral equipment, while the second is used for the vision processing The Suns are used as software development stations

A Grinnell monitor with a resolution of 256 by 256 pixels is installed on to the Vax 11/780 permitting images to be taken from a camera, which can be mounted on a microscope A Rembrandt camera system is linked to the Grinnell for the generation of slides

There are three robots available in the laboratory A PUMA 260 (Figure 1 2) from Unimation, an ECUREUIL from Microbo and a IBM7565 robot. The PUMA robot is an articulated manipulator with six degrees of freedom. It has a quasi-spherical workspace, and this offers great flexibility for the robot motions. The repeatability of the robot is forty microns. The ECUREUIL robot, shown in Figure 1.3, is a cylindrical robot

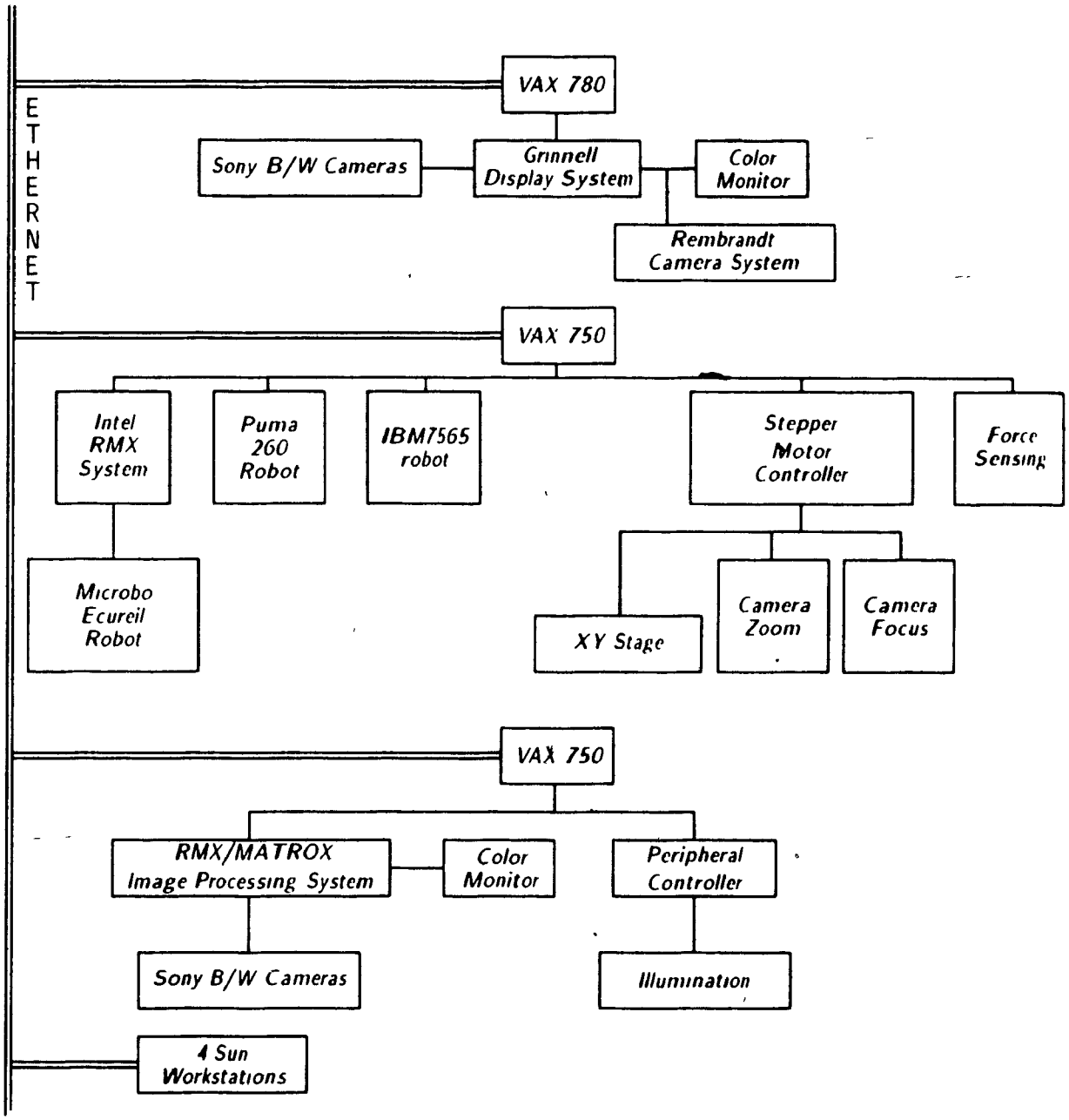

Figure 1.1 Facilities available in the CVaRL laboratory

with six degrees of freedom It has four revolute and two prismatic joints Although, the ECUREUIL allows more precision in movement (repeatability of five microns), its work area is much more restricted. Both the PUMA and the ECUREUIL robots are powered by DC electric motors The IBM7565 is a cartesian robot with 6 degrees of freedom (Figure 1 4), and a repeatability of one hundred thirty microns. The PUMA and the ECUREUIL

Figure 1.2 PUMA 260 robot

Figure 1.3 Ecureuil robot

robots are mounted on adjacent tables to share a common workspace The newly acquired IBM7565 robot stands on a separate table and is not used in the current IC assembly system The PUMA's native language is VAL, while the Microbo's native language is a similar but more primitive language called IRL In order to make the programming of cooperative tasks simpler, software packages have been written in the C programming language to serve as interfaces to the VAL and IRL robot languages [Carayannis 1983, Michaud 1985]. The IBM7565, shown in Figure 1 4 is provided with a poweful high level language AML and includes asynchronous networking software to link it into the VAX environment Basically, these programs convey commands from remote terminals to the robot controllers by way of RS-232 links The disadvantage of this option is that software differences reflect the characteristics of each robots Therefore, it was decided to unify the programming languages of the PUMA and the ECUREUIL robots through a common flexible language. This new manipulator level language called RCCL (Robot Control C Library), was originally developed at Purdue University and subsequently improved and implemented at McGill on the PUMA robot [Lloyd 1985] , and on the Microbo robot [Kossman 1986]. In addition, an object-level language, which uses RCCL as a base, is currently under development The ECUREUIL robot controller is linked to an Intel RMX System in order to support the RCCL robot language. The PUMA and ECUREUIL robots together form the robotics workcell used for hybrid and integrated circuits assembly and repair.

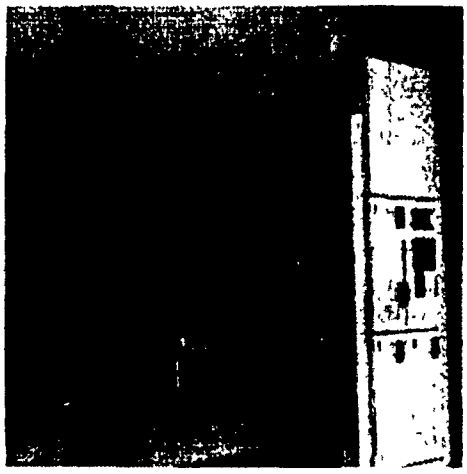

Figure 1.4 IBM7565 robot

A Stepper Motor Controller(SMC) is used to control an XY-stage, the zoom, and focus position of the microscope A software package has been written in order to control the SMC from a remote host The precision of the XY stage system is 1 3 microns

Multipurpose end-effectors are useful features in any robotics laboratory where a robot must perform a variety of different tasks A number of tools are required to maintain the flexibility of the robotic workcell. The hybrid circuit assembly and repair project at the CVaRL involves a spectrum of different electronic components, and for each component, a specialized interchangeable end-effector. All tools have been fitted with a standard plate which mates to brackets fitted to the end-effectors of both robots. This is possible due to the fact that the robot end-effector has a standard mechanical arrangement, and that all the tools can easily be interfaced to the wrist In a typical robot assembly task, the robot first picks up the tool for holding hybrid circuits, places the circuit on a jig below the camera, returns the tool to its tool rack, and then picks up the tool that is appropriate for the subsequent task In this way, different tasks such as capacitor de-soldering or IC DIP chip insertion can be performed by the same robot

1.2 Robotics Research at McGill

The goal of the Computer Vision and Robotics Laboratory is to develop a facility

for research in hybrid circuit inspection and repair In the following sections, we highlight certain parts of the research program

1.2.1 Distributed Control

CVaRL is currently involved in the development of a reliable and efficient interprocess communication scheme for our distributed robotics environment based on message passing [Gauthier et al 1985] Communication facilities are provided that allow messages to be passed over communication channels between network endpoints. It is based on the Transport Communication Protocol (TCP) [TCP 1982] which is a part of the Berkeley Unix BSD4.2 operating system. Use of the message passing facilities is achieved via function calls, similar to operating system calls. After, a communication session is set up, virtual communication endpoints on the Ethernet may be created between hosts. Subsequently, a virtual path may be created between two or more endpoints permitting the sending and receiving of messages

1.2.2 Database

Database research centers around the development of a standard methodology for the organization, storage and retrieval of information for a robotics environment. As evidenced by the variety of databases and knowledge representation schemes that have been developed and are currently under development, there is no obvious unique solution to the storage and representation of knowledge [Aristides 1984]

One project, in which the author was involved, focussed on modeling and state representation issues [Freedman et al. 1986]. Two representational approaches have been investigated: Abstract Data Type and expert systems Subsequently, one implementation of Abstract Data Type, written in C , plus two expert system implementations, written in OPS5 and in PROLOG, respectively, were developed. The expert system implementations were considered in order to explore the capabilities of these two languages for robotic

database applications More details are presented in Section 3.2 The Database work in C is now being extended for a project in visual inspection Here, the database is concerned with the inspection of hybrid integrated circuit boards [Blais 1986] It is based on a hierarchical Computer Aided Design (CAD) model of a printed circuit board and its components

1.2.3 Vision

The emphasis in vision research is on visual inspection and on robot hand-eye coordination As far as visual inspection is concerned, research is being performed on solder joint inspection, hybrid capacitor inspection, hybrid IC inspection and finally the inspection of conductive traces on the hybrid substrate. For the case of capacitor inspection, a line detection algorithm has been developed and successfully tested on hybrid circuit images [Mansouri et al. 1985b].

Research has been performed toward the building of a general purpose vision system that could be used for robotic visual inspection [Hong 1986] This system has the structure of an Expert System and is used for the low level segmentation of images It is planned to further enhance the knowledge base of this expert system in order to perform higher level processing and interpretation of images

Robot-vision coordination, also referred to as hand-eye coordination, is being performed for simple cases of robot end effector positioning As an example, sensory information is used in order to derive the position of an IC Dual In line Package (DIP) chip relative to the robot end effector. Once this position, which varies from one DIP chip to another, is derived, the insertion coordinate is updated in order to allow the insertion to be properly performed [Mansouri et al. 1985a]

More details concerning the environment and the research performed in the CVaRL at McGill may be found in [CVaRL 1985].

Figure 1.5 IC material to be handled

1.3 Project Description

The project presented here consists of developing the necessary software tools to use multiple robots with vision for IC die assembly. Several issues have been addressed: robot control, database design, and image processing.

Ideally, the process consists of putting dice on an IC DIP Chip on which solder paste has been applied. There are many ways to perform IC assembly in a multi-robot environment. Two robots, the PUMA 260 and ECUREUIL, and a microscope are used. The ECUREUIL is used for the dice manipulation, and the PUMA 260 for the DIP chip and dice array carrier manipulation The microscope is used for detecting the position and orientation of a die as well as for inspection. The material to be handled is shown in Figure 1.5.

The global sequence of operation begins with the calibration of the workcell. Then, the PUMA 260 robot takes a DIP chip and puts it on the XY stage. Meanwhile, the ECUREUIL robot moves near the microscope and is ready to put on the paste. As soon as the PUMA 260 has finished its move, it goes to get the dice array and puts it in the jig near the ECUREUIL robot while the latter applies the paste. Then, the ECUREUIL goes to take a die and puts it on the XY stage. Thereafter, the XY stage moves the die under the microscope. When it is in position, an image of the die is taken and the XY

stage moves immediately back to where it was while the vision system computes the exact position of the center of the die. When the result is known and the XY stage is in position, the ECUREUIL picks up the die at the center and moves it over the dice carrier. By that time, the orientation of the die is known, which enables the ECUREUIL to put it in place Meanwhile, the PUMA 260 has replaced the dice array by the next required kind of dice. This interleaved sequence is shown in Figure 1 6. Note that the synchronization between robots is also used for collision avoidance A proposal for the approach presented above is introduced in [Michaud et al. 1986b]

In order to develop the proper environment to perform these tasks, the following problems were identified

1- robot control.

2- modelling

3- database design.

4- finding of the location of the IC die via image processing

For the purpose of investigation, the problem was split into in three phases. First, the robot control, modeling and database design were addressed in the context of a multi-robot demonstration in which a trace is cut on a board. The demonstration was intended to prove the validity of message passing to control several robotic processes distributed over different host computers. In the second phase, image processing algorithms were developed to precisely locate an IC die. It is shown that these algorithms can be generalized for other types of problems. Finally, robotic workcell experiments were carried out for the IC die assembly task in order to evaluate the performance issues.

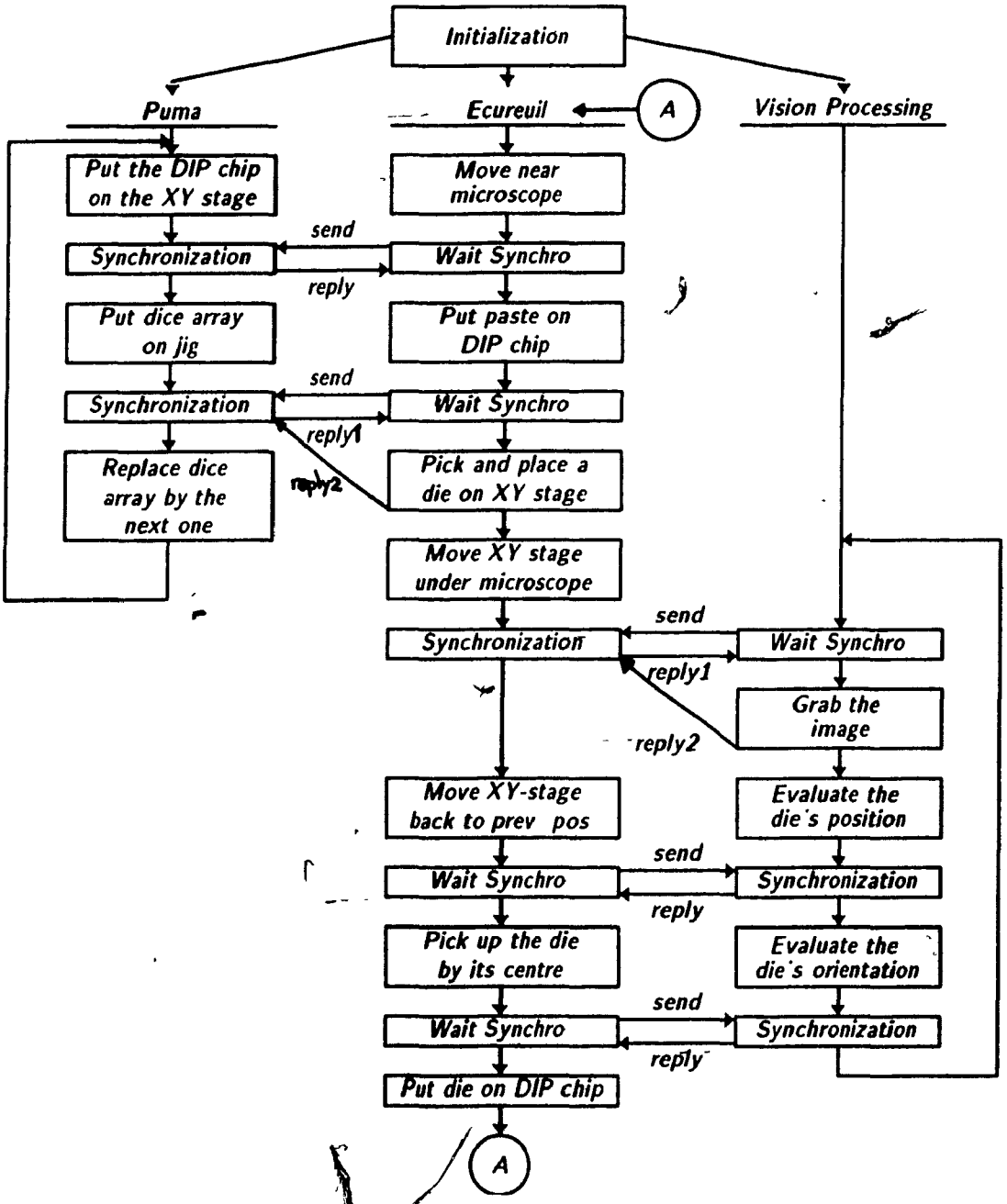

Figure 1.6 Global sequence of operations

Chapter 2

Literature Survey: Robotics, Concepts and Requirements

This chapter introduces concepts and requirements involved in the design of a robotic workcell. These concepts and requirements arise from the multidisciplinary nature and varied applications of robotics, encompassing a spectrum of areas such as sensory feedback, cooperative tasks, and world modeling. The robotic workcells considered here will examine the case of both single and multi-robot stations. Special care will be taken in the sensory feedback section concerning the use of vision feedback for hybrid and integrated circuits tasks.

The chapter is partitioned into three main sections. Namely, robot configurations, modeling and database design, and sensor-based robotics.

2.1 Robot Stations

The purpose of a robotic workcell is to perform object manipulation tasks. Clearly, the main advantage of robots over any dedicated object manipulation machine is that robots can be reprogrammed to perform different actions, while this is not usually true of dedicated machines.

Several considerations must be taken into account when a robotic workcell is to be designed. The proper robot configuration, which can be either single or multi-manipulator, has to be determined. In addition, there are usually a number of components

such as vision systems and other types of sensors and actuators that must be controlled. Efficient control of the manipulators and of the components usually implies the use of distributed processing. This raises the problem of synchronization among different processes. These considerations are examined in detail in the following sections.

2.1.1 Robot Configuration

Most tasks in industry can be performed with one arm whenever the use of proper holding fixtures is possible, eg. palletizing, welding, screwing and some assembly tasks. Therefore one might question the need for several robots. Conceivably the use of multiple robots would result in three advantages First, it speeds up the operation to be performed by introducing either parallelism or pipelining in the sequence of operations. Second, a higher degree of freedom is obtained in defining the tasks. Finally, it enhances the flexibility of the system by allowing it to perform tasks that require more than one arm.

The importance of multi-robot operation has long been realized [Chand and Doty 1983]. The choice of using one, two, or even more robots is dependent on the tasks to be performed. For example, a simple palletization task will require only one robot, while performing spot welding on a car body with only one robot is too time-consuming and can create a bottleneck in the assembly line. In this case, the use of a multi-robot configuration appears more attractive. These multi-robot configurations are especially applicable to assembly tasks which are becoming a major area of application in robotics [Fox and Kempf 1985]. Actually, the emerging trend is to use multiple multi-robot stations.

Whenever large quantities of the same kind of product are made, or whenever high is precision required, the use of dedicated hard automation is advantageous. In contrast, there is no way to justify the implementation of hard automation when producing small batches of different products. Here the use of robots is more promising. In the context of small batch production the use of multi-purpose tools is the most interesting

option. These consist of a group of different specialized tools attached to the same end effector which permits the robot to perform tasks requiring different tools without having to change the end-effector. Some tasks, which seemed to require several single-purpose end-effector robots, may in fact be performed by only one robot with a multi-purpose end-effector. Making such multi-purpose end-effectors interchangeable permits the exchange of one end-effector for another in order to perform a different kind of task. However, it is often desirable to use multiple robots with single non-interchangeable end-effectors. Spot welding of car bodies falls into this category. In this case, there is no reason to use interchangeable multi-purpose tools, but using several robots will enhance the production throughput. These specialized end-effectors satisfy specific needs and further increase the efficiency of the station for a particular application.

The tasks to be performed by a robotic workcell are part of a schedule which describes the order of the actions to be carried out. However, the use of distributed processing is more efficient and cost effective in providing the computing power required to efficiently process all of the incoming data and controlling the robots. Thus several operations must be performed in parallel [Shaw and Whinston 1985] , thereby making the generation of an optimal task schedule very difficult. To ease this problem, we can view a task or goal as a partially ordered [1] set of subtasks (*SoS*) or subgoals, which are themselves partially ordered sets of actions [Fox and Kempf 1985]. This permits further optimization of the schedule by performing tasks using the most suitable machine if it is available [Shaw and Whinston 1985]. Unfortunately, one cannot always assume that the original schedule will be valid for the entire course of the operation. For example, the system may detect a malfunction that requires the original schedule to be updated. The complexity of these robot systems is such that the scheduler, which produces the schedule, must be able to generate a new *SoS* at run time [Alami and Chochon 1985] from its knowledge of the current

---

[1] A partially ordered set is a set in which the order of some elements may be modified, within certain constraints, without affecting the end result. For instance, if an action A is to be performed before actions B and C which have to be done before action D, the two following valid sets will be obtained: { A B C D } and { A C B D }.

state of the world Scheduling, also called activity planning, is complicated by the sharing of resources such as conveyor belts, robots, space, and so on However, this constraint may be somewhat relaxed by implementing resource supervisors for each group of similar resources [Sedillot 1984, Chand and Doty 1983 Maletz 1983, Fox and Kempf 1985, Alami and Chochon 1985] These supervisors are responsible for the allocation of the specific resource they are controlling, thus releasing a part of the burden from the scheduler. This is illustrated in Figure 2 1

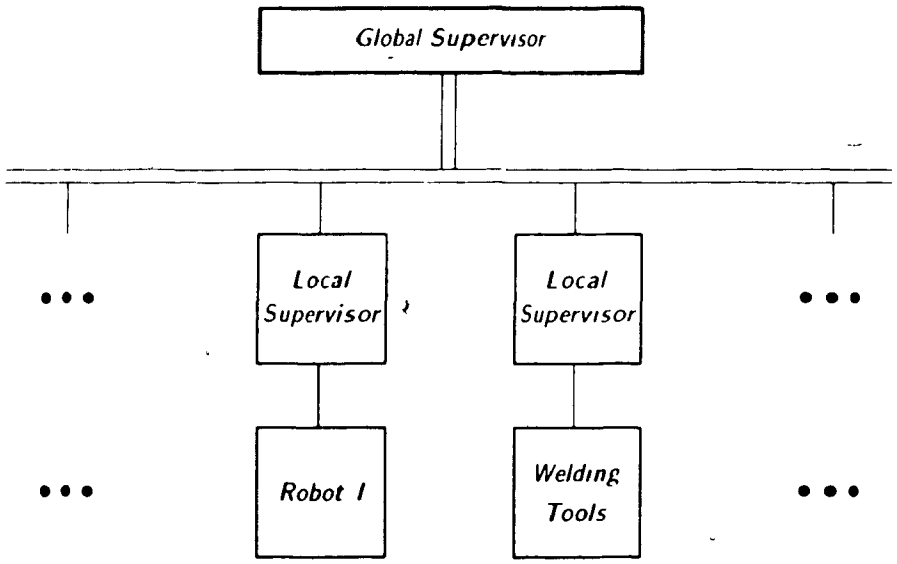

Figure 2.1 Example of resource supervisor control by a global supervisor

2.1.2 Distributed Control

Generally, robotic workcells are distributed systems with several processes co-operating to achieve desirable performance In order to do so the dominant architecture used is a global scheduler controlling several local ones [Sedillot 1984, Shaw and Whinston 1985, Tyridal 1980] . The degree of cooperation among different machines can then be measured by the amount of information exchanged among the different components of the

system [Chand and Doty 1983] The main advantages expected from using distributed architectures [Jensen 1986, Kopetz and Kuroda 1980 Williams 1979] are

1- Good reliability and fault tolerance can be offered via redundancy

2- Higher performance can be achieved through parallelism

3- The hardware and software of the system are expandable in a modular way, depending on the architecture of the system

In distributed processing, inter-process communication is an important issue because the problems are distributed and the partial results must be recomposed Standardized communication mechanisms are required to achieve this reconstruction efficiently The main advantage of a standardized communication mechanisms is that it permits the integration of products from different manufacturers Three differents approaches have been taken for inter-process synchronization These are the language, the network-oriented, and the backplane bus approaches

In the language approach, the communication primitives required for inter-process synchronization are embedded in the programming language An example of this is ADA, which allows parallel tasks to be executed Tasks synchronize themselves by a rendezvous between the task issuing the entry call and the task accepting it ADA has been considered [Voltz1984] for programming robot-based manufacturing cells The major drawback of the language approach is that it forces all programming to be carried out using one special language in order to make inter-process communication possible This would seem to be impractical in robotics since it may often be advantageous to write the scheduler in an Artificial Intelligence language such as LISP while heavy computations are more efficiently implemented in C or FORTRAN There is also the possibility that one wants to use in-house software developed in a different language Thus, what seems more

appropriate is an operating systems orientation in which the communication primitives are independent of the programming language

This network oriented approach, based on local area network technology, introduces a more flexible way of linking the distributed elements of a robotic workcell due to its independence of the communication primitives vis-à-vis the programming language used Several network oriented approaches have been proposed for robotic workcells [Harmon and al 1984, Garetti et al 1982, Milne 1983, Bruno et al 1984] and some manufacturers have implemented their own network interfaces For instance Unimation's VAL-II robot controller [Shimano and al 1984] contains a network interface for supervisory control Unfortunately, most manufacturers of flexible automation systems provide proprietary solutions to the communication problem This complicates the problem of integrating different robots together Fortunately, this situation is changing One recent development is the set of Manufacturing Automation Protocols(MAP) promoted by General Motors [Leopol 1984, Kaminski 1986] MAP is a layered communications standard based on the Open Systems Interconnection(OSI) model [Zimmermann 1980] for linking distributed elements within a factory It is rapidly becoming a *de facto* standard in factory automation, and has already gained the endorsement of such notable institutions as IBM, Ford, and Boeing

In contrast to loosely coupled networks, there are also tightly coupled multiprocessor systems These consist of multiple processors linked by buses developed by the microprocessor industry An example of a robot implementation that uses this approach is a tightly-coupled hierarchy of 16 microcomputers for the coordinated control of two PUMA arms [Alford and Belyeu 1984] The Multi-Arm Coordination Computer transmits new position commands to each robot via high speed block transfers to intermediaries called Prediction Computers (PC) The PC's then use a simple handshaking protocol to issue new setpoints to the joint controllers.

Of these three approaches, none can claim to be the ultimate one, although the most promising are the network-oriented and backplane approaches. The network-oriented approach provides a very flexible solution to the communication problem. However, due

to its relatively slow speed, it is practical only when the messages to be exchanged are short On the other hand, the backplane approach provides fast communication, which is required for most real-time applications involving several sensors Finally, the language approach suffers from the needs to restrict all the software development to one language This is a limitation for both software and system integration every part of the system must support the given programming language in addition to having a standard method of communication among them

2.2 Modeling and Database Design

To be able to perform efficiently in a robotics environment, it is important to have an adequate representation of the environment, which provides ways to retrieve or derive all the required characteristics of a desired object at a specific time These characteristics, which can be geometric or physical properties, or functional abstractions, must be stored in an efficient and consistent way in a proper database structure which represents the state of the world surrounding the robots This will be considered as the world modeling problem. The problem of building the facilities required to manage these representations will be referred to as the database management problem

2.2.1 World Modeling

The problems of world modeling in robotics are to store, collect, and manage the information This is done either via the use of sensors, from which the desired features are extracted by sensor processing, or by basic guiding[Lozano-Perez 1983]

Aristides [Aristides 1984] has noted that "*there is no single object representation scheme which is uniformly the best* " and that "*redundant data play a pivotal role in achieving efficiency*" This observation is strengthened when a Database (DB) must meet engineering needs, which typically demand a wide variety of data types, facilities to express complex relationship between entities, and support for knowledge representation [Hartzband

and Maryanski 1985] It is a simple fact of life that different kinds of tasks require different kinds of information A detailed Computer Aided Design (CAD) template of a given object might be appropriate for image processing, in order to generate a synthetic image (based on camera position and orientation) for simple matching based on features However, path planning on a primitive scale can be performed using just an object's bounding volume Hence, specific information about a particular object, such as a microscope, might be maintained in different forms, depending if one is interested in geometric information or general properties such as color or magnification This suggests that a single integrated information storage and retrieval system could be defined to satisfy all needs at once. In fact, a prototype of such a system is described in [Mitchell and Barkmeyer 1984] for the distributed manufacturing facility of the National (US) Bureau of Standards Information about work orders, inventories, and the states of the various workcells are all combined, in a hierarchical way, within a single logical database entity which provides uniform access to all data Their implementation physically exists as a set of disk-resident databases, plus shared memory used by the real-time control processes

Information *storage/retrieval* also means different things in different contexts Consider a typical CAD system, which provides some type of high-level model representation, such as a three-dimensional solid. Inside, processes interact in a low-level way (eg lock, store, fetch) with internal data structures which represent information at a much lower level, such as polygons. There is no knowledge intrinsic to the model; this is defined by the relationships between the parts of the model, which in turn is defined by the user when the CAD representation is built The "user interface" to the CAD system typically provides a means to perform high-level model transformations, and might also incorporate some natural language understanding.

For robotics-oriented work, data structures might be sufficient for simple collision avoidance (when shared work volumes are explicitly defined), but not for planning assembly or repair tasks. Here, the knowledge required cannot be represented at this low level; something more powerful or symbolic is needed, such as production rules embedded

in an expert system Note too that there is no "user interface" per se. we are talking about a collection of robotic processes interacting with the database

We can also draw distinctions between off-line and on-line usage of information In well-defined contexts. certain kinds of planning such as task decomposition can be performed off-line. i.e generating a series of sub-tasks from a single high-level directive such as "assemble pump" This is because the information required is *static* But for real-time coordination of concurrent activities within a workcell. we require on-line updating of state information which is *dynamic*

This coordination is further complicated by the distributed nature of complex robotic applications A typical program would consist of sets of control and sensing processes resident on distinct hosts [Sedillot 1984] These processes will require different kinds of information at different times for different purposes This makes the availability of the database information an important issue From a programmer's perspective. it would be simplest to have just one database which is equally accessible to each host This could be implemented by a set of dedicated servers. one per host. which would hide the actual location of the database. and provide a standard interface for each user process However. multiple access of this kind raises other issues about the writing. reading and ownership of data

2.2.2 Representational Issues: From Database to Knowledge Base

In general. we can say that information stored in a database can be classed as *entities*. and *relationships* among them [Date 1975] For example. if the entities are resistors. capacitors. and PCBs. then one relationship might be COMPONENT OF.

The *Data Model* is the user's view of what is in the database. and the *data sublanguage* defines the user interface. this is sometimes called the *Query Language* because most user interaction with a database takes the form of queries about the stored

information. When the database exists as a separate process, the interface is usually called the Database Manager (DBM).

As discussed in [Sowa 1983] , conventional data models were developed to represent many repetitions of a small number of data types eg 10,000 instances of an employee payroll account The data types are determined by the systems analyst, not the user Internally, data is represented as machine-oriented *records*, fixed linear sequences of field values Typically, the database is too large for main memory, and must reside on secondary storage (eg tape, hard disk) . Conventional database research continues to emphasise mechanisms for efficient data storage and retrieval, and topics such as error detection and recovery

There are three 'traditional' data models relational, hierarchical, and network. It is easiest to describe them by using a simple example Consider a set of Printed Circuit Boards (PCBs), and a set of COMPUTERS composed of PCBs. If the PCBs all have (n) components, then we can think of each one as a (n+1)-tuple defined by a serial number and the quantities of the components. All the tuples together would then define a *relation* called PCB. This is the Relational Data Model, and is typically represented as a table, see Figure 2.2. All information is represented by the basic structure (object, attribute, value). A class of objects of the same type constitutes a relation, associated with a table; each column in the table is an attribute.

Note the clear distinction made between the user view (tables) and the internal organization of the data (records) To determine what components belong to a given board (or what PCBs make up a given COMPUTER), we locate the appropriate serial number and then read across the row. To determine on which boards a given component (or a given quantity of that component) occurs, we locate the component column and then read across to the serial number. An extra level of searching is required to discover which COMPUTERs have certain components. Other kinds of queries based on the relational algebra using connectives such as AND, OR, NOT can be easily expressed. For instance, one may ask which board has specific components X1 AND X2.

relation PCB

Serial Number	# Resistors	# Capacitors	# Transistors
1	10	4	0
2	6	0	5

relation COMPUTER

Serial Number	# PCB_1	# PCB_2	# PCB_3
1	3	2	5
2	0	4	3

Figure 2.2 The Relational Data Model

In a Hierarchical Data Model, all the information about each entity is grouped Continuing with our example, we might choose to define the components as basic entities which 'belong' to the PCBs (Figure 2.3).

Note that the full meaning of a node is only apparent when the node data is seen in the context of the complete hierarchy. Responding to a query means first traversing the hierarchy to find the appropriate location But hierarchy is not an appropriate model for applications based on shared relationships, as when a single leaf node (record) belongs to multiple branches of the hierarchy.

The Network Data Model allows for more flexible associations by replacing the

COMPUTER_1
3 PCB_1
2 PCB_2
5 PCB_3

COMPUTER_2
0 PCB_1
4 PCB_2
3 PCB_3

PCB_1
10 Resistors
4 Capacitors
0 Transistors

PCB_2
6 Resistors
0 Capacitors
5 Transistors

Figure 2.3 The Hierarchical Data Model

hierarchical structure with a network The nodes represent the tuples of data, and the arcs represent the relations (Figure 2.4) In the context of artificial intelligence, this could be called a *semantic network* More formally, a semantic network is a directed graph consisting of nodes (objects) and labelled edges (relationships) [Gevarter 1985] .

Thus, the Network Model can represent associations more flexibly than the Hierarchical Model. In addition, arcs can express arbitrary 'many-to-many' relationships. However, the Network Data Model also has its disadvantages. All relationships are explicit; nothing can be inferred about inheritance of properties on the basis of position (which is central to the Hierarchical Model). Answering queries means searching the network for all possible paths. The user must also bear in mind how the arcs form chains, since this will affect how new relationships should be defined.

Of these conventional Data Models, the Relational one has become the most

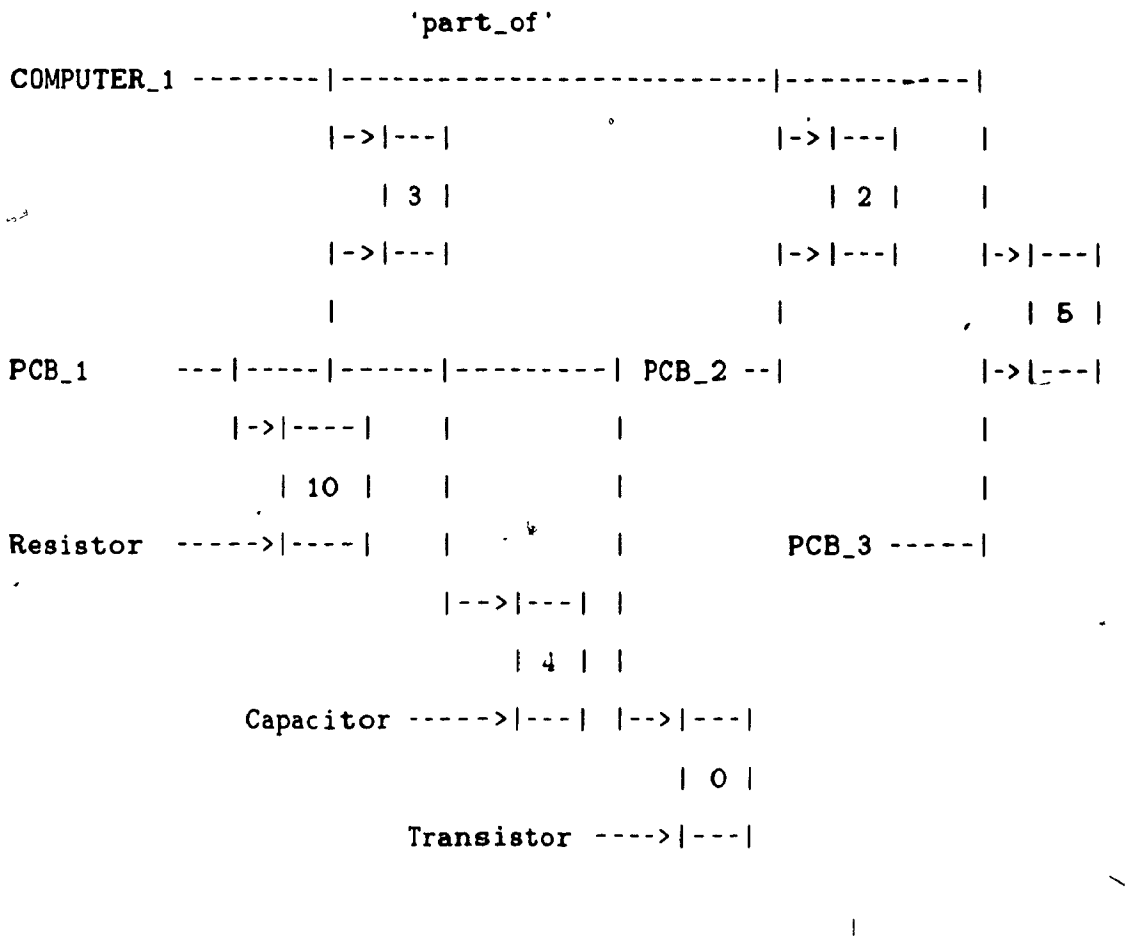

Figure 2.4 The Network Data Model

popular. In part, this is due to the flexibility of the table representation and the power of relational algebra behind the query semantics However, when the time comes to consider engineering applications where much complex information is present, a large amount of inter-related information must be kept [Dittrich and Lorie 1985, Haskin and Lorie 1982]. In the case of the Relational database, this is motivation for the concept of *complex object* [Dittrich et al. 1985] , which can be seen as a group of elements built in such a way that they naturally capture the hierarchical relationships in a relational framework The need for this concept points out the weakness of the relational database approach when dealing with general data types like those required in engineering [Kim and Banerjee 1985].

Further optimization of the hierarchical model leads to Abstract Data Type, which can be seen as a group of data structures with an associated set of applicable

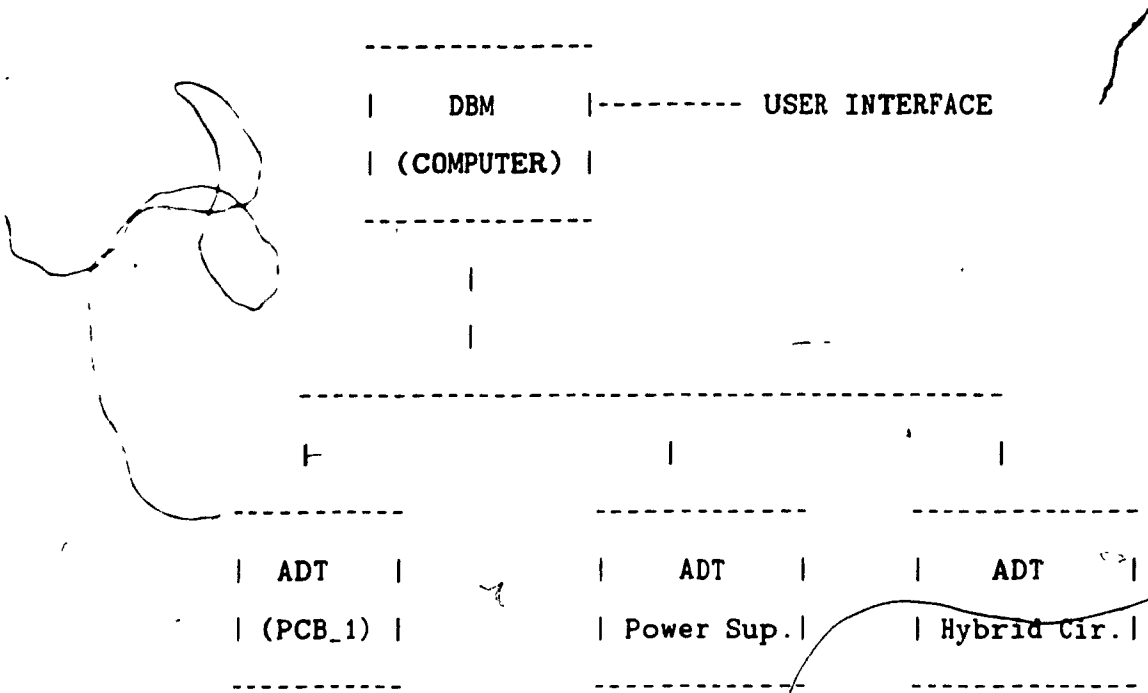

Figure 2.5 The Abstract Data Type Model

operations [Claybrook et al 1985, Guttag 1975, ACM Workshop 1980]. For example, consider a query about a PCB of type PCB_1 associated with COMPUTER_1. The query is sent to the top-level DBM, and then passed to the DBM which 'manages' information about PCB_1s. At this point, a group of routines accesses the appropriate data structures and then translates the data into an appropriate form. The structure of the COMPUTER/PCB data base might look like Figure 2.5.

This seems to be more appropriate for robotics because it offers a more flexible scheme to keep Information. The *frame* as defined in AI offers a similar representation. As described in [Winston 1977], a frame is a data structure for representing typical situations; in our context, these would be models. A frame is simply a collection of objects and relationships represented as an array of named 'slots'. The slots associated with the frame have default values, as assigned by the model. For example, we might define a frame called PCB, with a slot for the number of resistors.

The artificial intelligence community has also addressed the database issue, but within the context of expert systems for real-world modelling and 'knowledge' representation. The word 'knowledge' also implies something more powerful than just 'data' or 'information'. In [Bic and Gilbert 1986] , 'knowledge' is defined as 'information' in a form usable by an agent to alter the flow of control. This is why there is much more to representing 'knowledge' than just data structures. An artificial intelligence 'Database' is usually a blend of an *explicit* forms (facts) called *declarative* knowledge, and *implicit* forms (rules) to infer new facts from the existing ones; this is called *procedural* knowledge. Consider once again information about printed circuit boards and computers This could be expressed, for example, in first order predicate calculus as follows

The two templates:

```
pcb(Serial_num, Num_resistors, Num_capacitors, Num_transistors).
computer(Serial_num, Num_pcb_1, Num_pcb_2, Num_pcb_3).
```

Some specific examples:

```
pcb(1, 10, 4, 0).
computer(1, 3, 2, 5).
```

With this structure, we can derive all the remaining information. For example, here is a rule for determining the number of resistors X to be found in the computer Y (N1 times X1 from boards of type pcb_1, N2 times X2 from boards of type pcb_2, etc.).

```
num_resistors(X1,X2,X3,Y) <- computer(Y,N1,N2,N3) AND pcb(1,X1,_,_)
                                  AND pcb(2,X2,_,_) AND pcb(3,X3,_,_).
```

This description is more compact than making all information explicit, as 'facts', but most

queries will be answered more slowly, since information must be derived (inferred). Interest in logic programming has led to the development of many 'expert system database's such as [Genesereth 1985, Getta 1984, Nakashima 1984].

Finally, several 'hybrid' Data Models have been recently described in the literature. The intent is to incorporate information about how to 'reason' with and about the data stored in a traditional way, by building sophistication on top of a conventional Data Model. One such system called KM-1 is described in [Kellogg 1984, Kellogg 1986], which employs a logic-based 'reasoning engine' (deductive processor) to derive implicit information from the data stored by the "searching engine" in a relational database. In [Udagawa and Mizoguchi 1984] , an abstraction mechanism based on graphics is described for a CAD relational database. Another hybrid is described in [Lafue and Mitchell 1983] which incorporates an expert system to help guide the multi-step design of digital circuits. There is also much interest in using an expert system as a front end to a relational database, to provide sophisticated user features such as natural language understanding [Barnes et al. 1983].

2.2.3 Database Management

Database management in robotics means providing a standard way of arranging, storing, and accessing information, in a real-time manner.

Some database work can be better described as support for CAD (planning), instead of control. For example, a hierarchical structure can be used to store information about how the components in an assembly are connected [Lee and Gossard 1985] , in addition to geometric information about the components. Connections are represented by 'virtual links' between components or 'instance's of components, when there are several of the same kind in the assembly. The work described in [Lee and Gossard 1985] emphasises how to interactively create the database.

A production system for planning robot tasks is described in [Dufresne 1983] with knowledge represented exclusively as rules in a production system. Backward-chaining is used (as in PROLOG) to derive the required plan.

But in an uncertain environment, it might be wiser to plan in a limited way on-line; in [Fox and Kempf 1985], this is called 'opportunistic scheduling'. The sequencing constraints (partial orders) about tasks are expressed as statements in a sequencing language which resemble production rules These constraints and the current state of the world are then used to determine a schedule, based on forward-chaining

Already mentioned is the database work at the National (US) Bureau of Standards [Mitchell and Barkmeyer 1984] , which is a part of the Automated Manufacturing Research Facility research program. Dedicated 'data servers' provide uniform access to the complete database. Data managed by the server includes the current status (location) and contents of all trays, and markers associated with the progress of the active control programs. Conceptually, a control process communicates with the data server via shared memory organized as mailboxes. Each mailbox has an access control mechanism to lock it when a process is reading or writing The actual transport of 'mailgrams' is subject to flow control to balance the speeds of the communicating processes

An 'intelligent' monitoring and error diagnosis/recovery system is described in [Barnes et al. 1983] , for industrial robots with multiple sensors. Knowledge is represented as frames, with cause and effect relationships encoded as antecedent and consequent pairs of 'slots'.

An alternative database design methodology for robotics is described in [Gini 1983]. Knowledge about the current state of the world is stored in a symbolic way, as a frame-like structure. Knowledge about how to 'translate' sensory data into this form, and how to recover from 'errors', are encoded as production rules. Error recovery would be triggered when the expected outcome of an action is different from the actual one.

A hierarchical database is described in [Geisler 1983] for a robot vision system

performing inspection of industrial parts Objects are defined in terms of surfaces (eg top, bottom), interior regions (eg area, centroid), and finally segments Each surface also has an associated set of features to guide the pattern recognition step of the vision system

A knowledge-based system intended for a robotic assembly cell is presented in [Kak et al 1986] In their system, a Current World Model is used to maintain a description of each object in the workcell The current world model serves two purposes First, it is used to coordinate activities within the workcell For this reason, a semaphore is associated with each entry to regulate the reading and writing by multiple sources Second, it is meant to log sufficient information to reconstruct the state of the robotic workcell Two kinds of knowledge are distinguished information not subject to change during an assembly operation, and information that changes as the system state change

But more relevant to us is the work reported in [Blume 1984] A knowledge base called RODABAS (RObot DAta BASe) is described, which uses a relational database RODABAS is used to store and maintain a 'World Model', as part of a sophisticated hierarchical programming environment, but the author presents no details about how this interacts with other modules such as the 'Interpreter Virtual Machine' (workcell controller) or an expert system called the 'Planner'

2.3 Sensory Information Processing

Most current industrial robot applications are performed with binary sensing and the environment is tailored in such a way as to eliminate errors and uncertainties that might occur at different stages of a manufacturing process This is performed by means of specialized guides and fixtures which usually accommodate a very limited variety of components Although such an approach might be justified for a stable production line, a tremendous cost is associated with any slight change in production, due to all of the redesigning and reworking that has to be done

The other approach, that of "Flexible Assembly" systems, would require a

minimum of dedicated guides and fixtures Rather, these requirements are relaxed in order to cover a wider range of components and assembly types The obvious advantage of this approach is due to its flexibility In other words, the same assembly station can be used to assemble a completely different product with only a minimum of reworking of the workcell components The price to be paid however is that of application program complexity While in a fixed automation station exact information is available as to the position and orientation of different components, such information may not available in a flexible assembly context Sensory feedback may thus be required to determine such unknowns In his analysis using information theory techniques, Sanderson [Sanderson 1983] suggests that while sensory information may be expensive to acquire, it is cheaper to store, transfer and manipulate than information acquired in a fixed automation context

The ultimate goal in using sensors is to acquire knowledge about the environment This knowledge can either be used to inspect given objects or to find their exact position and orientation Each sensor, however, has a different output characterizing it An intermediate processing step is therefore needed to translate the sensory signal into a meaningful description of the surrounding world In what follows, a description of vision sensors is given

2.3.1 Vision

Vision is the most widely acclaimed type of sensory feedback, while being at the same time the most complex to use [Levine 1985, Rosenfeld and Kak 1982, Nevatia 1982] Typical visual sensing schemes involve a medium-to-high resolution CCD camera array, together with a fast link to a central processing computer The processing model involved in vision is the following A set of features are to be extracted from an image [Duda and Hart 1973, Tou and Gonzales 1974, Levine 1969] , and based on a classification of these features, an interpretation of the image is to be generated Visual information processing can itself be classified into many different categories In each category, however simple, the processing model mentioned above is valid

The simplest kind of vision is what is usually termed "Binary Vision" The image to be analyzed can only have two different intensities, black (logical zero) or white (logical one) Such systems are very limited since they give only a very approximate representation of the external environment Surface shading and light intensity changes are eliminated The advantage however resides in the ease of use of such systems A small amount of memory is required to store the image and its corresponding processing software Processing of these binary images revolves mostly around object identification via simple geometric properties such as perimeter and surface measurements [Rosenfeld and Kak 1982, Nevatia 1982] More complex methods of shape identification via Fourier analysis and moment calculations are also in use [Zahn and Roskies 1972] In general, simple algorithms which deal only with surface and perimeter comparisons can be performed in real time on dedicated single microprocessor systems, while additional computing power is needed to handle more sophisticated algorithms The main use of binary vision is in part localization and silhouette identification on conveyor belts, where the part of interest is backlighted In the image, the part appears as a dark region on a light background The visual inspection performed by binary vision systems is limited to simple cases where objects are easily discriminated from their background. In addition, slight surface intensity changes which might prove important for inspection tasks are usually not captured.

Next in complexity comes gray-level vision Gray level vision allows the processing of images which, in addition to the white and the black intensities found in binary images, contain a number of intermediate gray tones (thus the term gray-level). Gray level images contain more information about their surrounding environment than binary images do, and permit, under certain conditions, the reconstruction of the three dimensional information present in a scene An example can be found in shape from shading techniques, where variations in surface intensity are clues to the description of surfaces in three dimensions [Horn 1975]. Processing of gray level images can be sophisticated enough to justify the use of a dedicated computer. Due to the fact that gray level images provide a more faithful representation of a scene than binary images, the number of features that can be extracted from such images is correspondingly larger than the features that can be

extracted from binary images In addition because of the presence of a large number of intermediate gray levels, object boundaries are not as apparent in gray level images as they are in binary images

Before an object can be analyzed and identified, it must be located Intermediate processing steps are therefore required in order to discriminate between objects and their background This process is called image segmentation [Fu and Mui 1981, Baird 1977, Levine and Shaheen 1981] Two of the most commonly used processing techniques for segmentation are thresholding [Weszka 1978, Weszka and Rosenfeld 1978] and region analysis [Brice and Fennema 1970, Zucker 1976] Thresholding and region analysis (also termed region growing) concentrate on finding uniform regions in the image, where uniformity is defined in terms of intensity or texture

Thresholding is the simplest way of segmenting an image into different regions It consists of assigning the maximum intensity to all image points that have an intensity above a certain threshold and the minimum intensity to the remaining image points The threshold may be chosen adaptively as a function of the intensity distribution in the image The result of thresholding is therefore a binary image

Another class of image processing techniques concentrates on boundary detection by enhancing region differences The most commonly used technique is edge detection Finding edges is of importance in most object recognition algorithms since edges usually correspond to object boundaries In addition, edges are usually characterized by discontinuous intensity changes which are rich in high-frequency content Thus, it is possible to enhance region boundaries by a proper selection of spatial convolution masks Typical masks are the Roberts cross operator [Roberts 1965] , the Sobel masks, and the Hueckel masks [Hueckel 1971]. What distinguishes these masks from the perspective of performance is their immunity to noise. Once edges are found, it is desirable to retain only those that correspond to object boundaries, and to discard those that result from surface specularities and other undesirable effects. Such a selection is usually done by removing weak edges. As a result, in addition to boundary edges, a number of edge elements that do not

1	0	-1
2	0	-2
1	0	-1

-1	-2	-1
0	0	0
1	2	1

Figure 2.6 Two Sobel masks for computing the vertical and horizontal components of the gradient

correspond to any object boundary are usually retained The Sobel masks are shown in Figure 2 6

In order to be able to match object boundaries to object models, a set of features must be defined, and a similarity criterion determined to identify the closest match to the model These features should be carefully selected to yield robustness to changes in position, orientation and scaling. The matching can be performed either globally, which means that a complete model will be used to match features, or locally, where only features corresponding to sub-parts of the model are to be matched to features extracted from the digital picture. The latter scheme is needed for the identification of partially occluded objects [Bolles and Cain 1982]. Once the matching is completed, each object in the image is labeled with the label of the model that yielded the closest match Once objects are properly labeled and located, a number of processing steps can be performed to inspect the surfaces of these objects. Again, intensity and textural features of the surface of the located object are extracted and compared with those of the model in order to yield a quality measure

The desired features may be extracted directly from the image or from the edge image However, due to the considerable amount of information contained in an image, data compression must be done. The three most frequently used data representations for data compression are quadtree, medial-axis transformation and chain-code representation. Quadtree [Samet 1980] is an encoding of the spacial occupancy array. A quadtree can be considered as having a pyramid as an intermediate representation of a image array. The pyramid is build by dividing the image in four areas of equal size, and checking if all the

pixels have the same values for each area If so, there is no need to look any further in this area On the other hand, if some of the pixels are different, the division process is repeated over the area investigated until every area examined contains pixels of the same values The representation obtained is more compact than the original picture providing the picture has cluster of pixels having the same intensity The medial-axis transformation (also called skeleton) [Blum 1962] provides a method which is capable of computing a description of a natural shape It is possible to obtain the skeleton of an object by "grassfire analogy" [Levine 1985] Assume that we light a fire along the boundary of the object The fire is considered to grow in all directions from each point on the boundary When two sources of fire intersect, the fire is extinguished at the point of contact This point is part of the skeleton of the image The chain-code representation assumes that a binary image is available A chain-code is formed by the sequence of integers that describe a closed contour The integers may take values between 0 to 7, which represent the 8 possible directions of the next move from pixel A to the neighbouring pixel B

Color processing may also be useful in visual inspection and robot vision [Nevatia 1977] , since color provides strong cues for discriminating different objects and surfaces In addition, a large number of visible surface defects have an equivalent color representation When properly used, color information can therefore be a strong discriminatory feature between different objects and/or object quality levels. A major overhead involved with using color images resides in the computer memory required. While one memory plane can store a gray-level picture, three such planes are required for storing an equivalent color picture, each plane now storing a monochrome picture of one of the fundamental colors

Three dimensional vision is yet another important aspect of robot vision Since a large number of robot applications are tracking applications (seam welding, for example) where the robot end-effector is required to keep a constant orientation with respect to a continuous three-dimensional surface, the exact three-dimensional configuration of the surface has to be known. Three techniques are available stereo vision [Grimson 1985, Luh and Klaasen 1985] , structured lighting [Hall 1982] and laser range finding [Faugeras

1984] In stereo vision, complete and non-ambiguous three-dimensional information cannot be inferred from a single image Therefore, a number (usually 2) of images taken from different angles must be analyzed, and as a result of this processing, each point in the image is uniquely mapped to a coordinate in three-dimensional space. The computational problems involved in stereo vision have confined it to a research stage at present. The structured lighting technique has been developed to overcome the complexity of analyzing three dimensional scenes by projecting a uniform laser light pattern (usually a grid pattern) on the objects of interest Images taken under these lighting conditions are therefore independent of the surface properties and reflect only the three-dimensional outline of the perceived objects. A similar technique developed for three-dimensional scene analysis is laser range finding Here, a laser beam is directed towards the surface of an object, and the resulting image consists only of a spot of light which corresponds to the point of intersection of the laser beam with the surface of the object The transformation between the perceived position of the spot and its real three-dimensional position is done using information about the geometry of the setup. A complete range map of a scene can therefore be traced by sampling it at a number of different locations [Rioux 1984]

2.4 Further Considerations and Requirements for a Robotics Station

Although this survey has covered many topics of robot systems, there are still many facets to this discipline that have been omitted. These are mostly specialized areas of research which are constantly evolving and which cannot be fully detailed in the span of a single chapter. Nevertheless, a short insight into each of these areas is provided below, as well as their present and possible future contributions to the field.

The kinematics and dynamics of robot manipulators have been an active research topic since the introduction of the first robot manipulators [Paul 1981]. The kinematics of robots involves the computation of the frame transformations between different links of a robot arm. The direct kinematic problem consists of synthesizing the position and orientation of the robot end effector in cartesian world coordinates, given robot joint values.

The inverse kinematic problem involves doing the reverse: that is, generating the set of joint values that yield a desired position and orientation to the end effector A major area of research directly related to the kinematics of robot manipulators is that of planning paths to avoid singular configurations While robot kinematics deals with static configurations, robot dynamics deals with the dynamic aspects of robot motion such as the end effector velocity as a function of joint velocities, acceleration, and inertia

One important area of research is that of the adaptive control of robot manipulators [Nitzan 1985] Adaptive control involves the adjustment of robot parameters as a function of the current state of the robot manipulator The adaptive control of robot manipulators allows the design of "optimal controllers," where optimality is usually defined in terms of minimizing of the energy required by the robot for its motion

Robotics has also created a new market for peripheral equipment, which consists mostly of special purpose sensors and end-effectors Presently, numerous companies are competing for this market For the past few years, there have been several suppliers offering a wide range of end-effectors Therefore, today's robot owner can worry less about building his own customized tools for particular applications The most recent, and also the most promising development is related to computer vision systems Most have special hardware architectures to perform inspection and other visual sensing in real time The developments in this field are largely due to the rapid advances in the domain of VLSI

One of the major requirements of a flexible manufacturing station is to be programmable off-line This is very important because the setting-up of any specific application must take the workcell off-line and several trials may be required before obtaining the desired performance Therefore, proper simulation software packages are necessary They must fulfill several requirements. First, they should take into account the physical characteristics of the robots and of the equipments such inertia, size, limitations on speed and acceleration. Second, facilities must be provided to properly represent every object in the environment on a display device. Although much progress has been made in this field, there is still much to be done.

As far as artificial intelligence is concerned, it is foreseable that in a near future, major research results will be implemented in a practical industrial context. Its major applications will be in the development of friendlier user interfaces, automatic multi-robot task generators, collision avoidance strategies, and also sensory information processing. Among the many possibilities of merging artificial intelligence and sensory feedback, one can mention the development of rule-based expert systems for industrial visual inspection systems.

The reliability of a system is one of its most fundamental aspects, especially in a multi-robot industrial environment where the costs associated with faults are usually quite high. Implementation of fault tolerant schemes are thus required. There are three different methods for improving the robustness of a multi-robot system. At the lowest level, hardware redundancy offers the opportunity to perform as long as good components are available. At the next level, software redundancy provides support to recover from transient errors. Finally, process recovery can be done via the use of artificial intelligence techniques to recover from the more difficult problems [Gini 1983].

For a robot to operate safely, it must avoid collisions with obstacles in its work space. This is generally described as the collision-avoidance or path planning problem. The algorithms for collision-avoidance can be categorized into two groups: those concerned with finding collision-free "optimal" paths at the planning level [Lozano-Perez and Wesley 1979], and those concerned with avoiding collisions at the control level in real-time [Freund and Hoyer 1984]. The path-finding problem is formulated as follows: Given a robot in a cluttered environment, find a continuous collision-free path from the start position to the goal position. The problem of avoiding collision may be reduced to detecting an imminent collision and then taking the appropriate action.

Robot languages are the means by which a user can request a certain action to be performed by a robot. The languages can be classified into three main categories [Carayannis 1982] : manipulator-oriented languages, object-oriented languages, and task-oriented languages. Manipulator-oriented languages are independent of the task to be

performed and as such, provide the user with a general set of manipulator positioning and orienting procedures Object-oriented languages provide major advantages in the programming and maintenance of software for complex systems Tasks-oriented languages allow the user to specify a set of robot actions, not in terms of manipulator moves, but rather in terms of intended tasks These task-oriented languages generate a sequence of manipulator actions based on a high level goal specification from the user In this last case, therefore, the user does not need to specify the sequence of manipulator actions required to achieve a given end result, but rather, the end result itself A more complete survey may be found in [Michaud et al 1986a]

Chapter 3 Workcell Development

This chapter details the philosophy of the proposed IC assembly workcell The hierarchical control architecture and the robot programs are presented. The database requirements, architecture and implementation are detailed. Finally, the results of experimentation with the multi-robot workcell are discussed.

3.1 The Control Architecture

From a programming point of view, the system consists of a single 'master' module called the SEQUENCER, and five 'slave' modules· three for manipulating (PUMA, ECUREUIL, XY Stage), one for sensing (VISION), and one knowledge element (DATABASE). Figure 3.1 shows the architecture of the inter-process communication. Physically, they are distributed over the two VAX 750 hosts; a third host is used by VISION for data acquisition and display. The following modules execute on the first host. The SEQUENCER module coordinates all workcell activities by sending and receiving messages. Messages are passed using a set of virtual communication channels on the Ethernet called 'links', one associated with each module. The communication sub-system guarantees that messages will be properly delivered. In most cases, the messages sent are directives to perform a series of tasks. Each slave module is then expected to respond with an 'AOK' reply when the tasks are performed. The PUMA module is responsible for manipulating the DIP chip and the dice array as outlined in Figure 1.6. The ECUREUIL module ma-

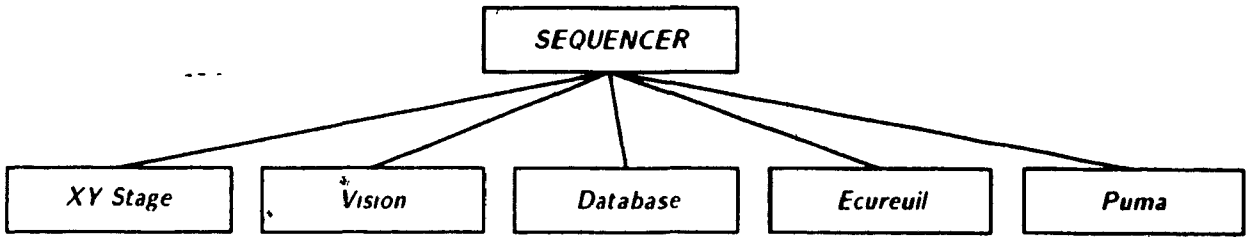

Figure 3.1 Control architecture

nipulates the die The XY Stage module presents the die for camera inspection under the microscope

On the second host, the VISION module locates the exact position and orientation of the die This information is passed to the SEQUENCER module which incorporates it in the command to the ECUREUIL module The DATABASE module stores all information concerning the task to be performed

The message passing is based on a Session Layer developed for inter-process communication in distributed robotics workcell [Gauthier et al 1985] This makes it easy and convenient to establish communication channels and send messages between various modules on the same or different hosts Message passing is strictly vertical: the slaves do not exchange messages. Each message is associated with and triggers a set of pre-defined elementary tasks called a *task block* For example, the module PUMA initializes the robot when it receives the message WARM START (see Table 3.1).

Module Name	Message Name	Associated Tasks
PUMA	WARM_START	Initialization without teaching new positions.
VISION	LOCATE	Find the die position and return it
ECUREUIL	FETCH_TOOL	Fetch vacuum tool.
XY STAGE	MOVE_MIC	Move the die under the microscope.
DATABASE	UPDATE	Update the desired database entry.

Table 3.1 Sample messages sent by the SEQUENCER and the module tasks associated with them.

The control architecture used for the IC assembly has been first developed for a Hybrid Circuit repair demonstration, which is presented is Section 3.7.

3.1.1 Robot Programming

At the beginning of this project, RCCL had been installed on the PUMA 260 robot only. Therefore a set of C interface routines called MROUTINES were developed by the author to control the ECUREUIL robot using its native IRL language. The author was responsible for the ECUREUIL programming

A fine/coarse approach motion strategy is used for the robot motions. When moving from point A to point B, the displacement is performed in three steps. First, the robot performs a fine motion at low speed in the desired direction to a small distance away from point A. Second, the robot does a coarse motion at relatively high speed, and at a safe height (defined as a height where no obstacle exists) to a position close to point B. Finally, a fine motion at low speed is done to point B in the desired direction. Programming the ECUREUIL robot is relatively straightforward, except for object or tool frame motions which are not supported by IRL. A typical sequence of operations to move from point A to B is:

```
speed(slow)
move(approach_position_A)
speed(fast)
move(approach_position_B)
speed(slow)
move(position_B)
```

The PUMA 260 does not present this complication since RCCL supports arbitrary reference frames. A second problem with the ECUREUIL robot involves the two distinct workspaces of joint one, as shown in Figure 3.2.

When the robot has to move from one workspace to another, the robot must stop in the

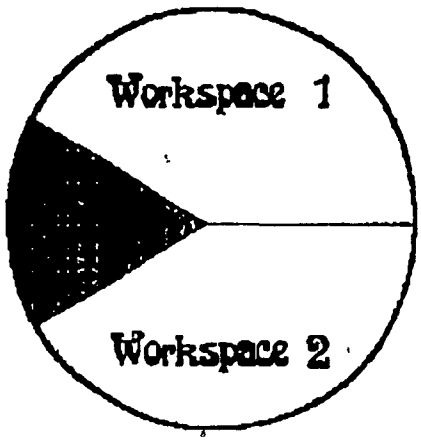

Figure 3.2 ECUREUIL robot workspace

common workspace (shaded area in Figure 3.2) and a special command must be sent.

3.2 Database Requirements

The author's main contribution to the workcell environment is the development of a robotic database which would provide a standard way of arranging, storing, and accessing information, in a real-time manner. After much discussion, it became apparent that the database database should support the following objectives:

(a) Increase the consistency of the software written in the laboratory by promoting the development of standard robot and image processing packages. This is achieved by defining a set of models of the workcell environment, such as frames (or specific data structure) associated with the tool racks and their tools.

(b) Simplify the planning of robot trajectories, by providing an approximate three-dimensional model for every object i.e. a bounding volume or combination of volumes [Lozano-Perez and Wesley 1979].

(c) Simplify the programmer's task, by providing "high level" aids such as the translation of symbolic descriptors, eg. TRAY, into physical locations in the robot world.

(d) Aid in the synchronization of concurrent tasks, by tracking key aspects of the robot world via the definition of an appropriate state vector Data driven program operation could then be based on the state vector contents stored in the database For example, when a state variable called JIG takes on the value OCCUPIED, the operation MOVE BOARD might be enabled This State Table representation would also be useful for performing simulations. Off-line programming and simulation are becoming more important in many industrial settings, because together they allow modifications to be tested before changes are made on the shop floor

(e) Store expert knowledge which may be required for domain-specific tasks, such as details about the grinding or welding tasks to be performed

3.3 The Database Architecture

It was decided to concentrate on the modeling and state representation issues, outlined in the previous section Note that the term "database" development used in the context of this work radically departs from its traditional meaning in the following ways

- A relatively small amount of information needs to be managed.
- Information is represented in a symbolic way, as in an expert system
- The information to be kept can be divided into three categories First, there is information that does not change during the lifetime of a manufacturing process, such as the graphical representation of the finished product Second, there is information that does change at the beginning of every assembly cycle, such as the serial number of the workpieces Finally, there is information that changes during the assembly process such as the position of a workpiece as it makes its way through the workcell. This category of information is represented as state variables which are used to monitor and synchronize the different activities of the workcell.

- The distributed nature of robotic applications means that the updating of information about the robot world, as represented by the State Table, is performed by many sources

- The store and update operations must be performed within a *real-time* context (Most robotic tasks, such as 'pick and place' motions, require on the order of seconds to execute)

The database has two parts the Dictionary and the Environment The Dictionary is a pool of templates or models for object definitions Each object is indexed by type, name, and is described by a set of attributes Attributes might be geometric properties of rigid bodies such as bounding volume, positions in world coordinates, or relative positions of components of an object An attribute must be designated *static* if it does not vary with time as robot activities take place, otherwise, we say it is *dynamic* One dynamic attribute is position, and another might be whether a component is present or not on a printed circuit board as in a repair task The motivation for distinguishing static attributes from dynamic ones will be clear momentarily

The Dictionary itself actually has two parts the *System* part, and the *Application* part The *System* part contains information about standard elements in the robot world such as robots, cameras, XY-stage, and feeders, while the *Application* part describes objects particular to the user's application, such as hybrid IC boards, capacitors, etc The same Data Base Manager will provide the user interface to both parts From the user's point of view, there is just one Dictionary

As for the Environment, it contains all the information required to completely describe the environment of the robot world. Thus, the environment can be thought of as a combination of database and State Table All objects must have *unique* names (identifiers). Information is classified as either static or dynamic, depending on the attribute designation in the Dictionary. Sharing of common data among several instances of the same object

type is the motivation behind distinguishing between static and dynamic attributes For example, there may be a set of PCBs with various solder joint faults, but all of them share the same physical size and component layout Thus, the presence or absence of a fault is a dynamic attribute, but the size and layout are static ones. This sharing of static information makes the implementation of the Environment more memory efficient, but it is transparent to the user Note that the Environment, in its initial stages, is meant to aid the *real-time coordination* of distributed robotic tasks This is in contrast to more complex knowledge representation systems for *planning*

An important issue that arises when dealing with a database implementation in robotics is whether there should be a direct connection between sensors and the database to provide for automatic updating [Wiederhold 1984], this is what we are presently contemplating More precisely, we associate a separate 'slave' process with each sensing and manipulating element in the workcell, these slaves operate under the direction of a single workcell 'master' (process) The master is responsible for all synchronization, based on the state of the robot world (Environment) However, it is left to the slaves to update the Environment as operations are completed

At this point, a representational framework has been defined. Using a microscope as an example, information is kept in the following form:

object name	*MIC1*
class of object	*microscope*
what the object is a part of	*WORLD*

As well, there are a set of attributes associated with the object, such as:

class	*position*
description_of	*MIC1*
qualifier	*joint_coordinates*
data	*46 75 98 90 34 87*

The object name *MIC1* is specified by the user, and must be unique. In this way, a name is

sufficient to identify a particular object Note that the description of the object is actually represented by a set of attributes eg *position* When an object is first added to the database, the name must be associated with a particular class. The class is used to index into the Dictionary to create a particular instance of the model i.e. object plus attributes, with the name supplied by the user

In order for the various system and user processes to interact with the database in our robotics environment, the following minimal set of commands was defined

- init database() This creates the Database Management process, and sets up the Environment and Dictionary

- load database(filename env, other_filename dic) This loads the Environment and Dictionary parts of the database to perform initialization Note that the user is free to have multiple env and .dic files in any directory for running different experiments

- save_env(filename env) This saves the current contents of the Environment in the specified file, and makes possible two features Experiments can be continued after a halt, and also a limited form of error recovery can occur, if the Environment is periodically checkpointed i e saved with a time stamp Note that an equivalent feature is not required for the Dictionary since it is edited off-line

- print_env() This prints the current state of the Environment.

- add_object(object_name, object_class) This adds a new object entry to the Environment, with the specified name and class.

- add_attribute(class,description_of,qualifier,data) This adds a new attribute entry to the Environment, as specified by the various fields. Note that the 'description_of' field must be a proper object name such as *MIC1*.

- delete object(object_name) This removes an entry from the Environment, all its attributes, and all its sub-objects and their attributes. In the example, MIC1 is a sub-object of the WORLD

- retrieve(object_names, attributes, current_values) This is how the user fetches information from the Environment about the current state of the robot world

- modify(object name, attribute, new_value) This is how the user updates information (attributes) in the Environment This function might return the old values of the attributes, although its usefulness is questioned. The old values may sometimes be useful in order to quickly recover from an error On the other hand, recovering from failure often means re-doing a set of tasks, either partly or completely, with some modifications This is a research topic for an expert system [Gini 1983]

We expect that more commands will be added in order to perform specific operations in the future, as our database needs become better defined. For example, a state re-construction feature, via rollback to a previous checkpoint, would make the database more reliable, but adding the periodic checkpointing (via save_env) would affect the speed of response Alternatively, the completion of each operation could be tagged with the current time, so that the state could be reconstructed from the time history of all the operations; most expert systems already perform this time stamping. The more complicated the recovery technique, the slower the system becomes

3.4 Database Implementation Considerations

Since the Database must be implemented, it makes little sense to explore an intriguing Data Model if it cannot be implemented efficiently. Many programming languages could be used, each with its own advantages and disadvantages, but none can serve as the general purpose language for database implementation. In the context of robotics, there

are three major issues to be considered First, data structures must be flexible to support a wide variety of information Second, the system must be reasonably fast such that accessing the database does not affect the real-time performance of the overall system. Third, robotics is a quickly evolving area and the database must be flexible If it cannot be easily changed and expanded, the system will not be able to cope with future requirements

We have explored three alternatives: two different Expert System implementations, using OPS5 [Forgy 1981] and PROLOG [Clocksin and Mellish 1984] , and the Abstract Data Type implemented in C (At this time, we did not consider any hybrid Data Models) The expert system work was motivated by several factors First, we anticipated the development of "knowledge elements" such as a 'Welding Expert' which would reason in fairly sophisticated ways about domain-specific problems With this in mind, it seemed wise to explore the expert system orientation OPS5 and PROLOG were chosen because they provide different ways of representing and inferring knowledge OPS5 has also become the standard programming language in our laboratory for developing expert systems C was selected because it is representative of many high level languages which offer powerful data structures suitable for Abstract Data Type C is also the standard language for robotic work in our laboratory

The basis of comparison of the three implementations was the following: speed (servicing user requests), implementation considerations (ease of representation, overall program size), and ease of 'maintenance' (degree of modularity, ease of making changes)

3.4.1 OPS5 Implementation

OPS5 is a general production system programming language developed at Carnegie-Mellon University for AI, Knowledge-Based Systems and Cognitive Psychology. It is a forward chaining programming system, which means that problems (or queries) are 'solved in a bottom-up way. Because OPS5 keeps track of all data change dependencies, it is a data-driven or event-driven language. The OPS5 architecture consists of

three basic components of a rule-based production system. global database, rulebase, and interpreter The DB and rulebase are referred to as Working Memory and Production Memory respectively Assertions and facts in Working Memory are represented as vectors or attribute-value pairs. This representation makes it easy to store objects along with feature vectors or their relations to other objects The rules stored in OPS5 consist of a condition part called the left-hand-side (LHS), and an action part called the right-hand-side (RHS) The LHS consists of a set of conditions element(s) that, the interpreter tries to match to the elements in working memory during the recognize-act cycle. If all the conditic ns elements are satisfied, the RHS is executed A "conflict " occurs when the LHS's of several rules are matched simultaneously. In such an event, OPS5 provides two conflict resolution strategies, Means-End Analysis(MEA) and LEXicographic ordering (LEX), to select a rule to fire

An object is described in our OPS5 implementation as

```
(object     ^name        MIC1
            ^class       microscope
            ^part_of     WORLD
```

While its attributes are kept in frames that look likes this:

```
(attribute  ^class       position
            ^description_ofMIC1
            ^qualifier   joint_coordinates
            ^data        46 75 98 90 34 87
```

Here, the data is represented as a vector-attribute and the other features are stored as attribute-value pairs. An example of a rule in OPS5 to fire the "help" command follows:

```
(p help     condition elements to satisfied
            {( user_request ^string ⟪ he help ⟫) ⟨ input ⟩ }
→
            (remove ⟨ input ⟩)
```

```
(write      ' help ' (crlf)
            ' ch(ange) at(tribute) ' (crlf)
            'sh(ow) pa(rts) ' (crlf)
            'sh(ow) at(tributes) ' (crlf)
            'quit ' (crlf)))
```

More details concerning the use of OPS5 may be found in [Dill and Hong 1984].

3.4.2 PROLOG Implementation

PROLOG, with its top-down approach, is a backward chaining (goal-driven) Artificial Intelligence language It assumes some hypothesis(goal) and tries to prove it. It has a built-in depth-first backtracking mechanism that allows it to look at different hypotheses until one is proved true or it fails Like OPS5, (and every rule-based language), PROLOG rules contain a RHS and a LHS but with different syntax and searching mechanisms than OPS5 Programming in PROLOG involves.

- - Declaring facts about objects and their relationships

- - Defining rules about objects and their relationships

- - Asking questions about objects and their relationships

PROLOG is referred as a declarative language and therefore declaring facts is done in a very natural way. For instance, the previous information is kept like this:

```
object(MIC1, microscope, WORLD)
attribute(MIC1, microscope, coordinate, (46, 75, 98, 90, 34, 87))
```

This makes it easy to store and maintain a wide variety of information. This list structure is simpler than the OPS5 object structure. The structure of the rules is similar to OPS5. The "help " rule becomes:

```
command(help).-
        nl. write('help').
        nl. write('ch_at (change attribute)').
        nl. write('sh_pa (show parts)').
        nl. write('sh_at (show attributes)').
        nl. write('quit').nl
```

Unlike OPS5, there is no special strategy governing the firing of the rules They are fired in the order that the user has entered them This has the advantage that the flow of the search is easier to follow, but backtracking somewhat negates this General information about PROLOG may be found in [Clocksin and Mellish 1984], our particular PROLOG implementation, CPROLOG, is described in [Pereira 1984].

3.4.3 C Implementation

C is a general-purpose systems programming language. Although it is not as "high level" as the PROLOG, OPS5 and LISP languages, it still offers powerful structures for data abstraction which provide for easy data access and very fast data retrieval. No special interface is required to access the database from C user programs, which means less overhead The data structure in C looks like the following:

```
structure object{
        char name[80],
        char class[80];
        char part_of[80],
        structure attribute *attr;
        structure object *next;},
structure attribute{
        char class[80];
        char qualifier[80],
        char data[1024];
```

structure attribute *next;}.

Here, there is no need to carry the information in the attribute frame concerning ownership of the attribute position. In order to fetch more specific information such as the microscope position, which items are blue ones and so on, separate data structures could be added with specific sets of routines to handle them in the same way as the rest of the data The control of the database is done via a special set of C routines.

3.5 Database Tests and Results

Before presenting the results of our experiments, some background remarks are appropriate The three database versions tested were all implementations of the Environment part of our proposed design However, the sample database consisted of 12 objects and 7 attributes per object (without static and dynamic distinctions), describing an IC die and its components

The host used in all tests was a VAX 11/750 running the Unix 4 2BSD operating system. The PROLOG interpreter [Pereira 1984] is written in C, but the OPS5 interpreter [Forgy 1981] is written in a dialect of LISP which itself is written in C. Thus, we anticipated that regardless of the specifics of our implementations, OPS5 would be slowest, C would be fastest, with PROLOG in-between

In order to compare the three languages, 3 test situations were used to examine 7 database operations. First, the database was tested with the sample database and the host running under 'light' load This means load values below 1.5 as measured by the UNIX command "la" (A shell script was used as a background task to monitor the load during the experiments. For the second situation, the database was made 5 times bigger to contain 60 objects, and the load was the same as before. For the third, we examined the case of the original database and heavy load: this means load values between 3.5 and 5.5. These situations are summarized in Table 1, and the results are shown in Tables 2.

Load	Size of database
Light	x 1
Light	x 5
Heavy	x 1

Table 3.2 Summary of the test cases

Operations	OPS5	PROLOG	C
Start database	57	6.2	0.53
Add object	11	0.11	0.02
Remove Object	16	0.14	0.007
Change attribute	11	0 14	0.008
Show parts	8.1	0.40	0.02
Show attribute	9 6	0.13	0.010
Incorrect user command	4.8	0.11	—

Table 3.3 Execution time for light load with normal size database (in seconds)

Operations	OPS5	PROLOG	C
Start database	158	10.6	0.70
Add object	18	0.15	0.03
Remove Object	97	0.31	0.06
Change attribute	15	0.22	0.02
Show parts	14	0.84	0.03
Show attribute	15	0.13	0.05
Incorrect user command	5.0	0.13	—

Table 3.4 Execution time for light load with a database 5 time bigger (in seconds)

3 and 4. All timing information is expressed in seconds to two significant values (where possible). The values shown are averages over multiple trials.

These results must be considered as qualitive rather than quantitative, since the load could not be stabilized exactly during the experiments. This is primarily due to the sharing of the host with other users, and network traffic since the minicomputer hosts within CVaRL are linked by an Ethernet local area network.

Operations	OPS5	PROLOG	C
Start database	80	42	1.3
Add object	27	0.24	0.04
Remove Object	100	0.28	0.01
Change attribute	32	0.42	0.05
Show parts	35	1.1	0.12
Show attribute	35	0.42	0.18
Incorrect user command	20	0.36	—

Table 3.5 Execution time for heavy load with normal size database (in seconds)

The operation "Start database" deserves special mention, since different activities must be performed for the C database when compared to the PROLOG and OPS5 databases For the latter pair, the system must create a C interface and then a special environment to run them. This is clearly not the case with C where only the loading of the information is required The C version differed in a second way too, since it incorporated a sophisticated pattern-matching user interface. For this reason, there are no entries in the tables for "Incorrect user command"

The results show that the C database is the fastest, followed by the PROLOG and then the OPS5 versions. OPS5 is in fact almost two or three orders of magnitude slower than the C version. The PROLOG version is generally less than one order of magnitude slower. Therefore, it becomes obvious that OPS5 is inappropriate for real-time or quasi real-time applications.

The effect of expanding the database most seriously affected the OPS5 version of the database, as demonstrated by an average relative increase[2] of 1.3 for the time response. In the case of PROLOG, there was an average increase of just 0.6; only the "show parts" operation was significantly slowed. Most operations of the C version were increased by a factor of 3.5. The fact that the C version was most affected is due to two

[2] Given a new value $N2$ and a reference value $N1$, the relative increase is obtained as $\|N2 - N1\| \div N1$.

reasons. Firstly, data is stored in linearly linked lists; this means that the operations will be slow when the database is large. The second reason is a consequence of the UNIX multi-tasking environment which uses a 'round-robin' scheduler. As soon as an operation cannot be executed within a single time slice (approximately 0.01 seconds, the same order of magnitude for most operations of the C database), it must incur the context-switching overhead of the scheduler, and the execution time will greatly increase

Operating the database under heavy load also affected performance. The OPS5 data in Table 4 shows an average increase of 2.6 over Table 2, while the PROLOG data shows an average increase of just 2.3. The C version was most sensitive to load, with an average relative increase of 9.3; again this is mostly due to the multi-tasking scheduler as described above

3.6 Database Discussion

From these experiments, we conclude that C and PROLOG come out far ahead of OPS5 when speed is of primary concern, at least for our UNIX environment. Of course, where other implementations of OPS5 and PROLOG are available, this may not hold true. For example, the originators of OPS5 have released OPS83 [Forgy 1985], written in C (for UNIX), which is a blend of the classic production system model and the emperical model (used by conventional programming languages). This is supposedly 30 times faster than the OPS5 dialect written in LISP. However, in fast evolving areas such as robotics and image processing, the ease of maintenance becomes an important issue. In fact, in the case of a research laboratory like our own, this is paramount. It is difficult to persuade researchers to use a database which cannot be easily adapted to their needs. This demands an implementation in which the addition of new options to the database can be done easily. This is a major limitation of the C database. Although it was built in a modular way, adding new functions requires understanding of the complex data structure, which is not the case with PROLOG or OPS5. Moreover, the source programs written in PROLOG and OPS5 are easy to understand and consists of far fewer lines. As a matter of fact, the PROLOG

database is the shortest one (200 lines of versus 365 lines for the OPS5 version and about 1300 lines for the C version) and it is also the easiest to understand. However, C remains the language of choice for our robotics work because the Artificial Intelligence programming languages do not provide the required features for sensor integration, interrupt handling, and intensive numerical calculations for which they were not intended.

The implementation described above has been extended to provide most of the intended database framework. A hierarchy of object models forms the Dictionary: models are identified simply by their class. The attributes of an object are also part of the model, but there is no 'static' or 'dynamic' distinction made at this time

In summary, C and PROLOG are best for speed, and PROLOG and OPS5 are best for ease of maintenance This makes PROLOG a promising choice It offers a very easy and natural way to build data structures, and is easy to maintain Like OPS5, PROLOG relieves the user of some low-level database management utilities, such as time stamping, which would be useful for future work in error recovery. In addition, partial information can be kept very easily [Genesereth 1985] by using dummy variables for "don't care" conditions. This is sometimes required for real time applications such as robotics when operations must be performed with incomplete information. As soon as more information is available, PROLOG can then instantiate these conditions.

However, in an industrial environment where the emphasis is on efficient data retrieval, the Abstract Data Type would be the prefered data model. Such a database would be less flexible than one designed around objects and attributes, but an industrial database typically evolves incrementally: existing modules are rarely changed, just new ones are added.

Although the PROLOG results were promising, the choice was made to pursue the C language option. The response time of the PROLOG version remains fast when the size of the database become large, but there is no guarantee that this will be true when additional rules are added. In fact, if these rules are not implemented carefully, they may

drastically affect the performance of the database [Genesereth 1985]

It also became clear that a C database could be incorporated more directly into our plans for a high level knowledge-based robot programming environment Furthermore, when the database is tied to an image processing system, a large number of transactions (e g. store, retrieve operations) may be required, which will degrade the performance of the PROLOG and OPS5 implementations more seriously than the C version

3.7 Workcell Experimentation

In order to evaluate the workcell philosophy, an alternate experiment concerning Hybrid Circuit Repair(HCB) was performed HCB was chosen instead of IC assembly for convenience since the image processing was not yet completed for IC assembly In this demonstration, the task involved consists of cutting traces on a printed circuit board The operation of cutting traces often happens in the electronics industry because of late changes in the production cycle This means that after the printed circuit board is fabricated, it must be partially re-designed to reflect the availability of new components or perhaps a design change As a result, certain traces must be cut, and new ones laid to accommodate the new design We decided to abstract from this real industrial need by imagining a robotics workcell dedicated to the cutting of traces, fed by an inspection workcell 'upstream' which would label each one with a simplified bar code to indicate which traces must be cut. The author's contribution to this work was two-fold, the ECUREUIL programming and the vision processing

We began with an idealized situation. the printed circuit board has just two traces. The bar-code reading phase is performed in parallel with the repair phase At any time, two boards are in play in the workcell. one in a jig under the microscope, and the other under repair in a separate repair jig.

The repair phase is cooperative: one robot performs the grinding, while the other checks the continuity of the trace after the grind, using a special tool. Once the

repair is complete. both robots change tools the grinding robot uses a gripping tool to move the repaired board to an output tray. while the checking robot uses a separate gripping tool to move the other board from the camera jig to the repair jig

A user interface to the workcell was provided by the following commands 'calibrate-puma', 'calibrate-microbo', 'execute-demo', and 'quit'

The experiment also successfully demonstrated our object/attribute database methodology An OPS5 database 'process' was defined to store the following

(i) static information about where to grind each trace on the board (displacements in X and Y from an origin on the jig for the 'begin' and 'end' positions of each grind).

(ii) static information about where to electrically check the continuity of each trace (displacements in X and Y from an origin on the jig)

(iii) dynamic information about the bar code of each board, which indicated which trace on the board must be cut (the repair)

As part of the workcell initialization, the static information was fetched from the database and sent to the robot control processes The dynamic information about the bar code was updated by the vision system (via the workcell master) OPS5 database was used because its development was completed at the time of this demonstration

The message passing paradigm is a sound general approach to distributed control. Separate slave modules were associated with each manipulating, sensing, and knowledge (database) element, under the control of one master module. Messages therefore precisely defined the interface between the slaves and the master, and this made it possible to develop all modules concurrently. In a sense, the message set associated with each slave constitutes a kind of 'menu' of allowable activities.

From a control point of view, the message passing paradigm also permits concurrency of module tasks to be specified in a simple way; the master just sends a set of messages to the various modules (without waiting for replies). When module tasks must be synchronized, the master simply waits for the appropriate set of acknowledgements

Note that a high network load does not affect the nature of the control carried out via message passing, it simply changes the speed at which activities in the workcell occur However, in the extreme case, timeouts associated with the message passing facility will take effect

Also, the message passing overhead associated with the Ethernet and TCP/IP is insignificant when compared with the actual execution time of the task performed upon receipt of a message eg moving a board, or grabbing a frame Tests have shown that the average end-to-end delay for most messages (10 characters or less) is of the order of 0 1 seconds, while a typical robot task is of the order of 10 seconds.

The use of a common high level robot language makes the robot programming easier and makes the programs more transportable when the language is available on a different robot. As a matter of fact, the RCCL environment has been extended to the ECUREUIL and the "cutting trace" demonstration has been successfully adapted to operate both robots using RCCL.

In the rest of this thesis, the use of an Abstract Data Type kind of database is intended unless otherwise specified.

The necessary tools to control multiple robots, and to fetch/retrieve/update information concerning the environment have been developed, and can be used to perform IC assembly. The image processing algorithms required to identify the position and orientation of the rectangular IC die are presented in the next chapter.

Chapter 4 Evaluation of the IC Die Position.

Corners, triangles, rectangles and circles are common man-made features in electronic assembly (e g PC boards, capacitors, resistors, IC die) This raises the interest in developing special algorithms to detect these features This chapter addresses the problem of precisely locating the position of an IC die in an image

In IC assembly, as well as in hybrid circuit assembly and repair, one of the most commonly used high level features of the analyzed image are ninety degree corners and quasi-perfect rectangles The problem of finding a die in a picture may be abstracted to that of finding the rectangle enclosing the die in a two dimensional image In most cases, local features, such as corners, are extracted and then matched to the overall image [Bolles and Cain 1982] A technique often used to find complex shapes is template matching. For instance, to find a die, a rectangle of a known size may be matched over the desired image This technique presents some drawbacks: it is sensitive to noise and requires knowledge about the exact size of the object. As shown in Section 4.2.1, it is also possible to approach the problem in a different way by mapping the information from the space domain to a parameter domain via the use of the Hough transform [Ballard 1981] However, this approach is computationally very inefficient

In order to provide a robust and general rectangle finder, a group of algorithms has been developed to merge lines, extract corners and rectangles, and rate the "*quality*" of the rectangles selected. Throughout this chapter, the word "line" is meant to denote a line segment, i.e. line of finite length.

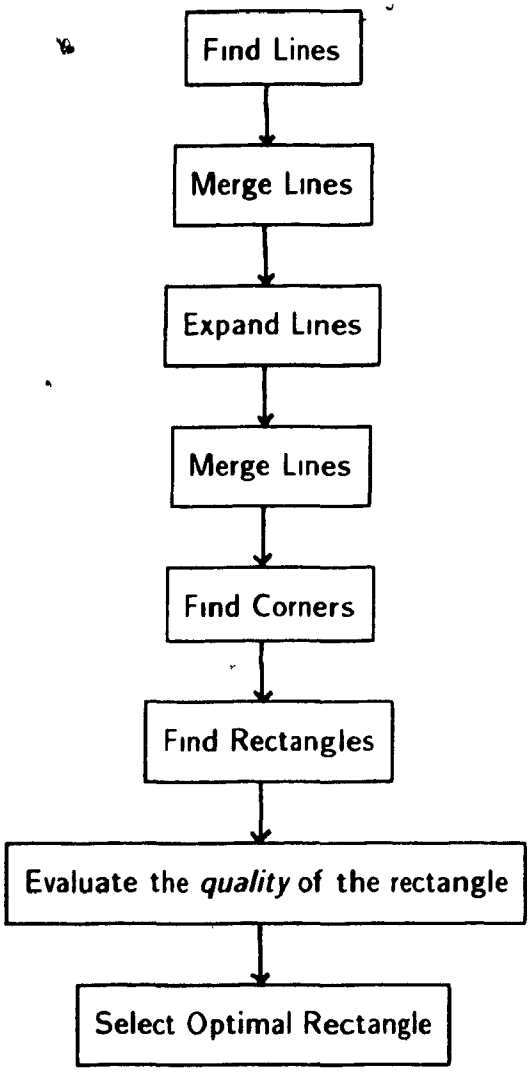

Figure 4.1 Approach used to detect a rectangle

The proposed approach is presented in Figure 4.1 First the lines are found in the image. Then, all the lines that are close in distance (i.e near by) and in orientation are merged. Subsequently, the lines are expanded in length in both directions. Then, another merging operation takes place. The purpose of this sequence of operations is to extract the corners from the images. The expansion is necessary because the natural corners of the scene are the parts most likely to be blurred by noise. Corners are detected and groups of corners that can form a possible rectangle are matched together. Finally the optimal rectangle in the image is selected.

When the die's position is found, standard pattern recognition techniques such

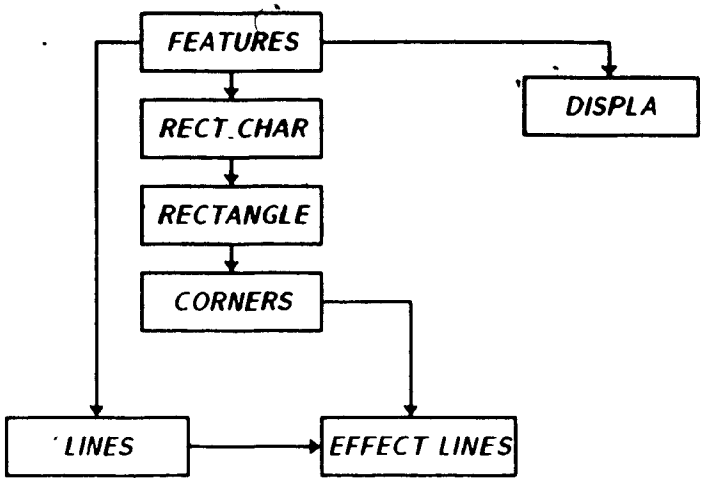

Figure 4.2 Data structure for image processing

as average, variance and moments of the die image may be used to identify the physical orientation of the IC die from the rectangular outline The use of template matching may also be done once the exact position of the rectangle is known In the case of a square, a maximum of four trials are required to identify the orientation of an asymmetric pattern.

4.1 Data Structure

In order to store all the information required for the image processing in a coherent way, the features of the images are arranged in a hierarchical data structure as shown globally in Figure 4.2 and detailed in Figure 4 3 This data structure has greatly ease the implementation of the rectangle finder by providing practical representation for lines, corners and rectangles. In Figure 4.3, the solid lines with arrows indicate pointers to the head of a list of a structure. An arrow pointing to three dots indicates that the same structure is repeated. When a declaration type is followed by "*", it indicates a pointer to a structure of the declaration type

At the bottom of the hierarchy are the lines. They are kept in a structure called *effect_lines* and are represented by their end-points. Every *effect_lines* structure consists of a pair of points and a pointer to the next *effect_lines* structure. The lines are then

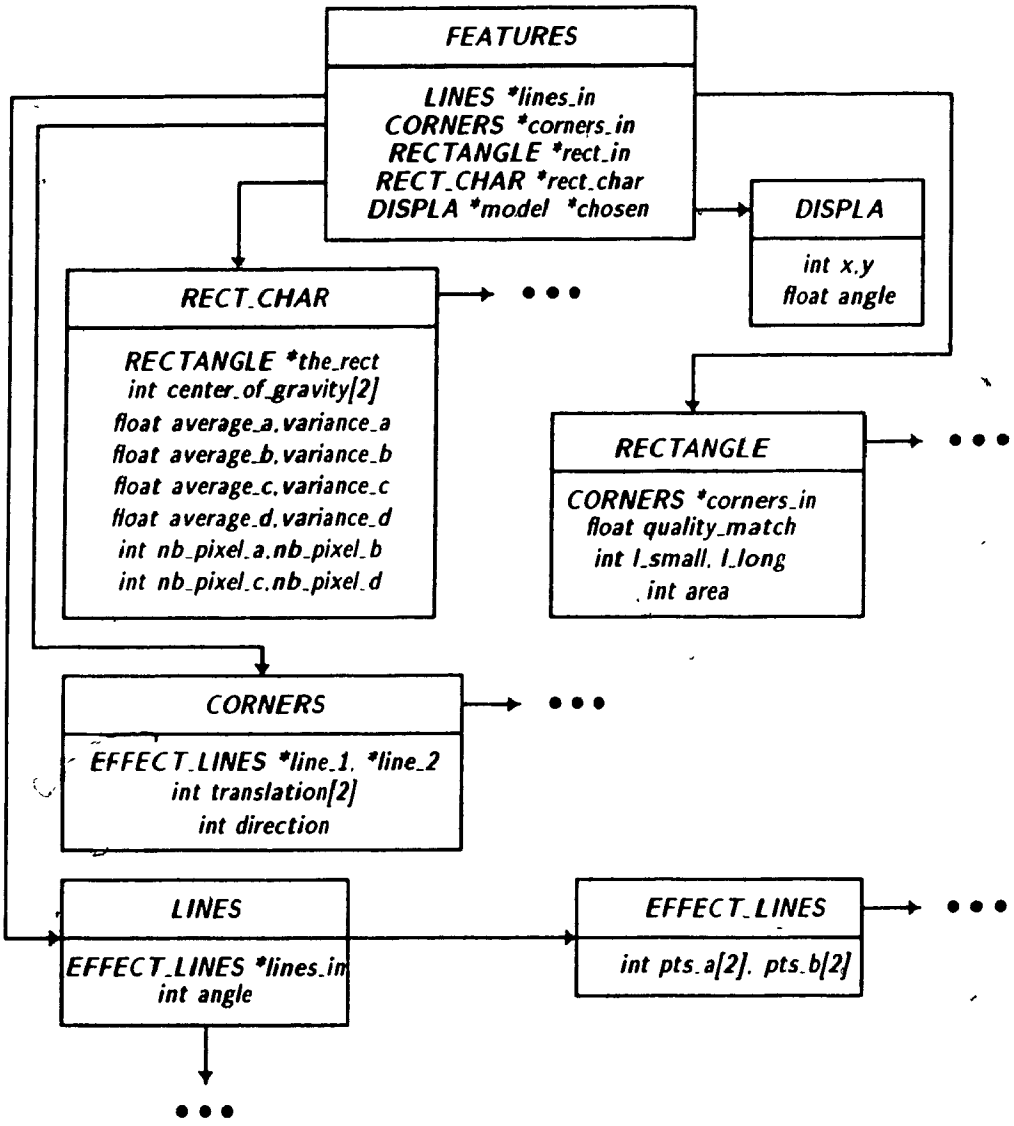

Figure 4.3 Detailed data structure for image processing

grouped together according to their orientations in a *lines* structure. This structure has an orientation angle and a pointer to an *effect_lines* list. Lines of the same orientation are kept in the same list. In addition, the x and y coordinates of the endpoints in the *effect_lines* structures as well as the *effect_lines* structures themselves are kept in a specific order. If a *lines structure* has an orientation (angle) smaller in absolute value than 45 degrees, the x and y data for the endpoint with the smallest x coordinate is kept in *pts_a* of the *effect_lines*

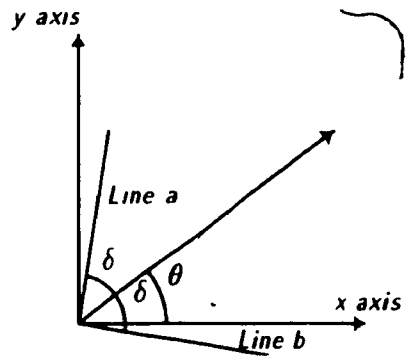

Figure 4.4 Defining the direction of a corner

structure, the other data becomes pts_b Moreover, the *effect_lines* structures are stored in increasing order of the y coordinate data in *pts_a* If the absolute orientation of the *lines structure* is greater or equal to 45 degrees, the data for the endpoint with the smallest y coordinate is kept in *pts_a* The other data becomes pts_b. The *effect_lines* structures are then stored in increasing order of the x coordinate data in *pts a* A maximum of 181 *lines structures* varying from minus ninety degrees to plus ninety degrees may be kept This ordering is done to ease the identification of overlapping (or quasi-overlapping) lines At a higher level there is the *corners* structure, which consists of the pointers to two lines, the corner position and its direction. As shown in Figure 4 4, the "direction" of a corner is defined to be the bisector of the angle 2δ between the two lines, the angle measured with respect to the x axis is θ. The angle θ may vary between plus or minus one hundred eighty degrees.

The next level contains the rectangle representation which consists of a list of *corners'* pointers, the area, the length of the two different sides of the rectangle and a quality match factor, which will be described in Section 4 2 7 All the structures presented above are general and do not contain any specific information related to the image. In the upper level, called the rectangle characteristics level, more customization may be added in order to store information for directing the low level image processing. In order to do so, the image may be split in four smaller rectangles of equal size and a set of standard tests may be applied to them. The current structure for this level is: a pointer to a rectangle, a field

for the center of gravity, and 12 fields for the number of pixels, averages and variances of the four sub-rectangles. This data structure may be avoided if template matching is used. Since it is intended to find the position and orientation of a die, a structure is reserved (DISPLACE) to keep the value of the displacement in x and y as well as the rotation in world coordinates. Finally, all the above structures are embedded in a FEATURES structure which contains pointers to the head of the structures presented.

4.2 Algorithms

There are already many algorithms which perform line and corner detection. But not all are computationally efficient or robust in the presence of noise. The following subsections describe the existing algorithms used as well as specific procedures developed *de novo*

4.2.1 Line Detection

To explore existing line detection schemes, experiments were performed with three approaches: Hough transform [Ballard 1981] , chain-coded detection [Duda and Hart 1973] , prediction/verification paradigm [Mansouri et al. 1985b]

The Hough transform consists of mapping the problem of finding lines in the spatial domain to that of finding clusters in the parameter space As Figure 4.5 shows, a line can be characterized by two parameters, namely ρ and θ, where ρ is the normal distance to the origin and θ the angle defined by the normal with respect to the x-axis.

In effect, the Hough transform maps a given point(X,Y) in the spatial domain into a curve in the parameter domain. Mathematically,

$$\rho = Y * sin(\theta) + X * cos(\theta) \qquad (4.1)$$

The interesting property of the Hough transform is that points on the same line in the space domain become curves in the parameter space with a common intersection point. Therefore,

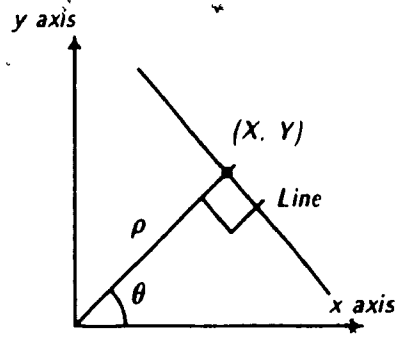

Figure 4.5 Hough transform

by keeping the result of the parameterization in an accumulator array, we can detect lines of a given slope and intercept They are represented by high counts in the accumulator array The advantage of this technique is that it can be generalized for any type of shape [Ballard 1981] However, there is no information concerning the exact location of a line in the image, and the Hough transform does not discriminate between lines made of connected or disconnected points Moreover, the Hough transform is computationally inefficient on a VonNeuman architecture machine because for every treated point in the image, a value of ρ must be computed for every angle under investigation For instance, investigating an image with steps of one degree over a scale of 90 degrees results in approximately 90n additions, sine and cosine evaluations, and 180n multiplications in the program, where n is the number of pixels considered

Another approach is based on chain-code. A chain-code is formed by the sequence of consecutive integers that compose a closed contour The chain-code line detection techniques are of two types: those fitting a small straight line segment by moving it over the chain-code, and those taking advantage of the coherency of a digital arc. In the first category, the boundary segment is approximated by a series of piecewise linear segments [Duda and Hart 1973] . Similarly, it is possible to move a fixed size window over the chain-code, fit a line segment using least squares technique, and keep the segment providing it matches within a certain threshold. On a Vax 11/750 supporting Berkeley UNIX 4.2,

finding the lines takes 20 seconds. Other techniques based on chain-code representation is the coherency-based technique [Hung 1985, Wu 1982]. This technique takes advantage of the fact that in a chain-code a straight line is represented by a periodic pattern of two directions Thus, it is possible to determine the straightness of a digital arc by looking at the chain-code itself without any distance and angle calculations. The main disadvantage of the chain-code based techniques is that they are very sensitive to noise because they generally use only local information in their search

Another line detector investigated makes use of the Hypothesis Prediction/ Verification Paradigm (HPVP) According to some observations, the program forms a hypothesis and then examines its validity The observations come from the results of the Sobel edge detector operator This operator is made of two masks, one in the horizontal direction, the other in the vertical direction. Thus, the direction produced by these masks gives a hypothesis about the direction of the line. Knowing the position of the observed pixel, it is possible to verify if a line of a specific length exists by putting a line of a known size centered on the desired pixel. Then all the pixels on the line having an orientation consistent with hypothesized line raise the probability of having a line at that position. This technique is much faster than the Hough transform, and performs well when there is noise in the image, which is not the case of the chain-code based techniques. However, it has the drawback of finding many lines that are close to the one desired. This makes a pruning, or a merging algorithm compulsory On a Vax 11/750 supporting Berkeley UNIX 4.2, the computing time is about 5 minutes

Other techniques exists for finding lines such as relaxation techniques [Zucker et al. 1977] . Here, the objective is to attain a consistent orientation labeling that propagates via local computation. This process is also computationally inefficient. In general the relaxation technique takes at least as much time as the Hough transform. However, both of these techniques can be favourably implemented on a parallel architecture machine.

From this review, both the HPVP and the moving window chain-code based algorithms were promising choices for our application. However, due to its robustness to

noise and relatively fast speed, the HPVP algorithm was chosen. It provides a solid basis for developing higher level feature extracting algorithms

4.2.2 Line Merging

The process of merging lines together consists of comparing in orientation and distance separation of lines taken two at a time. Respecting pre-defined selection criteria, the result of merging two lines is the longest line that can be built with their endpoints. The merging process, if not done carefully, may generate undesired results Because the process is repeated recursively until completed, the merged line may deviate badly from the desired result as shown in Figure 4 6.

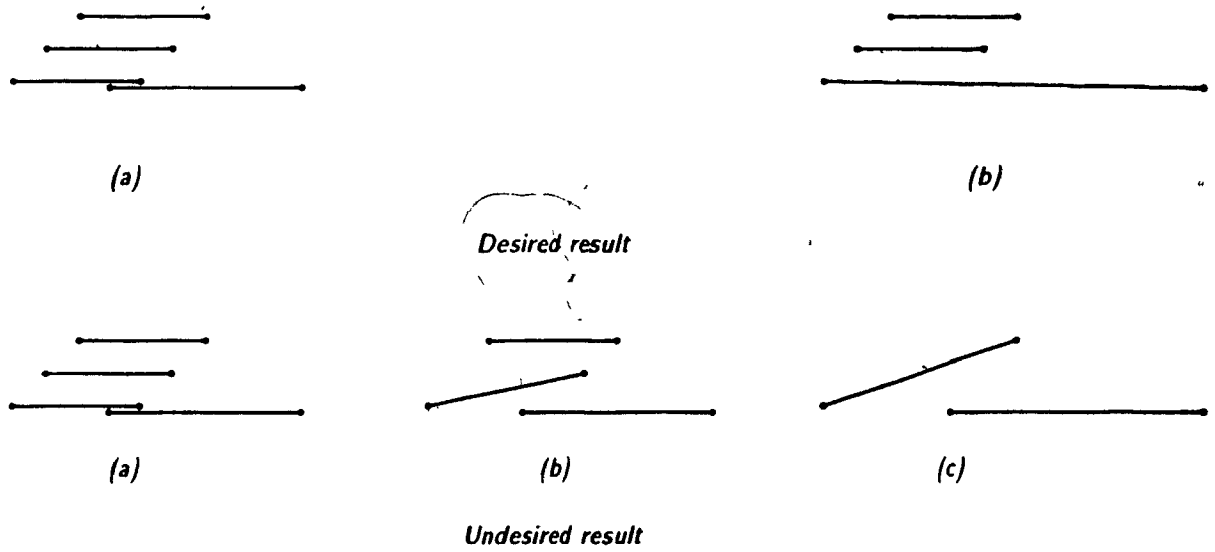

Figure 4.6 Problem of the merging process

The essence of the merging algorithm proposed consists in comparing the differences in directions and the distances between the lines at multiple resolutions. The use of multiple resolutions is mandatory in order to avoid the problem of Figure 4.6. The merging process is done in two stages. In the first stage, a very small tolerance on the orientation

is allowed in the comparison. If these lines overlap or have their endpoints almost touching (i.e. at a distance smaller than a given *tolerance*), the two farthest endpoints are kept and the others are discarted. This process is repeated at a coarser resolution. During the first pass, a high resolution (very small distance), is selected. Thus, only the lines that overlap or almost are merged. In the second pass, a lower resolution is selected in order to merge line segments which are further apart but still having the same orientation. In the second stage, the same process is repeated with a large tolerance on the orientation. This last merging operation removes lines that have slightly different orientation but that are close to each other.

The merging algorithm takes a line (to be called line A), and tries to merge it with each of the remaining lines (called line B) in the list. The decision of merging two almost overlapping lines is based on the line orientations and the Distance Between the Lines (DBL). The line orientation condition is implemented by taking a group of lines with the same orientation from the same group or groups that fall within a selected tolerance. The DBL is evaluated by taking the minimum distance between the endpoints of each line. If this distance is below a specified threshold, the lines are merged. The DBL is calculated using equation 4.2:

$$\frac{(X \times M_1) + B_1 - Y}{(M_1^2 + 1)} \tag{4.2}$$

where X and Y are the endpoint coordinates of a point in the first line, and M_1 and B_1 are the slope and the y-intercept of the equation of the other line. In order to avoid merging two lines having the same slope and y intercept, but which are widely separated, a special strategy has been developed. The strategy consists in tracing an elliptical area around the line to be merged. The use of an elliptical area has been chosen to merge overlapping lines having a relatively large separation while a smaller distance is tolerated when they do not overlap. Because of the line ordering mentionned in Section 4.1, only one of the endpoints of line B needs to be tested. The elliptical area is obtained by using the endpoints of line A, and evaluating the sum of the arcs made by both endpoints of line A to the chosen endpoint of line B, as shown in Figure 4.7. Line B is merged with Line A providing equation 4.3 is

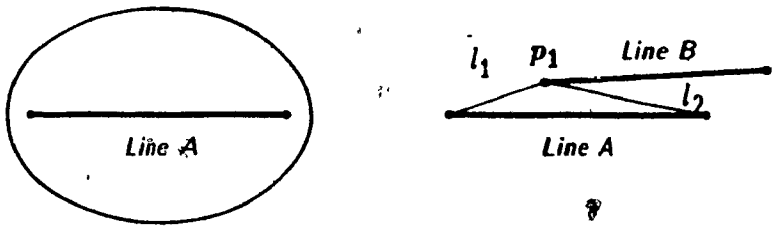

Figure 4.7 Elliptical area of tolerance on merging

satisfied.

$$l_1 + l_2 - l_t < threshold \tag{4.3}$$

where l_t is the length of Line A.

The complete algorithm for merging lines of the same direction is the following:

— For every group of lines:

— For every line in the group (let's say line A):

— With all the next lines in the group of (let's call each of them line B).

1) Evaluate the distance between lines.

2) If the distance is within the acceptable tolerance, proceed to the next step.

3) Verify if the endpoints of the two lines are close enough. If so, a merging operation is done. Otherwise, verify if the first endpoint of line B is inside the elliptical area of tolerance of line A.

— If a merge operation has been done, re-evaluate the lines' direction, re-order the lines, and then restarts the process.

In the second stage when a larger tolerance is accepted on the line's orientation, the merging is performed slightly differently. The merging is done among the lines of acceptable orientation, and the step one is removed

4.2.3 Line Expansion

In order to enhance the detection of corners, it is very useful to expand the lines However, expanding short lines may induce the detection of false corners since small lines are often due to noise This problem may be avoided by expanding the lines that are considered long enough Thus a measure of what is long enough is required A solution is to evaluate the average length μ and the variance σ^2 of the lines length L_k

$$\begin{aligned} \mu &= E\{L\} \\ &= \frac{1}{n}\sum_{k=1}^{k=n} L_k \\ \sigma^2 &= E\{(L-\mu)^2\} \\ &= E\{L^2\} - E^2\{L\} \end{aligned} \tag{4.4}$$

where n is the number of lines

With these two values, it is possible to perform line expansion to a selected set of lines only To select the set of lines, the value of σ subtracted from μ, may be used to evaluate the minimum line length accepted. Better results are obtained if 2 σ is used. Another issue to be resolved is the selection of the line expansion factor. The program developed allows two different ways of doing it First, the lines may be expanded by the same number of pixels at both endpoints. Second, it is possible to expand the lines by a certain percentage. For instance, if twenty percent is selected, then both endpoints are expanded by ten percent. This technique has proven its usefulness for images of objects with rounded or chipped corners.

4.2.4 Corner Detection

Corners represent one of the most useful features for image segmentation. In the following, we will present five published algorithms, which cover the different classes of techniques to extract corners.

Chain-code based corner finders generally consist of studying variations in the gradient directions [Kitchen 1982] or chain-code values [Freeman and Davis 1978]. For instance, in the latter, corner detection is done by moving a line segment along the chain-code sequence. The segment is defined by its end points in this chain-code sequence. As the segment is moved from one chain node to the next, the angular variations between the successive positions are used as smoothed curvature measures of the chain-code sequence. A succession of positions that have curvature measures around zero indicate a straight line. Thus, a corner may be detected by looking for the position of points in small neighborhoods surrounded by zeroes in the smoothed curvature list. This technique has the drawback of being sensitive to noise since it is based on the acquisition of the chain-code sequence.

A different approach has been proposed by Paler et al. [Paler et al. 1985]. Their method is based on one property of the median filter, which is its insensitivity to features of dimensions smaller than half the size of the filter window. These features are re-created by subtracting the median filtered image from the original input image. They assume that a grey level function of the image may divided in three regions: object, edge and background region. Thus, by moving a median filter window over the image, a corner may be detected when a significant change in the count of pixels coming from one of the three regions. The advantage of this algorithm is that it requires no edge detection, no edge thinning and no chain-code. However, Paler's method is restricted to simple noise-free images.

An interesting approach to detect lines has been developed by [Fang 1982] and presented in [Gu and Huang 1984]. The method is based on the assumption that corners are the points where the direction of the edge changes by a large amount. The procedure

consists of applying a Sobel operator, storing the values $\triangle X$ and $\triangle Y$ from the two masks, and the gradient intensity of the edge-map at this point, $\Delta = \sqrt{\triangle X^2 + \triangle Y^2}$. The edge-map is then thinned. Thereafter, the corners are picked up from the thinned edge points if the edge intensity exceeds a specified threshold, and if the corner strength is a local maximum. The corner strength is a quantification of how much the edge intensity changes at the examined point. This technique, based on very local features, has the drawback of being very noise sensitive.

The uses of B-Splines has been proposed for corners detection [Medioni and Yasumoto 1986] . B-Splines are used to produce a curve that approximates or approaches a given set of points [Mortenson 1985] The technique consists of taking the result of a very precise edge detector [Medioni and Huertas 1985] , and then fitting a B-Spline to it. The idea is to let the points "move around" their original position and measure their displacement. Points "moving a lot" and having a high curvature are marked as corners.

Finally, corners may be found via template matching [Bolles and Cain 1982] . However, this technique requires that the image possesses corners which exactly match the masks. This restricts the template matching technique to a very small subset of images

4.2.5 Algorithm Developed

The corner detector developed is somewhat a mirror image of the lines detector developed by [Fang 1982] . As explained previously, Fang finds the corners via local features and then matches lines between the corners. To do this, he assumes knowledge about the image is available. Our corner detector proceeds in the other direction: the lines are found first, then the corners. The algorithm developed does not require any particular knowledge of the scene.

A corner is defined as two lines almost touching with a specified aperture α. In Figure 4.8, line 1 and line 2 are candidates for a corner with an aperture $\alpha = 90$ degrees. The term "almost" is used because perfect corners are rarely found in IC images. Rather, the

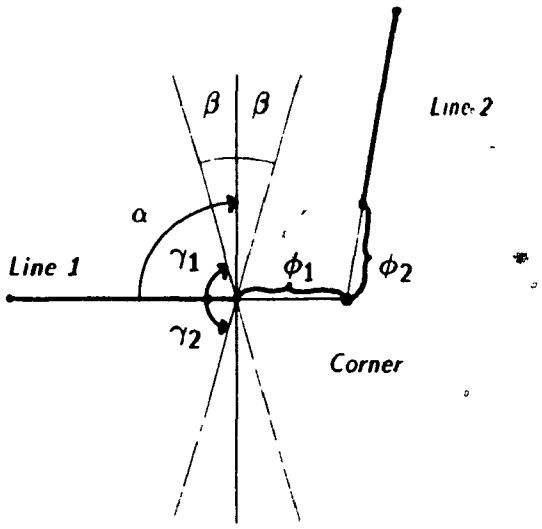

Figure 4.8 Tolerances accepted for the detection of a corner

natural corners are degraded by highlights, distortions and noise to form rounded corners in the image Both lines are extended to calculate the corner's location and the lengths ϕ_1 and ϕ_2 Two lines almost touch if the distances ϕ_1 and ϕ_2 are small enough. This is the first condition to assert a corner

In order to satisfy the second condition, the angle between line 1 and line 2 must be in the range spanned by γ_1 and γ_2. The testing angles, γ_1 and γ_2, define the set of acceptable angles

$$\gamma_1 = \alpha - \beta, \gamma_2 = \alpha + \beta \tag{4.5}$$

β is the tolerance on the aperture γ_1 and γ_2 must be between -90 and 90 degrees For this reason, if γ_2 exceed 90 degrees, γ_2 is set to β - α. For instance, in Figure 4.8, $\gamma_1 = 90 - \beta$ while $\gamma_2 = -90 + \beta$. The tolerance β is used to eliminate the effect of quantization. When the angle α is very different from 90 degrees, which may occur if the camera is mounted with an angle relative to the scene observed, a new situation arises. The group of lines that will be acceptable to form corners will be at plus or minus α degrees from the investigated group of lines. Thus, they must be considered in the corner searching process. To do so, the group of testing angles is divided in two sets: one at plus α and one at minus α from the line tested (i.e Line 1 in Figure 4.8). For each set, the situation

described in Figure 4.8 may occur, which makes four testing angles per group. As a result, eight angles are needed to discriminate all the possible cases in our general corners finder.

In the algorithm, only half of the line orientations need to be examined since they are matched with the other half. For every group, the orientation is extracted and orientations at plus or minus β degrees of difference are evaluated.

For each line from the first group, determine whether it forms a corner with any of the lines in the other groups that respect the orientation restriction. For every line in the acceptable groups, the position of the corner is evaluated and a test is done to check if it is an acceptable corner or not. Thus, evaluating the position of the corner is reduced to finding the intersection of two lines, which is trivial. Special care must be taken for vertical lines (infinite slope).

The algorithm to extract the corner is the following.

— For the all groups of lines between -90 and 0 degree

— Take a group of lines. (let's say group A) Now we will be looking for a group of lines at α degrees plus or minus β.

— Evaluate the testing angles for the lines which may form corner(s).

— For every line in the group of lines A do:

— For every group of lines in the proper range of orientation

—1) Take a group of lines. (let's say group B)

—2) For every line in the group B

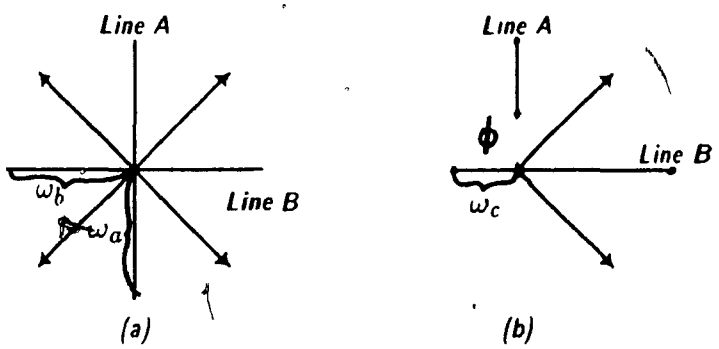

Figure 4.9 Examples of multiple corner points

—3) Evaluate the corner's position between the line from group B and the line from group A

—4) Verify if this corner position is close enough to the line segments to be a valid corner. If so keep it/them

In addition, the system must be able to extract multiple corner points and divide them into different corners. This may occur after the expansion/merging process previously performed and is shown in Figure 4.9

The decision of keeping, one, two or four corners per pair of lines is based on how close the lines are. If the two lines cross the corner position by a predefined threshold, four corners are kept. For instance, in Figure 4.9a, Line A and B have overlap ω_a ω_b that are greater than the threshold, thus the four corners are kept On the other hand, if only one of the lines passes over the corner position, like in Figure 4.9b, and if the other line is at less than a distance ϕ from the corner position, two corners must be kept. However, when no lines overlap the corner position, the two lines under investigation must be at a distance less than ϕ of the corner location. The lines with arrows are the directions of the corners.

The *validate and save* corner process goes as follows:

1 – Evaluate the shortest distance between the two endpoints of line A and the corner location Save the distance and set *flag_a* to indicate which endpoint is the closest

2 – Verify if line A overlaps the corner. If so, set the distance to zero, set the *flag_a* to indicate overlap, and evaluate ω_a

3 – Evaluate the shortest distance between the two endpoints of line B and the corner location. Save the distance and set a *flag_b* to indicate which point is the closest.

4 – Verify if line B overlaps the corner. If so, set the distance to zero, set the *flag_b* to indicate overlap, and evaluate ω_b.

5 – Set the distance value to the largest of the value found in step 1 and 3. (This will be zero if both overlap the corner)

6 – If the distance is under a predefined threshold save the corner(s) according to the information in *flag_a*, *flag_b*, ω_a and ω_b

When processing real images, false corners may be found. However, these false corners are generally clustered around a true corner. Most of these false corners are removed by taking the average in the neighborhood of the corner positions. Any remaining false corners are eliminated by the rectangle detection process described in the next section. An interesting property of our corner finder is that it can extract corners with angles different from 90 degrees. This may be useful for the detection of non-rectangular polygonal shapes.

4.2.6 Rectangle Detection

Operators to detect low level features have been heavily investigated, but according to a recent survey [Michaud et al. 1986a] , few are applicable for finding rectangles.

Aside from the approach developed here, only three other techniques have been found in the literature

1 - Template matching

2 - Hough Transform.

3 - Features Matching(Shape Matching) [Bolles and Cain 1982]

The two first techniques were previously explained in Sections 4 2.1 and 4.2 4 and are applicable to extract rectangles. Features Matching consists of extracting local features of the object from the image and then trying to instantiate them to match the complete object

Our rectangle detection method does something similar to local features matching with the addition that it makes use of global information from the image It performs the extraction of all the possible rectangles from the image based on the corners which have been generated. It is assumed that the corners are kept in increasing order of direction with aperture α of approximately 90 degrees The procedure works as follows. First, a corner is taken from the list and the three directions that complete the rectangle are computed. The corner chosen is labeled as corner *"a"*. Subsequently, the other corners that are at 90, 180 and 270 degrees from the corner *"a"* are labeled *"b, c,* and *d"* as shown in Figure 4.10.

Second, all the corners of type *"b"* that match corners of type *"a"* are identified and for every corner pair of type { *a, b* } a rectangle structure is created. Thus, if the corners b_1 and b_2 are identified, two rectangle structures will be created: $\{a, b_1\}$ and $\{a, b_2\}$. In order to match a corner of type *"b"* with a corner of type *"a"*, two conditions must be fulfilled:

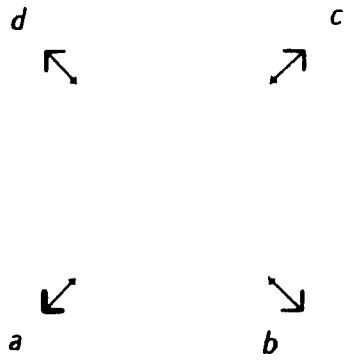

Figure 4.10 Example of rectangle labelling

1 – The line formed between the two corners called the Line Connecting Corners(LCC) must have a respectable percentage of its length covered by lines in the image. For instance, the LCC in Figure 4.11, 25 percent of the LCC from "a" to "b" is covered by the projections of lines l_1 and l_2. If the threshold selected is 30 percent, this corner would be rejected. The minimum distance between a projected line and the LCC must be within a threshold ϵ l_3 is too far away to contribute a projection.

2 – Each of the corners must be "visible" to each other For this, a tolerance is added to the corner aperture angle by enlarging it slightly beyond 90 degrees to permit the two adjacent corners to be "seen" from the first.

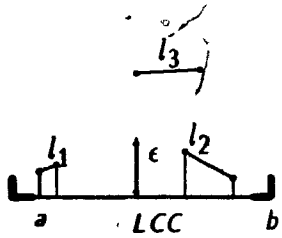

Figure 4.11 Evaluation of the line projection

These conditions guarantee that the corner matched is part of a possible rectangle.

After this step, a series of corner pairs, kept in rectangle structures, now exists.

The identification of corners of type *"d"* is now performed using the same approach. Corner *"c"* is omitted since it is not adjacent to corner *"a"*. For every corner of type *"a"* present in the existing rectangle structures, look for corners of type *"d"* that respect the conditions mentioned. For every corner pair of type { *a, d* } a rectangle structure is created. Thus, if the corners d_1 and d_2 are identified, four rectangle structures may be created: $\{a, b_1, d_1\}$, $\{a, b_2, d_1\}$, $\{a, b_1, d_2\}$ and $\{a, b_2, d_2\}$. These four sets represent four possible rectangles. It is also possible that a corner of type *"a"* matches only one corner or none. This gives rise to three possible situations:

1 – Only one element is present in the list of rectangle structures i.e. { *a* }

2 – Only two elements are present in the list of rectangle structures i.e. { *a, b* } or { *a, d* }

3 – Three elements are present i.e. { *a, b, d* }

When two elements are present, the same process used for finding corners of type *"d"* using corners of type *"a"* is repeated to find corners of type *"c"* using corners of type *"b"* (or *"d"* according to which corner was matched). However, when three elements are present, the program evaluates the possible position of the fourth corner from these three, and looks for corners that are in the hypothesis area, as shown in Figure 4.12. The evaluation of the position is easily obtained by taking the average position of the two corners of opposite direction i.e. *"b"* and *"d"*. Then, we evaluate the difference Δ between this new position and the corner *"a"*, add it to the average position found, and call this a corner of type *"c"*, see Figure 4.12. If no valid corner is found, the calculated position is kept as the fourth corner. For every corner of type *"c"* found, a new rectangle structure is created.

These steps for evaluating the fourth corner position are valid for any parallelogram. If only a corner of type *"a"* is found (case 1 above), it is then removed from the list

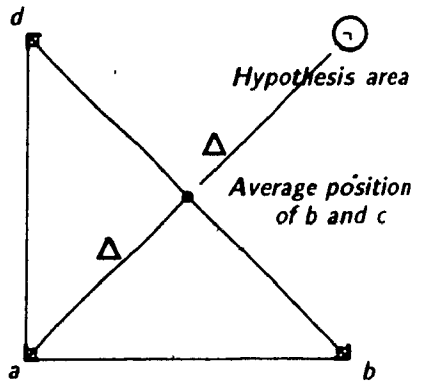

Figure 4.12 Evaluation of the fourth corner

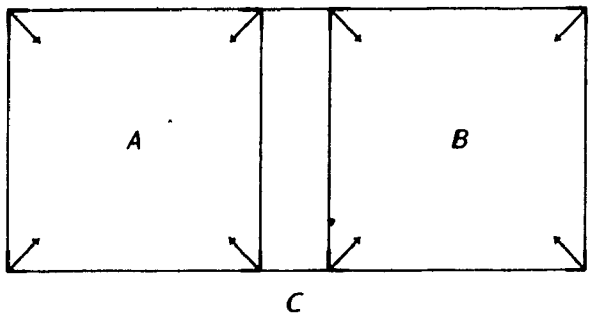

Figure 4.13 Example of a false rectangle

of rectangle structures since accepting rectangles based only diagonally opposite corners. i.e just corner of type "c", with corner of type "a" generates too many false rectangles in a complicated image. A false rectangle is defined as a rectangle created by the association of true (or false) corners that are not already part of other rectangles as shown in Figure 4.13. Rectangle A and B are real rectangles but the rectangle C enclosing them is a false one.

All the way through the identification of the possible rectangles, every corner matched to a rectangle is put in a special list in order not to be re-used as a first rectangle's corner. Without this precaution, the process would find the same rectangles over and over again.

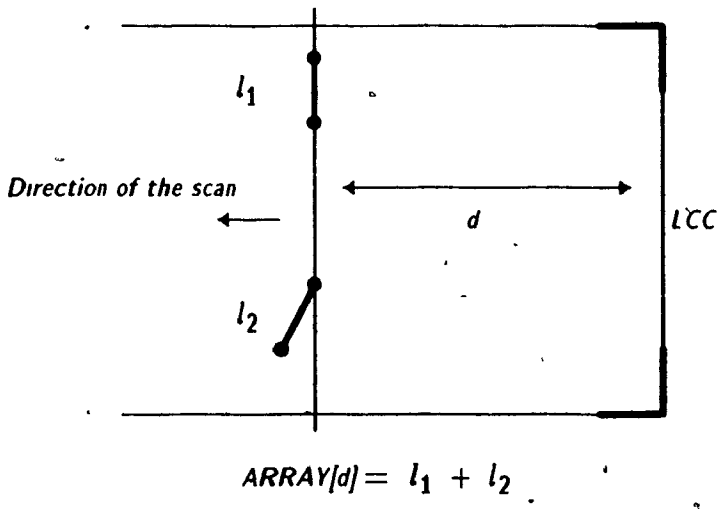

Figure 4.14 Example of a search for an accumulation of line segments

After the corner matching phase, all possible rectangles in the image are now in the list of rectangle structures. Nevertheless, many of these structures contain only two and three corners rather than four corners When three corners are present, it is considered that there is strong evidence for the presence of a rectangle, and the fourth corner, as estimated in Figure 4 12, is instantiated. On the other hand, when two corners are present, more investigation is required to check before instantiating a rectangle. In order to do so, we will look for an accumulation of lines quasi-parallel to the LCC This is performed by scanning the LCC in the direction of the possible corners. Every time a line of the same orientation of the LCC is encountered, a count equal to the line length is kept in a *linear array* at the position corresponding to the distance from the LCC, as shown in Figure 4.14.

A triangular mask of size "n" $f_{\Delta_n}(i)$ is used for smoothing the sequence of values in the *linear array*. They represents the accumulation of lines contributing to the scanning line at a distance d in the *linear array*. The sequence of values in the *linear array* is correlated with $f_{\Delta_n}(i)$. Its Fourier transform $\Phi_{\Delta_n}(\xi)$ is represented by a quadratic *sinc* function. In other words, the correlation of the sequence of values in the *linear array* with a triangular mask is equivalent to a lowpass filter operation in the frequency domain. Thus,

$$f_{\Delta_n}(i) = \begin{cases} n - |i| & \text{if } |i| \leq n; \quad n = 1, 2, \ldots, k \\ 0 & \text{elsewhere.} \end{cases} \tag{4.6}$$

$$\Phi_{\Delta_n}(\xi) = \frac{\sin^2(\pi \xi n)}{(\pi \xi n)^2} \tag{4.7}$$

The correlation of the linear array with the triangular mask $f_{\Delta_n}(i)$ leads to a smoothed array and is defined as:

$$\begin{aligned} SA_i &= f_{\Delta_n}(i) \times A_i \qquad (\text{where} \quad \times \stackrel{\Delta}{=} \text{correlation}) \\ &= \sum_{k=-n}^{k=n} f_{\Delta_n}(k) A_{i+k} \end{aligned} \tag{4.8}$$

Then, the magnitude and the position of the maximum value are extracted from the smoothed array SA. If the ratio of this value over the length of LCC is greater than a threshold, the two missing corners are instantiated.

4.2.7 Evaluation of the Quality of a Rectangle and Selection of the Optimal Rectangle

The rectangle of interest in a die image is the one that best surrounds the die. A characteristic of this rectangle is that the lines that are part of the die are enclosed by it. This observation may be generalized for any type of image in electronic assembly where the observed objects are of rectangular shape. At this point in the processing, there are many rectangles that have been found. Thus, an algorithm is needed so that only the rectangle(s) surrounding real rectangular object(s) or object feature(s) are stored. In order to do so, a measure of the *quality* of the rectangle is required. The sum of the projection of all the lines in the image that are inside the rectangle within a distance "d" from the LCCs defines the *quality* of the rectangle. LCC1, LCC2, LCC3 and LCC4 are the lines connecting the different corners. $l_1, l_{2a}, l_{2b}, l_3, l_{4a}$ and l_{4b} are the lines in the image. This is shown in

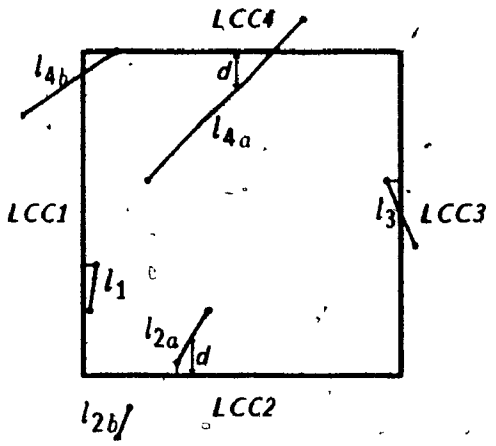

Figure 4.15 Evaluation of the *quality* of a rectangle

Figure 4.15. This is done by creating a linear array for each length of LCC and marking it for each valid projection, as will be presented shortly.

A line in the image must satisfy certain conditions to be considered for projection. First, it must have almost the same orientation as the LCC. However, here again a tolerance is accepted on the orientation of the line. Second, the line must be within a distance "d" of the LCC. For instance, in Figure 4.15, line l_1, is completely projected while l_{2a} is partly projected on the LCC; l_{2b} does not contribute a projection.

In order to evaluate the projected parts of any line that has one of its endpoints within a threshold distance *d*, we calculate the length of the line starting at its *Closest Endpoint* from the LCC, up to the other endpoint of the line. If this other endpoint is within the threshold distance d, the complete length is used as the projection value. This is reasonable as long as the orientation of the line and of the LCC are similar.(see l_1 in Figure 4.15). However, when the tolerance accepted on the direction becomes large, like l_{4a}, using the complete length may cause a false rectangle to be preferred to a good one. This is why the tolerance between the orientation of the line and LCC must be kept small. If the other endpoint of the line is beyond the threshold (l_{2a} in Figure 4.15), the length is evaluated from the *Closest Endpoint* to the point intersecting the perpendicular from the LCC at a distance *d*.

Another situation that may occur is when the line tested crosses the LCC, e g l_3 in Figure 4 15 In this situation, the intersection between l_3 and the LCC3 must be evaluated Then, the length from this intersection point to the endpoint inside the rectangle is used A combination of the two previous situations can also happen, i e a line crossing LCC but having its interior endpoint at a distance greater than the acceptable threshold, like l_{4a} in Figure 4 15 Finally, a line may overshoot the LCC, like l_{4b} There, the distance from the intersection of the line with LCC1 to the intersection with LCC4 is calculated, and the projection operation is done only with the segment inside the rectangle

Before evaluating the *quality* of the rectangle it is necessary to slightly enlarge the rectangle to compensate for quantization problems such as having a line perfectly parallel to a LCC but out of the rectangle by one pixel!

The global algorithm to evaluate the *quality* of a rectangle and the details concerning the procedures used in the algorithm are described below

a - For every rectangle

b - Enlarge the rectangle

c - For each side of the rectangle

d - Evaluate the *quality* of the rectangle, as follows

e — Create a linear array of appropriate length for the LCC

— — For all lines almost parallel to LCC do

— — 1) Verify if the line is close enough i.e within the threshold distance 'd

— — 2) Verify if the line is inside (or partly inside) the rectangle

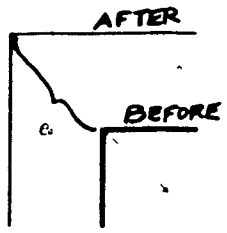

Figure 4.16 Position of the corner of a rectangle after the rectangle expansion

— — 3) If the previous conditions are fulfilled evaluate the begin and end positions for the projected line

— — 4) If the projection is valid, mark 1's in the linear array over the projection range

— — 5) Evaluate the ratio of the projections found with the length of LCC

— Sum all four ratios belonging to the same rectangle, and store the result

The first operation to be described is rectangle expansion The rectangle expansion is required in order to insure that lines which are a few pixels away from being inside the rectangle are considered during the rest of the process The corner expansion is done by using the projection of the unit vector in the inverse direction of the corner multiplied by a small distance e. (typically a value of 5) as shown in Figure 4 16 The direction of the corner is not modified during this process

Now comes the step to verify if a line is close enough To do so, the procedure checks if the line overshoots the LCC, like l_{4b} in Figure 4 15 This is done by calculating the angles between the LCC and the lines starting from the corners to the first endpoint of the line using the dot product. Using Figure 4.17, let v_1 be the vector made by LCC4 from endpoint A to endpoint B, while v_2 is the vector made from A to endpoint ep_2 of the line l_1 In this example, the angle defined between v_1 and v_2 is smaller than 90 degrees. Thus, the endpoint ep_2 of line l_1 is not modified On the other hand, the angle between v_3 and

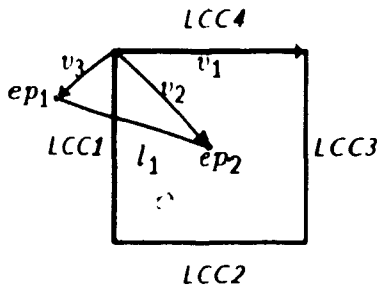

Figure 4.17 Dot product test

v_1 is greater than 90 degrees therefore, the endpoint ep_1 is replaced by the intersection of LCC1 and l_1 The angle ϑ is evaluated from the dot product

$$cos(\vartheta) = \frac{v_1 \bullet v_2}{|v_1| |v_2|}$$

$$\vartheta = cos^{-1}\left(\frac{v_1(x) \times v_2(x) + v_1(y) \times v_2(y)}{(v_1(x)^2 + v_1(y)^2)^{\frac{1}{2}} \times (v_2(x)^2 + v_2(y)^2)^{\frac{1}{2}}} \right) \qquad (4\ 9)$$

where ϑ is the smallest angle defined between v_1 and v_2

When the angle ϑ exceeds 90 degrees, the intersection of the line with the rectangle is evaluated to replace this endpoint during the evaluation of the projection If both endpoints overshoot, both endpoint values are adjusted for this computation Following this adjustment, both endpoints are inside the rectangle The distance between the endpoints to the LCC is then evaluated using 4 2 The distances of both endpoints from the LCC, a series of flags indicating which endpoint(s) is (are) overshooting the LCC, and which endpoints is (are) within the acceptable distance from LCC are evaluated These values are useful to evaluate the projection parameters, which define where to start and to stop filling the linear array If a line has one of its endpoints (or re-evaluate endpoints) within an acceptable distance, a test is done to know if the line is inside or outside the rectangle The cross product operator is used for this test This requires the construction of three vectors, as shown in Figure 4 18 The vectors starting from a defined corner of the rectangle (e.g. A in Figure 4 18) are evaluated as well as the one starting from this corner to the line's endpoint The cross products are evaluated as follows a vector formed by LCC1 ($vect_1$) is picked up and cross multiplied with the other vector formed by a LCC2 ($vect_2$) and with

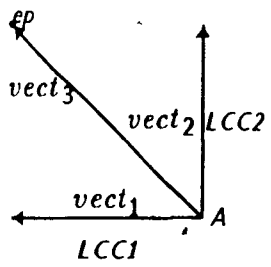

Figure 4.18 Cross product test

the vector $vect_3$ joining A to *ep* (one of the endpoints of the line tested) If the signs of the results are different, the endpoint is outside the rectangle This cross product test is applied to both endpoints If both are outside the rectangle, the line is rejected, if both are inside, another test is required If the line crosses the LCC, the intersection between the LCC and the line is evaluated, and the endpoint that was out of the rectangle is replaced by the intersection value for the *quality* evaluation of this rectangle

When both endpoints are inside the rectangle, the correct line length to be projected, as well as the position where the projection starts, remain to be found. If both endpoints are within an acceptable distance, the entire length of the line is projected on the LCC The beginning position to fill the linear array is defined as the closest distance from a chosen endpoint of the line to the LCC. On the other hand, if an endpoint is farther than this distance, a new endpoint is evaluated to replace it, as shown in Figure 4.19. In this figure, a new point NP, on the line P_a P_b, is evaluated at the threshold distance d.

$$NP(x) = \frac{d - d_a}{sin(\theta)} + P_a(x)$$
$$NP(y) = \frac{d - d_a}{cos(\theta)} + P_a(y) \qquad (4.10)$$

where $P_a(x)$ and $P_a(y)$ are the (x,y) coordinates of the point closest to the LCC. d_a, d_b are perpendicular distances to the endpoints from the LCC θ is the orientation of the line.

4.2.8 Vision Experiment and Results

The algorithms developed have been successfully tested on many different images. In Figures 4.20 to 4.32, the performance of the algorithms is shown.

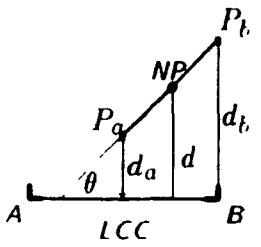

Figure 4.19 Evaluation of the new point NP

Figure 4.20 Image of an IC die

Figure 4.21 Result of line detection

Figure 4.22 Result of line merging for lines of the same direction that are less than 3 pixels away

Figure 4.23 Result of line merging for lines of the same direction that are less than 15 pixels away

First, from the image of the IC die (Figure 4.20), the lines are extracted (Figure 4.21) Then, the lines are merged together The importance of using two stages at multiple resolutions in the line merging are well illustrated in Figure 4 22, 4 23, 4 24 and 4.25. When almost overlapping lines of the same orientation are merged, almost no *deviation* effect is noticed, as shown in Figure 4.22 On the other hand, there are still many lines that could be merged If the merging is done with a tolerance on the orientation of the lines, the merging is much stronger but so is the *deviation* effect (Figures 4 24 and 4.25) Thus, the sequence of operations performed in order to obtain a meaningful image consists of merging lines of the same orientation that are 3 pixels away, then those at 6 pixels away, and finally merging lines with a tolerance on the orientation of 5 degrees that are less than 5 pixels away. Afterwards, the lines are expanded as shown in Figure 4.26

In the corner extraction process, two parameters must be taken into account:

Figure 4.24 Result of line merging algorithm for lines of orientation varying by less than 10 degrees and less than 10 pixels away

Figure 4.25 Result of line merging algorithm for lines of orientation varying by less than 10 degrees and less than 20 pixels away

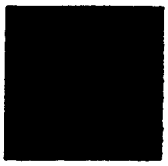

Figure 4.26 Result of line expansion All the lines are expanded by 25 pixels

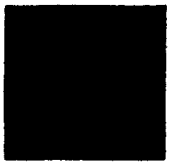

Figure 4.27 Detection of corners ($\alpha = 90$, $\beta = 10$ *and* $\phi = 10$)

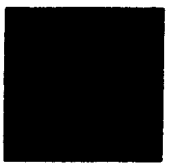

Figure 4.28 Detection of corners ($\alpha = 90$, $\beta = 20$ *and* $\phi = 10$)

Figure 4.29 Detection of corners ($\alpha = 90$, $\beta = 7$ *and* $\phi = 10$)

the tolerance (β) on the aperture (α) and the distance of the lines to the corner location (ϕ). In the image selected, the tolerance on the aperture does not seem to significantly affect the quality of the corner detection As long as the value selected remains within a certain range, the algorithm performs very well (Figure 4 27 and 4 28). However, the selection of the distance ϕ is more critical (Figure 4 29 and 4.30). Even though the detection of false corners generally has no effect on the final result, it strongly degenerates the time performance of the system Thus, one must be careful in the selection of this parameter. Further experiments would be required to select an appropriate value for another application.

Finally, all the possible rectangles are extracted (Figure 4.31), and the optimal rectangle is selected (Figure 4.32).

The worst case variation between the position of the true corner and the position of the detected corner, is 0.1 millimeter (10 pixels), when a maximum resolution of 50×

Figure 4.30 Detection of corners ($\alpha = 90$, $\beta = 7$ *and* $\phi = 20$)

Figure 4.31 Rectangles found when the corner detection is made at $\beta = 10$ *and* $\phi = 10$

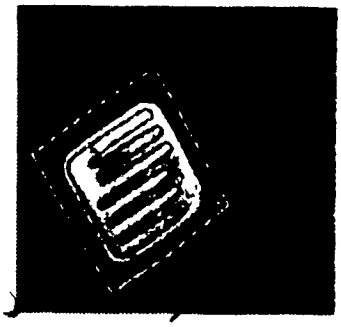

Figure 4.32 Rectangle selected for the die picture of Figure 4 20

is used on the microscope This happens because the HPVP line detection algorithm does not detect jagged lines Therefore, the image must first be smoothed using a three by three averaging mask This smoothing has varying effect on the precision of the algorithm. For instance, when the rectangle has an orientation of 45 degrees as in Figure 4.20, the precision of a result is 0.1 millimeter (10 pixels) The precision becomes 0 02 millimeter (2 pixels) when the rectangle has an orientation near 0 degrees

In order to enhance the precision, the following steps could be taken:

- A a smaller average mask could be used (two by two instead than three by three)
- Tuning the line detector to accept jagged lines.
- Using a chain-coded line detector

The program implementation involved about 5000 lines of source code in C language. The execution times of the different parts of the rectangle extraction process are

shown in Table 4.1 A VAX 11/750 running Berkeley UNIX BSD4.2 was used

There are three ways to improve the speed of the system First, many sections of the code could be programmed to be more efficient During the development phase of this project, the software was written to prove the feasibility of the approach rather than optimizing the program execution time We estimate that a speed improvement of 30 percent is possible Second, the use of heuristics can be made in the selection of the optimal rectangle Actually, we have presented a brute force method that takes into account many rectangles that could be ignored in the selection process To minimize this problem, only rectangles having an area close to the one desired should be considered Furthermore, the rectangles generated may have severe *assymmetry*, due to the tolerances used A test based on the lengths of the sides of the rectangles can be done to identify *asymmetric* rectangles Up to three minutes may be removed from the *selection of the optimal rectangle* processing time in this way Finally, special purpose hardware is essential for real-time image processing. The use of parallel processors linked with a commercial vision system making use of VLSI technology will make our approach usable in real-time. All the algorithms used and developed can be implemented on parallel architecture machines. Moreover, a very fast realization of our rather lengthy *optimal extraction process* could make it a general purpose higher level image processing step, just as the line detector is used in our approach.

The major limitation of the rectangle finding approach presented is that the duration of the merging and the extraction of rectangles changes according to the amount of information in the image, i.e. the number of lines obtained by the line and corner detection processes. The effect is more important for the rectangle extraction process which is exponential in the number of lines detected

Because of its generality and robustness, the rectangle detector developed may be applied to other categories of images such as hybrid circuits. Finding components on a hybrid circuit is a difficult task because of the noise and of the complexity of the image. The image intensity level and shape difference between a capacitor and a conducting path

Operator	Duration
Sobel	2 minutes
Line detection	3 minutes
Merge 3 pixels	22 seconds
Merge 6 pixels	7 seconds
Merge 6 degrees 6 pixels	1 minute
Expand lines	0.01 seconds
Merge 3 pixels	0 01 seconds
Merge 6 degrees 6 pixels	10 seconds
Extract corners 10 degrees, 10 pixels	8 seconds
Extract rectangles	1 minute
Select Optimal Rectangle	4 minutes
Total:	≈ 16 minutes

Table 4.1 Time required to perform the rectangle detection on a VAX 11/750 running Berkeley UNIX BSD4 2 The operators are listed in their order of appearance during the process

may be very small In order to avoid detecting many undesired rectangles, heuristics may be used. Only rectangles having approximately the desired area and width/length ratio may be kept. Our approach has been investigated on test images such as the one in Figure 4 33. Figured 4 34 and 4.35 show some very good results However, other rectangles formed by the lines inside of the welding area (bright area inside the rectangle in Figure 4.35), and the three other sides are a valid rectangles and may sometimes be selected. Thus, further processing is suggested to guarantee that the rectangle found completely encloses the capacitor, if this is required

Figure 4.33 Image of a capacitor on an hybrid circuit.

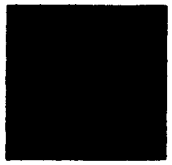

Figure 4.34 Rectangle selected for the capacitor

Figure 4.35 Rectangle selected overlayed on the capacitor

Chapter 5

IC Die Assembly

In the two previous chapters, a multi-robot environment with vision was described to perform IC assembly. In order to verify the completeness of the environment, an experiment was conducted with the workcell using just the PUMA 260 robot. The RCCL robot program was based on the code written for the multi-robot demonstration; only minor modifications were required. The purpose of this experiment was to determine the limits of the workcell to perform IC die assembly. The sequence of operations is the following:

- The workcell is calibrated.

- The PUMA 260 picks up the IC die from the die holder and puts it on the XY stage, using a vacuum tool.

- The XY stage moves the IC die under the microscope.

- An image of the IC die is grabbed.

- While the position of the IC die is evaluated, the XY stage moves the IC die back to the robot.

- The PUMA 260 picks up the die and places it on the IC chip by performing the appropriate corrections obtained from the image processing

The IC die, which has dimensions of 1.72×1.65 millimeters, is picked up under vacuum. In order to perform this operation without damaging the IC die, a special tool was developed, which mates with the multi-purpose end-effector of the robot. The tool consists of two parts. The first part is the tool body, which allows the robot gripper to hold the tool. The second part consists of a vacuum cylinder supported by a spring, which makes the tool compliant in the "Z" tool axis. The cylinder has also a small hole drilled through its upper part to provide for fast IC die release. The information concerning the types and dimension of the IC die were kept in the Database program. The RCCL database was used to store the locations of the different positions taught to the robot.

5.1 Results and Discussion

The experiment demonstrated that the image processing data structure chosen, and the software developed to control the robot, store and fetch the information and evaluate the position of the IC die, were appropriate tools for IC assembly, but additional problems remain to be solved.

- When the IC die is picked up, it often clings to the tool cylinder after the suction is turned off, due to electrostatic forces. Blowing air back into the cylinder to release the die was investigated. It turned out to be an ineffective solution for two reasons. First, our laboratory did not have the facility to accurately control low air pressure. Thus, the adjustment of the pressure was very critical. Second, the electrostatic forces themselves are unpredictable and would vary with the environment conditions as well as with the IC types.

- The precision of tool modeling was found to be of the same order of magnitude as the precision of the image processing, about 0.1 millimeter. Thus, better measurements are suggested for the tool modeling. A redesign of the suction tool, perhaps using different materials, might improve the electrostatic problem.

- The motion of the XY stage causes vibration so that the IC die moves by at least 0.1 millimeter. This vibration problem is inherent in the stepper motor drive. Obviously, a better approach would involve keeping the die on the robot suction tool for the image acquisition under the microscope. In the present workcell configuration, this was not possible.

- The use of a compliant tool causes two sources of inaccuracies. First, the compliance reduces the accuracy of the tool model. Unfortunately, the compliance is accompanied by small a displacement in the XY plane. The importance of this error has not been thoroughly investigated. The second cause of error is due to the fact that the robot performs its pick and place motions using compliance along the Z axis of the tool. Provided the tool Z axis is perfectly aligned the tool compliance should not cause unwanted motions. The present workcell environment makes these high precision adjustments difficult to set up. More versatile utility programs are needed. These drawbacks of using a compliant tool remains the price to pay for the gained flexibility.

Chapter 6 Conclusion and Future Work

This thesis has addressed the development of a multi-robot workcell with vision for IC assembly in the context of robotics research in the Computer Vision and Robotics Laboratory at McGill University. The literature was reviewed outlining some of the concepts and requirements for robotic workcells. The design of the robotic workcell was presented outlining the control architecture, robot programming, and database development. Three alternative database implementations were evaluated using OPS5, PROLOG and the C programming languages. A multi-robot demonstration was configured to validate the operation of the workcell. This was based on a hierarchical master/slave architecture using message passsing over an Ethernet Local Area network. Original image processing algorithms including line merging, corner detection and optimal rectangle finding were developed and tested. IC assembly experiments were performed to asses the handling of IC dice using the RCCL robot programming environment. The problems encountered were discussed.

Future extensions of this work are envisaged. First the database has to be enhanced in order to handle the geometric representation of objects. This work is already underway. Second, the image processing has to be made much faster. Rewriting part of the code and running it on a dedicated machine would be a beginning. Third, the tool modeling problem needs to be investigated. More accurate tool transforms and robot parameters are required for the high precision tasks involved in IC assembly. Fourth, the addition of error recovery is required in order to make this system more robust.

From this work we can say that using robot for IC dice assembly is a very challenging task The high precision required seems to dictate the use of hard automation However as the precision of the robots is increased, the tool modeling improved, and the execution times of computer vision algorithms are accelerated, the use of robots for IC manufacturing will become a fact of life

References

[Alami and Chochon 1985] Alami, R., and Chochon, H., "Programming of Flexible Assembly Cell Task Modeling and System Integration," *Proc IEEE Robotics and Automation* March 1985, pp 901-907

[Alford and Belyeu 1984] Alford, C., and Belyeu S "A Computer Control Structure for Coordination of Two Robot Arms," *Proc IEEE American Control Conf* 1984 pp 880-881

[Aristides 1984] Aristides, R., Representation for Rigid Solids Theory Methods and Systems' *Computing Surveys* vol 12 no 4 Dec 1984 pp 437-465

[Arnon 1976] Arnon D., *A Cellular Decomposition Algorithm for Semi Algebraic Sets* Tech Rep 353 Computer Science Dept University of Wisconsis Madison June 1976

[Baird 1977] Baird, M., "Image Segmentation Technique for Locating Automotive Parts on Belt Conveyors," *Proc of the 5th Int Joint Conf on AI*, 1977, pp 694-695

[Ballard 1981] Ballard, D., "Generalizing the Hough Transform to Detect Arbitrary Shapes," *IEEE Pattern Recognition*, vol. 13, no 2 1981, pp 111-122

[Barnes et al 1983] Barnes, D., Lee, M., and Hardy, N., "A Control and Monitoring System for Multiple-Sensor Industrial Robots," *Proc Int Conf on Robot Vision and Sensory Control*, 1983

[Bic and Gilbert 1986] Bic, L., and Gilbert, J., "Learning from AI New Trends in Database Technology," *IEEE Computer*, March 1986, pp 44-54

[Blais 1986] Blais, B., *Model-Based Inspection of Surface Mounted Circuits*, Master's Thesis, Computer Vision and Robotics Laboratory, Dept E E, McGill University, Montréal, Canada, 1986. In preparation

[Blum 1962] Blum H . "An Associate Machine for Dealing with the Visual Field and Some of its Biological Implications." *Synthetic Systems*, edited by M Kace. Flenum Press. New York, 1962. pp 244-260

[Blume 1984] Blume, C . "Implicit Robot Programming Based on a High-level Explicit System and Using the Robot Database RODABAS " *Proc First Robotics Europe Conf* . 1984. pp 156-171

[Bolles and Cain 1982] Bolles. R . and Cain. R . "Recognizing and Locating Partially Visible Objects The Local-Feature Focus Method " *Int Journal of Robotics Research*, vol 1. no 3. Fall 1982. pp 57-82

[Brice and Fennema 1970] Brice. C . and Fennema. C . "Scene Analysis Using Regions." *Artificial Intelligence*, vol 1. 1970. pp 205-226

[Brooks et al 1979] Brooks. R . Greniers. R . and Binford. T . "The ACRONYM Model-Based Vision System " *Proc of the 6th Conf on AI*. Aug 1979. pp 105-113.

[Bruno et al 1984] Bruno. G . Demartini. C . and Valenzano. A . "Communication and Programming Issues in Robotics Manufacturing Cells." *Proc IEEE Int. Conf on Robotics and Automation*. 1984. pp 361-367

[Carayannis 1983] Carayannis. G . *Controlling the PUMA Robot from a VAX*. Tech Rep TR83-3R. Computer Vision And Robotics Laboratory. McGill Univesity. Montréal. Québec. Canada. April 1983

[Carayannis 1982] Carayannis. G . *A Survey of Programming Languages for Intelligent Robots*. Tech Rep No 82-15. Computer Vision and Robotics Laboratory. McGill University. Montréal. Aug 1982

[Chand and Doty 1983] Chand. S., and Doty. K . "Cooperation in Multiple Robot Operation." *Proc. IEEE South East Conf. '83*. April 1983. pp 99-103

[Claybrook et al 1985] Claybrook B Claybrook A and Williams, J. "Defining Database Views as Data Abstraction" *IEEE Trans on Soft Eng*. vol SE-11 no 1. Jan 1985. pp 3-14

[Clocksin and Mellish 1984] Clocksin W and Mellish C. *Programming in Prolog*. Springer-Verlag. 1984

[CVaRL 1985] Laboratory. Computer Vision and Robotics. *Progress Report*. Computer Vision And Robotics Laboratory. McGill Univesity. Montréal Québec. Canada. Oct 1984- Sept 1985

[Date 1975] Date. C. *An Introduction to Database Systems*. Addison-Wesley. 1975

[Dill and Hong 1984] Dill. A and Hong. W. *Everything You Wanted to Know About OPS5 but Were Afraid to Ask*. TR84-5R. Computer Vision and Robotics Laboratory Dept E E . McGill University. Montréal. Canada. Aug 1984

[Dittrich and Lorie 1985] Dittrich. R. and Lorie. A. "Object-Oriented Database Concepts for Engineering Applications." *Proc IEEE COMPINT'85*. Sept 1985. pp 321-325

[Dittrich et al 1985] Dittrich. R. Kotz. A. and Mülle. J. "A Multilevel Approach to Design Database Systems and its Basic Mechanisms." *Proc IEEE COMPINT'85*. Sept 1985. pp 313-320

[Duda and Hart 1973] Duda. R. and Hart. P. *Pattern Classification and Scene Analysis*. John Wiley New York. 1973

[Dufresne 1983] Dufresne. P. "Representation and Processing of Production Rule Interactions in a Robot Planning System." *Proc of IFAC Artificial Intelligence*. 1983. pp 227-230

[Fang 1982] Fang. J. "A Corner Finding Algorithm for Image Analysis and Registration." *Proc. of AAAI National Conf.*. 1982

[Faugeras 1984] Faugeras, O. "New Steps Towards a Flexible 3-D Vision System for Robotics." *Proc IEEE 7th Int Conf on Pattern Recognition*, 1984, pp 796-805

[Forgy 1981] Forgy, C. *OPS5 User's Manual*, The Robotics Institute, Canergie-Mellon University, Pittsburg, PA, USA, July 1981

[Forgy 1985] Forgy, C. *The OPS83 User's Manual*, Production Systems Technologies Inc., 1985

[Fox and Kempf 1985] Fox, R., and Kempf, G., "Opportunistic Scheduling for Robotic Assembly." *Proc 3rd Int Symp of Robotics Research*, Oct 1985, pp 111-117

[Freedman et al 1985] Freedman, P., Carayannis, G., Gauthier, G., and Malowany, A., "A Session Layer for a Distributed Robotics Environment." *Proc IEEE COMPINT'85*, Sept 1985, pp 459-465

[Freedman et al 1986] Freedman, P., Michaud, C., and Malowany, A., *Considerations for a Robotics Database*, Tech Rep TR86-6R, Computer Vision and Robotics Laboratory, Dept E.E., McGill University, Montréal, Canada, 1986

[Freeman and Davis 1978] Freeman, H., and Davis, L., "A Corner Finding Algorithm for Chain Coded Curves." *IEEE Trans. on Computer*, vol C-26, 1978, pp. 297-303

[Freund and Hoyer 1984] Freund, E., and Hoyer, H., "Collision Avoidance for Industrial Robots With Arbitrary Motion." *Journal of Robotic System*, vol 4, no 1, 1984, pp 317-329.

[Fu and Mui 1981] Fu, K., and Mui, J., "A Survey of Image Segmentation," *Pattern Recognition*, vol. 13, no 1, Jan 1981, pp 3-16.

[Garetti et al 1982] Garetti, P., and al., et. "MODOSK A Modular Distributed Operating System Kernel for Real-Time Process Control." *Microprocessing and Microprogramming*, vol. 9, 1982, pp. 201-213.

[Gauthier et al 1985] Gauthier, D., Carayannis, G., Freedman, P., and Malowany, A., *A Session Layer Design for the CVaRL Local Area Network*, Tech Rep TR-85-7R, Computer Vision And Robotics Laboratory, McGill University, Montréal, Québec, Canada, 1985

[Geisler 1983] Geisler, W., "A Versatile Database for a Robot Vision System," *Proc of SPIE, RoViSec3*, Nov 1983, pp 632-636

[Genesereth 1985] Genesereth, R., Michael, R., and Ginsberg, M., "Logic Programming," *CACM*, vol 28, no 9, Sept 1985, pp 933-941

[Getta 1984] Getta, J., and Rybinski, H., "HOLMES A Deduction Augmented Database System," *Information Systems*, vol 9, no 2, 1984, pp 167-179

[Gevarter 1985] Gevarter, W., *Intelligent Machines*, Prentice-Hall, 1985

[Gini 1983] Gini, M., "Recovering from Failures a New Challenge for Industrial Robots," *Proc IEEE Fall COMPCON'83*, 1983, pp 220-227

[Grimson 1985] Grimson, W., "Computational Experiments with a Feature Based Stereo Algorithm," *IEEE Trans. on PAMI*, vol PAMI-7, no. 1, Jan 1985, pp 17-34

[Gu and Huang 1984] Gu, W., and Huang, T., "Connected Line Drawing Extraction from a Perspective View of a Polyhedron," *Proc IEEE First Conf on AI and Applications*, 1984, pp 192-198

[Guttag 1975] Guttag, J., *The Specification and Application to Programming of Abstract Data Types*, Ph D. Dissertation, Tech Rep CSRB-59, Dept of Computer Science, University of Toronto, Toronto, Canada, 1975

[Hall 1982] Hall, E., Tio, J., McPherson, C., and Sadjadi, F., "Curved Surface Measurement and Recognitioin for Robot Vision," *Conf Record, Workshop on Industrial Applications of Machine Vision*, May 1982, pp. 187-199.

[Harmon and al 1984] Harmon, S and al, et. "Coordination of Intelligent Subsystems in Complex Robots." *Proc IEEE 1st Conf on AI Applications*. 1984

[Hartzband and Maryanski 1985] Hartzband, D. and Maryanski, J.. "Enhancing Knowledge Representation in Engineering Databases." *IEEE Computer*. vol 18, no 9, Sept 1985, pp 39-48

[Haskin and Lorie 1982] Haskin, R. and Lorie, R. "On Extending the Functions of a Relational Database System." *Proc ACM Int Conf Management of Data*. June 1982

[Hong and Levine 1986] Hong, W. and Levine, M. *A Knowledge-Based Approach to Computer Vision Systems*. Computer Vision and Robotics Laboratory, Dept E E, McGill University, Montréal, Canada, 1986.

[Horn 1975] Horn, B.. "Obtaining Shape from shading Information." *The Psychology of Computer Vision*, edited by P.H Winston, McGraw-Hill, New York, 1975

[Hueckel 1971] Hueckel, M.. "An Operator which Locate Edges in Digitized Pictures." *Journal of the ACM*, vol. 18, no. 1, 1971, pp. 113-125.

[Hung 1985] Hung, S.. "On the Straightness of Digital Arcs." *IEEE Trans. on PAMI*, vol. PAMI-7, no. 2, 1985, pp 203-215

[Jensen 1986] Jensen, E.. "The Honeywell Experimental Distributed Processor–An Overview." *IEEE Computer*, vol 11, no. 1, pp. 28-33

[Kak et al. 1986] Kak, A, Boyer, K, Chen, C., Safranek, R., and Yang, H.. "A Knowledge-Based Robotic Asssembly Cell." *IEEE Expert*, vol 1, no. 1, Spring 1986, pp. 63-83.

[Kaminski 1986] Kaminski, M.. "Computers Protocols for Communicating in the Factory." *IEEE Spectrum*, vol. 23, no 4, April 1986, pp 56-62

[Kellogg 1984] Kellogg, C.. "The Transition from Data Management to Knowledge Management." *IEEE Int. Conf. on Data Eng.*, 1984, pp. 467-472.

[Kellogg 1986] Kellogg, C., "From Data Management to Knowledge Management," *IEEE Computer*, Jan 1986, pp. 75-84

[Kernighan and Ritchie 1978] Kernighan, B. and Ritchie, D. *The C Programming Language*, Prentice-Hall, Sotfware Series, 1978

[Kim and Banerjee 1985] Kim, W. and Banerjee, J. "Support of Abstract Data Types in a CAD Database System," *Proc IEEE COMPINT'85*, Sept 1985, pp 381-385

[Kitchen 1982] Kitchen, L. "Grey Level Corner Detection," *IEEE Pattern Recognition Letters*, vol 1, 1982, pp. 95-107

[Kopetz and kuroda 1980] Kopetza, H. and Kuroda, S. "DCCS for Integrated System Control," *Proc 2nd IFAC Workshop on Distributed Control System*, Sept 1980

[Kossman 1986] Kossman, D., *A Multi-Microprocessor-Based Environment for Industrial Robots*, M.Eng Thesis, Computer Vision and Robotics Laboratory, Dept E E, McGill University, Montréal, Canada, 1986

[Lafue and Mitchell 1983] Lafue, G., and Mitchell, T. "Database Management Systems and Expert Systems for CAD," *CAD Systems Framework*, edited by K Lillehagen, North Holland, 1983, pp. 313-323

[Lee and Gossard 1985] Lee, K., and Gossard, D. "A Hierarchical Data Structure for Representing Assemblies Part 1," *Computer Aided Design*, vol 17, no 1, Jan 1985, pp. 15-25.

[Leopol 1984] Leopol, G., "Factory Nets Follows MAP," *Electronics Week*, Dec 1984, pp 20-21.

[Levine 1969] Levine, M., "Feature Extraction: A survey," *Proc of the IEEE*, vol. 57, no 8, Aug. 1969, pp. 1391-1407.

[Levine and Shaheen 1981] Levine, M., and Shaheen, S. "A Modular Computer Vision System for Picture Segmentation and Interperetation," *IEEE Trans on PAMI*, vol. PAMI-3, no. 5, 1981, pp. 540-556.

[Levine 1985] Levine, M . *Vision in Man and Machine*. McGraw-Hill, 1985.

[Lloyd 1985] Lloyd, J.. *Implementation of a Robot Control Development Environment*. M.Eng Thesis, Computer Vision and Robotics Laboratory, Dept E.E . McGill University, Montréal, Canada, 1985.

[Lozano-Perez and Wesley 1979] Lozano-Perez, T . and Wesley, M . "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *CACM*, vol 22, no 10, Oct 1979, pp 560-570

[Lozano-Perez 1983] Lozano-Perez, T . "Robot Programming," *Proc of the IEEE*, vol 71, no 7, July 1983

[Luh and Klaasen 1985] Luh, J . and Klaasen, J . "A Three-Dimensional Vision by Off-Shelf System with Multi-Cameras," *IEEE Trans on PAMI*, vol. Pami-7, no. 1, Jan 1985, pp 35-45.

[Maletz 1983] Maletz, M . "An Introduction to Multi-Robot Control Using Production System," *IEEE Computer Society:Workshop on Languages for Automations*, Nov. 1983, pp 22-27

[Mansouri et al. 1985a] Mansouri, A . and Malowany, A.. "Using Vision Feedback in Printed-Circuit Board Assembly," *Proc. of the 1985 IEEE Microprocessor Forum*, April 1985, pp. 115-122

[Mansouri et al 1985b] Mansouri, A . Malowany, A.. and Levine, M.. *Line Detection in Digital Pictures: a Hypothesis Prediction/Verification Paradigm*, Tech Rep. TR85-17R, Computer Vision and Robotics Laboratory, Dept E.E , McGill University, Montréal, Canada, Aug. 1985.

[Medioni et al 1985b] Medioni, G.. and Huertas, A.. *Edge Detection with Subpixel Precision*, Tech. Rep. Report USCISG 106, University of Southern California, 1985.

[Medioni and Yasumoto 1986] Medioni, G., and Yasumoto, Y., "Corner Detection and Curve Representation Using Cubic B-Splines." *Proc IEEE Conf on Robotics and Automation*, vol 2, 1986, pp 764-767

[Michaud 1985] Michaud, C., *Mroutines c Using the Microbo Robot With Style*, Tech Rep TR-85-3R, Computer Vision And Robotics Laboratory, McGill University, Montréal, Québec, Canada, January 1985

[Michaud et al 1986a] Michaud, C., Mansouri, A., Malowany, A., and Levine, M., *Some Issues in Robotics*, John Wiley, 1986

[Michaud et al 1986b] Michaud, C., Malowany, A., and Levine, M., "Multi-Robot Assembly of IC's." *Computer Generated Images the State of the Art*, Springler-Verlag, Tokyo, Japan, 1986, pp 106-117

[Milne 1983] Milne, B., "Board Testing: The Future is Software Networking." *Electronic Design*, Nov 24, 1983, pp 87-96

[Mitchell and Barkmeyer 1984] Mitchell, M., and Barkmeyer, E., "Data Distribution in the NBS Automated Manufacturing Research Facility." *IPAD II. Advances in Distributed Database Management for CAD/CAM*, NASA Conf Publ no 2301, 1984, pp 211-227

[Mortenson 1985] Mortenson, M., *Geometric Modeling*, John Wiley, 1985

[Nakashima 1984] Nakashima, H., "Knowledge Representation in Prolog." *Proc. IEEE Int 'symp on Logic Programming*, 1984, pp 126-130.

[Nevatia 1977] Nevatia, R., "A Color Edge Detector and Its Use in Scene Segmentation." *IEEE Trans. on SMC*, vol SMC-7, no 11, 1977, pp 820-826

[Nevatia 1982] Nevatia, R., *Machine Perception*, Prentice-hall, New Jersey, 1982.

[Newman and Sproull 1973] Newman, W., and Sproull, R., *Principle of Interactive Computer Graphics*, McGraw-Hill, 1973

[Nilsson 1980] Nilsson, N., *Principles of Artificial Intelligence*, Tioga Publisher Company, Palo Alto, California, 1980

[Nitzan 1985] Nitzan, D., "Development of Intelligent Robots Achievements and Issues," *IEEE Journal of Robotics and Automation*, vol. RA-1, no 1, March 1985, pp 3-17

[Paler et al] Paler, K , Foglein, J , Illingworth, J , and Kittler, J , "Local Ordering as an Aid to Corner Detection," *IEEE Journal of Robotics and Automation*, 1985, pp. 351-360.

[Paul 1981] Paul, R , *Robot Manipulator Mathematics, Programming and Control*, MIT Press, Cambridge Mass , 1981

[Pereira 1984] Pereira, F , *C - Prolog User's Manual*, Dept. of Architecture, University of Edinburg, Feb. 1984

[Pressman and Williams 1977] Pressman, R., and Williams, J., *Numerical Control and Computer-Aided Manufacturing*, John Wiley, 1977.

[Rioux 1984] Rioux, M , "Laser Range Finder Based on Synchronized Scanners," *Applied Optics*, vol 23, no 21, 1985

[Roberts 1965] Roberts, L , "Machine Perception of Three-Dimensional Solids," *Optical and Electro-Optical Imformation Processing*, edited by J.T. Tippett et al , MIT Press, Cambridge Mass., 1965, pp. 159-197.

[Rosenfeld and Kak 1982] Rosenfeld, A., and Kak, A., *Digital Picture Processing*, Academic Press, New York, 1982.

[Samet 1980] Samet, H., "Region Representation: quadtrees from boundary codes," *CACM*, vol. 23, no. 3, March 1980, pp. 163-170.

[Sanderson 1983] Sanderson, A., "Parts Entropy Methods for Robotic Assembly System Design," *Annual Research Review*, 1983, pp. 33-39.

[Sedillot 1984] Sedillot, S., "Some Computing Issues in Multiple Robot Systems," *Advance Software in Robotics*, Elsevier Science Publishers B.V., North Holand, 1984, pp. 245-258

[Shaw and Whinston 1985] Shaw, M. and Whinston, B., "Automatic Planning and Flexible Scheduling: A Knowledge-Based Approach," *Proc IEEE Robotics and Automation*, March 1985, pp. 890-894

[Shimano and al 1984] Shimano, B., Geschke, C., and Spalding, C H., "VAL-II:A New Robot Control System for Automatic Manufacturing," *Proc. IEEE Int Conf. on Robotics*, 1984, pp. 278-292

[Sowa 1983] Sowa, J., *Conceptual Structures Information Processing in Mind and Machine*, Addison-Wesley, 1983.

[TCP 1982] "Transmission Control Procedure/Internet Protocol," *Internet Workbook*, Network Information Center, SRI International, March 1982.

[Tou and Gonzales 1974] Tou, J., and Gonzales, R., *Pattern Recognition Principles*, Addision-Wesley, Reading, MA, 1974.

[Tyridal 1980] Tyridal, P., "New Ideas in Multi-Task Real-Time Control System for Industrial Robots," *Proc. of the 10th Int. Symposium on Industrial Robot, 5th Int. Conf. on Industrial Robot Technology*, 1980, pp 659-670.

[Udagawa and Mizoguchi 1984] Udagawa, Y., and Mizoguchi, T., "an Advanced Database System ADAM - Towards Integrated Management of Enginering Data," *IEEE Int. Conf on Data Engineering*, 1984, pp .3-11.

[Voltz1984] Voltz, R., Mudge, T., and Gal, D., "Using ADA as a Programming Language for Robot Based Manufacturing Cells," *IEEE Trans. on SMC*, Nov./Dec. 1984, pp. 278-292.

[Weszka 1978] Weszka, J., "A Survey of Threshold Selection Techniques," *CGIP*, vol. 7, 1978, pp. 259-265.

[Weszka and Rosenfeld 1978] Weszka, J., and Rosenfeld, A., "Threshold Evaluation Techniques," *IEEE Trans. on SMC*, vol. SMC-8, no. 8, 1978, pp. 622-629.

[Wiederhold 1984] Wiederhold, G., "Knowledge and Database Management," *IEEE Soft.*, vol. 1, no. 1, Jan. 1984, pp 63-73.

[Williams 1979] Williams, T., "Two Decades off Change–A Review of the 20-year History of Computer Control," *Control Engineering*, vol. 24, no. 9, 1979, pp. 71-76.

[Winston 1977] Winston, P., *Principle of Artificial Intelligence*, Addison-Wesley, 1977.

[Workshop 1980] "Workshop on Data Abstraction, Databases, and Conceptual Modeling," *Proc. ACM*, 1980.

[Wu 1982] Wu, L., "On the Chain Code of a Line," *IEEE Trans. on PAMI*, vol. PAMI-4, no. 3, 1982, pp. 347-353.

[Zahn and Roskies 1972] Zahn, C., and Roskies, R., "Fourier Descriptions for Plane Closed Curves," *IEEE Trans. on Computers*, vol. 21, no. 3, March 1972, pp. 269-281.

[Zimmermann 1980] Zimmermann, H., "OSI Reference Model – The ISO Model for Open Systems Interconnections," *IEEE Trans. Comm.*, April 1980, pp. 425-432.

[Zucker 1976] Zucker, S., "Region Growing: Childhood and Adolescence," *CGIP*, vol. 5, 1976, pp. 382-399.

[Zucker et al. 1977] Zucker, S., Hummel, R., and Rosenfeld, A., "An Application of Relaxation Labeling to Line and Curve Enhancement," *IEEE Trans. on Computer*, vol. 26, no. 4, April 1977, pp. 394-40.