Plethora:
A Framework for the Intelligent Control of Robotic Assembly Systems.

By

I. P. W. Sillitoe

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology

February 1992

Contents

Synopsis 1
Publications 2

Structure of Thesis 3

Chapter 1 Introduction 5
Background 5
Manufacturing Equipment Flexibility 6
Summary 7
Objective 7
Current Programming Languages 8
Task Orientated Languages 9
Knowledge as a Solution 9
Traditional Expert System Architectures 11
Summary 12
Conclusion 12
References 15

Chapter 2 Control 17
Previous Work 17
Commercial 17
Thorn EMI 17
Flymo 18
Yet Another Manufacturing System 19
Laboratory 20
Hierarchical Control 21
NNS 22
Subsumption Architecture 24
Current 25
Purdue 26
Problem Analysis and Operator Hierarchy 27
Opportunistic Scheduling 28
Discussion 28
Determination of Assembly Steps 28
Plan Execution 29
Error Detection 30
Error Correction 30
Summary 30
References 33

Chapter 3 Knowledge Based Control of Resources 35
Planning and Control 35
Why Incremental Opportunism? 36
The Mechanism of Incremental Opportunism 37
BB1 Architecture 38
Knowledge Specialists 38
Blackboard Structure 40

Conflict Resolution 41
Strategies 41
Foci 42
Conflict Set Generation 42
Attention Focusing 43
Summary 44
Apparent Complexity 45
Plan Representations 45
Implementation 46
Summary 47
Plethora's Blackboard 47
Execution Plane 48
Event Level 49
Reason Level 49
Error Level 49
Solution Level 50
Chosen External Action 50
Confidence 51
Determining Confidence Values 52
Confidence Function 54
Pre-Condition Action 57
Groups of KS's 58
Implementation Level 59
Modes 59
Temporal Ordering and Dependence 59
Focusing Attention 59
Basic KS's 62
BlackBoard Interface 62
Previous BlackBoard Architectures 63
Summary 64
References 65

Chapter 4 Communications **67**
Message Passing 67
Why Use the message passing paradigm? 68
Issues in Message Passing Systems 68
Blocking or Non-Blocking 69
Addressing Mechanism 70
Message Format 70
Communication Failure 71
Requirements 72
Previous Message Passing Systems 72
General Distributed Programming 73
Message Passing Applied to Industrial Robotics 73
IMRS 73
MAP 74
Summary 75
Plethora's Message Passing System 76
Players, Prompters, Companies and Groups 76
Message Passing Primitives 79

Header Format 80
Message Address 80
Message Types 81
Requests and Reply 81
State Changes 81
Trigger, Precondition 82
Acknowledgement 82
Priority 82
Departure Expiry Times 83
Acknowledgements 83
Transactions 83
Primitives and Protocols 84
Sending Messages 84
Receiving Messages 84
Streams and Pipelines 86
Player Scheduler 87
Message Passing and Prompters 87
Network Interface 87
Network 88
Summary 90
References 91

Chapter 5 Objects and Rules 92
Objects 92
Definition of an Object 92
ROOP 94
Vision Server 94
Representation 94
Garbage Collection 96
Language 99
Rules 103
Rule Interpreters 104
HC State Machine 104
Procedural Interpreter 106
Implementation 106
Summary 107
References 108

Chapter 6 World Modelling 109
Previous Work 109
SMGR 109
NBS 110
Purdue 110
Which Knowledge? 111
Relational and Geometric Information 112
Geometric Modelling 113
Solid Modelling 113
Swept Volumes 113
Boundary representation 113
Cellular/Spatial Occupancy Methods 113

Constructive Solid Geometry 113
GeoMod 114
Body Frames 114
Volume Frames 116
Volume Hierarchy 116
Interference Test 120
Chain Frames 122
Collision Detection 125
Solid in Motion 125
Swept Volume Representations 125
Incremental Motion 125
Algebraic Representation 126
Summary 126
Collision Detection in GeoMod 126
Relational Information 128
Descriptions of Spatial Constraints 129
Primitive Coordinate Frame Constraints 131
Assembly Descriptions 133
Previous Work 133
Assembly Descriptions in Plethora 135
Solve-Construction 136
Summary 139
References 141

Chapter 7 Object Recognition 143
Overview 143
Introduction 144
Previous Work 145
Methodology 145
Generation of the Volumetric Model 145
Outline of the algorithm 146
The Parallel Projection Approximation 151
Generating the Centralised Moment Matrix 152
Moment Invariants 153
Principle Axes 153
Results 154
Verification 154
Classification 155
Discussion 155
Illumination 156
Calibration 156
Conclusions 156
References 158

Chapter 8 Path Planning 159
Introduction 159
Method 161
Free Space Representation 161
Determination of the Initial Path 163
Modification of the Z coordinates 165

Experimental Results 168
Conclusions 169
References 170

Chapter 9 Experimentation 171
Introduction 171
The Problem 172
Description of the Work Cell 172
The Assembly 173
Planning the Task 176
Summary 180
Description of the Implementation goal 182
3D-Moment 182
RPplanner 182
2D-Moments 182
Approach and Unapproach 182
Mgrasp and Mungrasp 183
Execution 184
Summary 186
Execution with Events 187
Execution with an Error 189
Summary 190
Discussion 191
References 192

Chapter 10 Implementation 193
System Implementation 193
Dynamic Storage allocation 194
Rule Tables and Message Interpretation 195
Implementation of Objects 196
Server Architecture 197
Network 198
Compilation and Hardware 199
Summary 200
References 201

Chapter 11 Conclusions 202
Requirements 202
Control 202
Knowledge 203
Communications 204
Further Work and New Avenues 205
Partition and Mapping 205
Model based Recognition 205
Path Planner 206
Parallel Collision Detection 206
Parallel External Actions 206
Confidence 207
Planning and Acting 207
References 208

Appendix A 209
Appendix B 212
Appendix C 219
Appendix D 222
Appendix E 232
Appendix F 239
Glossary 243

Synopsis

The thesis describes a distributed software environment designed for the development, evaluation and comparison of new techniques in knowledge based control of robot assembly work cells. It has characteristics which fulfil deficiencies within previous systems and contains within it new techniques in task specification, distributed control[1,2], object recognition[3,4] and path planning[5].

The control of the resources within the cell is based upon an extension of the facilities of a classical blackboard architecture to include plan execution. Unlike previous schemes, these additions allow Plethora to reason about the intent of an action, the current state of the cell and asynchronous events within a single framework. It is this seamless operation and extended representational adequacy that allows Plethora to explore new techniques dealing with the uncertainty inherent in a flexible work cell.

The task is specified in domain terms and interpreted to produce a partially ordered set of goals. This new technique is based upon a two-stage ordering process using constructional constraints and necessary collision avoidance.

Two new methods, one for object identification and the other for path planning, have also been developed using the system. These have two advantages, efficiency and the ability to operate on data from a vision system or Plethora's geometric modeller. Both methods can be completed within the critical times typical of an assembly work cell.

Finally, results of an experiment using the system on a laboratory work cell illustrate how it encompasses previous techniques and can be used to develop new techniques not possible with earlier architectures.

Publications

[1].Sillitoe,I.P.W.
"Towards Knowledge-Based Control of a Flexible Assembly Work cell",
IEE Colloquium on Knowledge Based Environments for Industrial Applications, Savoy Place London,June 1989,p1-4. (see Appendix A)

[2].Sillitoe,I.P.W
"An Approach to the Programming of Distributed Shared Resources within an Experimental Robotic Work cell", EuroFORML'90,Southampton,Oct. 1990. (see Appendix B)

[3].Sillitoe,I.P.W Edwards,J.
"A Multi-view Robotic Vision System for Efficient Object Recognition", Proc. Int.Conf. Systems Science,Wroclaw,Poland 1989. (see Appendix D)

[4].Sillitoe,I.P.W. Edwards,J.
"The Design of a Real Time Three Dimensional Vision System for Object Identification", Proceedings of the 12th Occam User Group, April,1990,p198-205 (see Appendix E)

[5].Sillitoe,I.P.W
"A Gross Motion Planner for Robot Path Generation", 6th International Conference on Systems Engineering, Coventry, 1988,p499-504. (see Appendix F)

Structure of the Thesis

Figure 1.0 shows the title of each chapter, followed by its chapter number and its reliance upon information found in previous chapters.

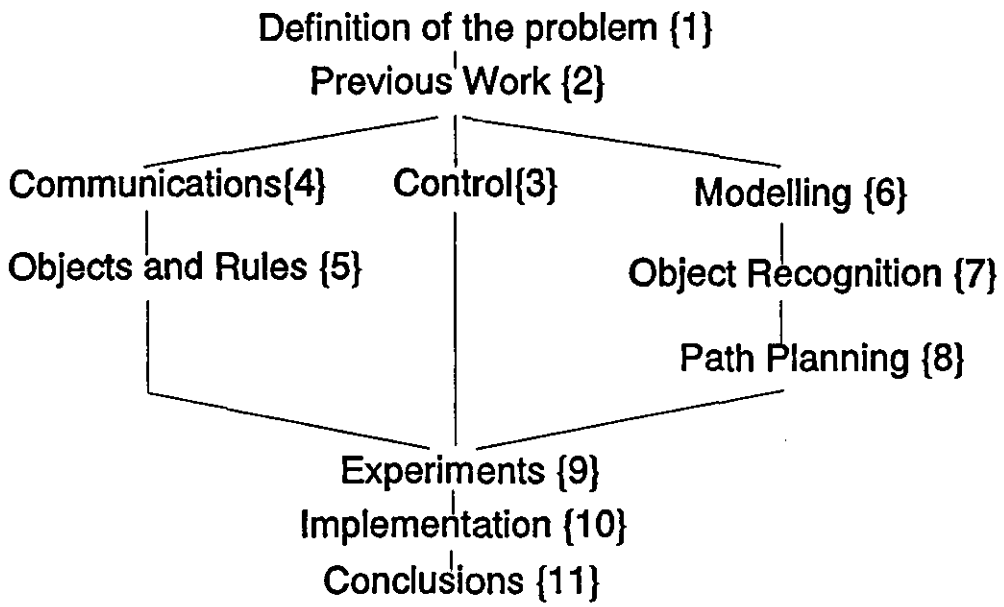


Figure 1.0 Chapter Dependency

The first chapter initially describes the work within the wider context of flexible manufacturing, it then proceeds to define a problem, which is a precursor to true flexible manufacture, known as the Fitter Problem. The necessary requirements of a solution to one aspect of the Fitter problem are then outlined. The following chapters expand upon these requirements under the headings of communications, control and modelling. Each chapter reviews previous techniques in the light of the requirements specified in chapter 1 and then describes the approach taken in Plethora. Chapters 7 and 8 illustrate the use of Plethora as a tool for the development of new techniques.

Chapter 9 draws the individual aspects of Plethora together using a number of small scale experiments. It illustrates the operation of Plethora as an entity and allows a comparison to be made with other techniques. The penultimate chapter

discusses issues of implementation and Plethora's mapping on to different hardware architectures. Chapter 11 addresses possible additions to Plethora and how it might be used to tackle other facets of the Fitter Problem.

Chapter 1
Introduction

Background

In an attempt to adapt to competition within the market place, many industries have recently moved away from large batch single product manufacture. Instead, such industries have tended towards the manufacture of smaller batches of related products and a corresponding frequent change in product line. The reasoning behind this change in approach is two fold. Firstly, those manufactures which adapt to demand most quickly will be the first into a new market and secondly, by tailoring its product to a particular demand it is possible for the manufacturer to provide itself with a niche in a particular market. The means by which this change in approach to manufacture is achieved is the theme moving through flexible manufacture.

The cost of storage and investment in unsold product makes maintaining large stocks of each product line uneconomic as an answer to the problem. However, with the application of the "Just-in-Time"[1] (or Kanban) philosophy, successful attempts have been made to solve some of the economic and managerial problems associated with fluctuating demand, while still maintaining the necessary response time and variation in product line. In such a system, the stock and flow of work within the factory are controlled by the instantaneous demand for the product rather than the predicted demand, so reducing the need for the maintenance of large stocks.

A similar argument can equally be applied to specialised manufacturing equipment which has application to a single product, since such equipment when it is not in use, also represents an additional cost of manufacture. Thus, in order that the investment made in manufacturing equipment for a Just-In-Time system is most effective, the equipment must be applicable or able to adapt to more than one function[1] dictated by the demand for a particular product. A work cell which can perform a number of related tasks is known loosely as a flexible work cell. In practice, the creation of such a cell is very difficult to achieve and present day flexible cells fall short of many of the attributes required of them by the rest of the

manufacturing system. Hence, the term flexible, when used within the literature, often describes specific features of flexibility applied to particular aspects of the cell, rather than the behaviour of the cell as a whole. Since this work is concerned with the provision of flexible behaviour for the cell, the next section discusses the form of flexibility which is required of the cell as an entity, in order that it might fit within the wider framework of a flexible manufacturing system.

Manufacturing Equipment Flexibility

The flexibility of a work cell is ultimately dependent upon the answer to two interrelated questions. Firstly, whether the equipment which constitutes the hardware of the cell is capable of flexible operation (i.e can its jigs, grippers etc manipulate a range of workpieces ?). Secondly, can the cell be made to change its behaviour rapidly enough to adapt to the changes in external demand and the internal state of the cell. The time taken to make these changes constitute the latencies of the cell. Hence, the latencies of such a system can be regarded as being those formed from the set-up times and those associated with the changes that take place in the cell at run time. Set up time is the time taken to modify a machine prior to it beginning a new operation. In conventional systems this would involve realignment of jigs and the changing of tools. However, in a flexible system it will also include re-planning of the operations, re-programming and perhaps a change in the sensors - a more complex and time consuming task.

At run time the flexible system will also have new requirements over and above those of the conventional system. Firstly, it must adapt to commands from the overall factory scheduler. Secondly, it must adapt to local events outside the cell, such as the variation in the arrival of parts, and thirdly, provide error/fault detection and correction. This latter facility becomes a requirement because with the increased flexibility of the manufacturing equipment, there will also be a greater uncertainty associated with the state of the cell and its operation and hence a greater possibility of error.

Summary

Ideally, a flexible assembly work cell should be able to assemble a large number of different structures with differing work pieces and be able to change its function with minimum latency. However, at present practical "flexible cells" (e.g [2]) can only accommodate a few variations in structure and/or have large latencies[1]. This is in part due to physical design of the work cell. Present work cells are designed to minimise any uncertainty in the work piece's size, shape, type, orientation and arrival, this in turn makes control of the cell a much simpler process and reduces the cost of the software development to $\sim 10\%$ of the total cost of the work cell[3]. However, this understandable striving for certainty, limits the flexibility of the cell.

Thus as the cell hardware becomes less specific in order to allow flexible behaviour, the complexity of the controller will radically increase, as one of its major functions will include the reduction of the potential latencies within the system.

Objective

The objective of Plethora is to provide a framework in which the investigation of methods to reduce the latencies can take place. More precisely the objective can be reformulated in terms of the provision of the necessary requirements for the solution of the Fitter problem. Where the Fitter problem is,

> ***Given descriptions of,***
>
> > ***The assembly to be carried out, in terms of the final relationships between objects.***
> >
> > ***The effectors and sensors present in the work cell.***
> >
> > ***The initial state of the work cell.***
>
> ***dynamically schedule, control and monitor the assembly.***

The solution of the Fitter problem is one of the precursors to a truly flexible assembly cell.

The following section illustrates the deficiencies in present day commercial robotic programming languages, task orientated languages and traditional expert system architectures as a means of providing a basis for a solution to the Fitter problem.

Current Robot Programming Languages

Although current robot programming languages [e.g. 4-11], can be used to deal with the procedural aspects of the cell (i.e. to sequence and synchronise events within the cell), they are for the most part based upon adapted general purpose procedural languages (such as Basic and its derivative VAL [2]), and are as such inappropriate tools for the solution of the Fitter problem, as [13]

> They separate the control level from the user interface, so that new algorithms cannot be easily tested.
>
> They make communication with external devices difficult.
>
> They emphasise the procedural component of programming, making changes to the original system difficult and so require all the error conditions to be explicitly anticipated.
>
> They are not portable between different work cells.
>
> Many of them are not multi-tasking, which complicates the synchronisation of actions within the cell. This often leads to a sequential execution of a task which has a more natural parallel solution.
>
> There is no general way of handling sensor information.
>
> They do not allow the user to specify the actions in domain terms.

The fact that these apparent deficiencies are real, is borne out by the lack of use of the controller language in most conventional industrial applications. In the majority of industrial applications, actions are taught by guiding the end-effector through a sequence of poses and storing them, the controller is then used to replay the motion by continuously moving the robot between the sequence of stored poses. The teaching process is usually completed without any recourse to the controllers language [13].

Task Orientated Languages

In the light of the previous list of deficiencies, it can be summarised that most conventional robot language primitives do not correspond to the primitive actions of the work cell and so make programming simple tasks overly complicated. For example, picking up a workpiece from a pallet requires the calculation of the relative transform between the gripper and pallet, which in turn requires the dimensions and orientation to be known and programmed explicitly. A partial solution to this problem is to specify the task in terms of the desired relationships between the parts, rather than the robots action, so allowing the system to specify and calculate the necessary transformations (i.e. a task level language).

Several attempts at task level languages have been made AL [14,15,16], AUTOPASS [17], LAMA [18], RAPT [19] and LM-GEO [20]. In the main, the only additional source of knowledge available to the controller was a 3D geometric modeller, which modelled the position and shape of the objects within the cell. This was used to save the user the inconvenience of calculating the relative frame calculations required for point to point motion and for checking collision detection off-line. LM-GEO used high level primitives to specify the final assembly and a set of rewrite rules to produce the manipulator level program. LAMA, making most use of the predictive aspect of its knowledge, exhaustively generated possible error conditions and sensory states based upon the results of the 3D simulation and then asked the user for corrective action.

However, such languages still share the major deficiencies of commercial programming languages when coping with sensory data, extensibility and error detection.

Knowledge as a Solution

The deficiencies highlighted by the previous section can be ameliorated by,

Increasing the amount of sensory information reported to the controller.

Introducing knowledge of the cell's capabilities and the assembly process to be undertaken into the controller. This would allow the controller to make implicit use of sensor information within the context of the task and specify the task in terms of the required goals (i.e. domain terms).

Thus sensory information can be used to reduce the uncertainty of the physical properties of the work cell. While knowledge of the assembly process and its objectives allows planning, scheduling of action, error detection and correction in the light of the state of the cell.

The production of reliable sensory information is notoriously difficult (see [21] for an overview) and so while it will be necessary to improve the sensors there will also be a need to make better use of the sensory information which is available. This can once again be achieved if the sensory information is used in conjunction with knowledge of the task. Such knowledge can be used in a number of ways,

To predict the likely form of the sensor data and use previous data to refine the measurements made.

Predict the area of the sensor data required for the accomplishment of a task and so reduce the processing time. For example, picking the appropriate window of an image which is predicted to hold the item required, rather than processing the whole image plane.

Predict the form and value of the sensory data and so automatically generate verification conditions for the completion of the task.

Choose a processing strategy aimed at identifying a particular feature for identification, rather than globally processing and matching all the sensory data available.

Integrate diverse sensor data (i.e. using tactile, force measurement and vision in order to validate a grasping operation) so as to compensate for the deficiency in the individual sources.

Thus model based knowledge about the cell, sensors and task can be used to help

tackle the problems associated with the additional flexibility, unreliable or incomplete sensory data and incomplete modelling which are to be found in a truly flexible cell. Hence, prerequisites to a solution of the Fitter problem must include the answers to the questions: What knowledge is to be introduced into the controller and how is the controller to manage this knowledge?.

Traditional Expert System Architectures

It has already been argued that if there is to be a solution to the Fitter Problem, then more use of the available knowledge must be made (i.e the determination of verification strategies based upon the sensor information currently available) and more knowledge sources will be required (i.e. a data base which could be used to determine the likely weight of an object and so help in the choice of verification strategy). So the problem now becomes how to represent, manage and utilize this increase in knowledge.

Many of the elements of the Fitter problem (i.e. piano movers path planning problem discussed in Chapter 8) have general solutions which are NP-Complete as well as having a number of more efficient partial solutions which only deal with a particular case of the general problem. An efficient system, must therefore manage a large number of techniques which each solve a particular case. The problem of collation and selection of the most applicable technique is a well researched area in expert system design and hence methods used within such systems could be suitable for managing the knowledge within Plethora. However, some of the characteristics of the Fitter problem are difficult to cope with within the structure of a conventional expert system. These include, [22]

> The problem has non-monotonic elements, making retracting steps when planning and determining goals in advance of execution very difficult.
>
> The environment can only be partially predicted or known. This is in part due to the lack of sensory data and partly due to the incomplete modelling of the environment which is inherent in any practical system.

Processing of sensory information can often produce spurious results.

The processing of sensory information or the completion of a physical action within the cell is often greater than the time taken to specify the next action. Hence, errors in selecting an action can dramatically reduce the efficiency of the system.

The geographical separation makes it inherently distributed.

Possible changes in the structure and composition of the cell during its lifetime.

Summary

It has been shown that neither procedural, task langauge or traditional expert system architectures alone provide a basis for a solution to the Fitter problem. Rather each has attributes which makes it applicable to some part of the problem and it is these attributes which need to be included in a final solution. The procedural languages provide for many of the real time needs of the cell, the task level languages provides implicit task specification and techniques borrowed from expert system design, provide generalised techniques for management of the sources of knowledge within the cell. However, these disparate techniques only provide partial answers and still do not address the areas of distribution, planning, error recovery or the inclusion of sensory data.

Conclusion

The necessary requirements for a solution to the Fitter problem cannot be confined within a single existing framework. Many of the requirements have aspects which straddle the boundaries of research areas which are usually regarded as distinct (i.e. real-time control and knowledge based systems). However, if these requirements are to be solved satisfactorily they need to be treated as an entity rather than forcing parts of the problem into preconceived and artificial boundaries. This will require a different architecture to those available at present, one centred upon the problem requirements in context rather than conventional topic boundaries.

The general requirements of Plethora can be categorised under the following headings.

Control
The selection and sequencing of both action and knowledge.

ct1. The ability to automatically generate and schedule, actions and plans.

ct2. To automatically interpret sensory information within the framework of the plan and actions.

ct3. To implicitly detect and cope with error conditions.

ct4. To provide a modular implementation which allows the addition of knowledge sources and devices.

Knowledge
The type of knowledge and its form of representation.

kn1. The provision of physical information about the contents of the cell.

kn2. The provision of relational information to plan and interpret the task.

kn3. The provision of information in a suitable form for its end use.

kn4. The provision of information rapidly enough to satisfy critical times.

Communication
The underlying mechanism by which the control is possible.

cm1. To be amenable to implementation in a distributed environment.

cm2. To have attributes which uniformly support both the real time and knowledge based requirements of the system.

cm3. To adapt to changes in the structure of the cell.

Although these headings seem to form distinct categories, the solution for a particular requirement cannot be seen in isolation and is dependent upon other chosen solutions (i.e. The choice of representation of relational information will effect the possible choice of methods used to generate the plan).

What is Plethora?

Plethora is an experimental real time expert system shell consisting of loosely coupled distributed agents. It is designed to assist in the achievement of the aims above and thus endeavours to integrate knowledge based and real time techniques within a single framework.

In common with other shells it provides many of the necessary resources for the manipulation of the domain, and a method for introducing new user defined techniques. In this case, Plethora provides a flexible control mechanism and geometric modelling, relational modelling and vision preprocessing modules. Figure 1.0 illustrates the modules and their connection through a distributed message passing system. The path planning and object recognition modules are examples of new user defined techniques developed using Plethora.

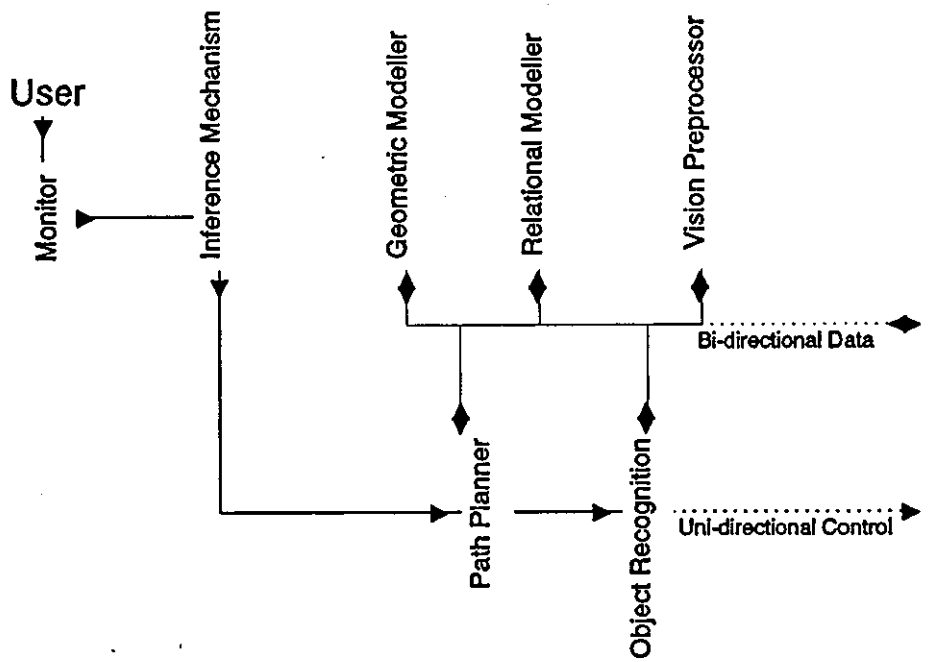


Figure 1.0 Plethoras overall structure.

Plethora is an object based system and so new techniques or devices must only conform to the simple message passing protocols in order that they may be added to system.

Figure 1.0 also illustrates that the systems resources are controlled and synchronised through a single centralised mechanism. However, Chapter 11 discusses a method by which this may also be distributed providing a completely distributed system.

The following chapters expand upon the requirements within the categories above and describe the local solutions adopted and their influence upon the overall design of Plethora.

References

[1] Lubben,R.T
"Just-In-Time Manufacturing: An Aggressive Manufacturing Strategy", McGraw-Hill, ISBN 0-07-038911-X, 89.

[2] Williams,A.M. Walters,M. Reay,D.
"A Flexible Assembly Cell" and
"Commercially available flexible Assembly Cell",
AUTOMAN, May 85, Birmingham.

[3] Miles,B., Lucas Research Centre, Private Communication, Jan.85.

[4] Gruver,W.A Soroka,B.I. Craig,J.J. Turner,T.L.
"Evaluation of Commercially Available Robot Programming Languages", 13th ISIR, p12-58 12-68, 83.

[5] Wood,B.O., Fugelso,M.A.
"MCL : The Manufacturing Control Language ",16th ISIR, p12-84, 86.

[6] Latombe,J.,Mazer,E.
"LM : A High Level Programming Language for Controlling Assembly Robots", 11th ISIR, p683-690, 81.

[7] Voltz,R.A. Mudge,T.N. Gal,D.A.
"Using ADA as a Robot System Programming Language", 16th ISIR, p12-42 12-57, 86.

[8] Haurat,A. Thomas,M.C.
"LMAC: A Language generator System for the Command of Industrial Robots", 16th ISIR, p12-69 12-78, 86.

[9] Park,W.T
"The SRI Robot Programming System (RPS)", 16th ISIR, p12-21 12-41, 86.

[10] Hayward,V.,Paul,P.R.
"Robot Manipulator Control Under Unix", 16th ISIR, p20-32 20-44, 86.

[11] Inoue,H. Ogasawara,T. Shiroshita,O. Natio,O.
"Design and Implementation Of High Level Robot Language", 16th ISIR, p6-75 6-81, 86.

[12] VAL 1 User Manual, Unimation.

[13] Sorka,B.I.
"What Can't Robot Languages Do ?", 15th ISIR, p12-1 12-8, 85.

[14] Goldman,R.
"Recent Work With the AL System", 5th IJCAI, vol. 2, p733-735, 77.

[15] Govindaraj,S. Doty,K,L.
"General Purpose Robot System and Task Development Facility", 16th ISIR, p3-15 3-37, 86.

[16] Mujaba,M.S.
"Current Status of the AI Manipulator Programming System", 10th ISIR, p119-127, 80.

[17] Lieberman,L.I. Wesley,M.A.
"AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly",IBM Journal Of Research and Development, p321-333, July 77.

[18] Lozano-Perez,T. Winston,P.H.
"Lama: A Language For Automatic Mechanical Assembly", 5th IJCAI, p710-715, 77.

[19] Popplestone,R.J Ambler,A.P Bellos,I.M.
"An Interpreter for a Language for Describing Assemblies", Artificial Intelligence, vol. 14, p79-107, 79.

[20] Mazer,E.
"LM-GEO : Geometric Programming of Assembly Robots", Advanced Software in Robotics, Elsevier Science Publishers, 84.

[21] Coiffet,P.
"Interaction with the Environment", Robot Technology vol.2, Kogan Page, ISBN 1-85091-402-8, 87.

[22] Building Expert Systems,
Edited by F.Hayes-Roth, D.A.Waterman and D.B Lenat, Addison-Wesley publications, ISBN 0-201-10686-8, 83.

Chapter 2
Control

This chapter describes how related work tackles control and scheduling problems and then proceeds to discuss these architectures in the light of the Fitter problem requirements.

Previous Work

The discussion draws upon work from both robotic assembly and related domains which exhibit uncertainty (such as mobile robot and factory control) and hence have requirements similar to those of the Fitter problem. However, as no single piece of work directly addresses the Fitter problem (the closest being the work at Purdue [1]), but all illustrate approaches to sub-problems of control within differing architectures. A bald description of each approach is given followed by discussion of the systems with respect to the requirements in Chapter 1.

The work has been categorised into three areas, that which has been successfully applied to commercial work cells and factories, the approaches taken in laboratory work cells and proposed approaches still under development.

Commercial

Thorn EMI

This work is applied to a commercial work cell [2] and broadly takes a hierarchical approach to the control. The control [3] takes as its input a file previously created by the user via a graphical simulation package, teach package (used to teach the manipulator positions) and a task orientated program which enables the user to create the final assembly sequence. The assembly sequence is interpreted by the run time system to schedule the cells operations by communicating with the cell's manipulator and PLC. The sequence is decomposed into *Assembly*, *Component*, *Task* and *Action* levels, where the *Action* operations correspond to the possible

primitive functions within the cell and the *Task* level is described in terms of 3 primitives *Pickup*, *Measure* and *Assemble* (a full definition of what is meant by these levels is not given). Errors are detected and recovered from using standard routines developed for each of the levels, if the routine at one level cannot cope with an error the error is passed up the hierarchy to be dealt with. The primary function of the system is to allow the user to program the cell without the need to understand the operation of its low level control, and so reduce the latencies associated with its re-programming.

Flymo

This work arose from the control of a commercial work cell producing assemblies for Flymo lawn mowers [4]. Analysis of the errors in the cell showed that for approximately 80% of the time the product was assembled correctly relying on an operator to correct its operation for the remaining 20%. The assembly instructions are described in terms of the following stages.

Component Feeding
Where a new component has to be brought within reach of the robot.

Component Transport
Once the new component has been fed into the cell, it has to be moved from the feeding outlet to the point of assembly.

Component Mating
The joining of the components in an assembly.

The actions required to perform these operations are then formed into a library of generic assembly actions. This library is implemented as a set of forward chaining rules and include explicit error detection and correction strategies within the rule. The problem solver uses these rules and the product file to control the assembly by expanding each action in the assembly sequence in terms of the rules. The product file created by the user contains explicit error detection-correction information, the assembly sequence and expected sensor information associated with each action. These expected values can be determined by taking the robot through the particular action and recording the sensor data. During the execution of a task a *History* of the

progress of the plan is made by the rules on the *History Stack*. The *History Stack* is a means of recording events, so that under error conditions not covered by the rules, the operator can provide the correction strategy in the knowledge of the previous actions undertaken by the cell. This procedure of adding error correction routines as the error occurs is the means by which the system is said to "learn from its mistakes".

Yet Another Manufacturing System (YAMS)

Unlike either of the previous approaches YAMS does not attempt to enumerate what it understands to be all possibilities, but relies on negotiation between sources of knowledge at run time to plan, execute and monitor, using current rather than predicted information. Negotiation has been applied to a number of domains. The protocol used by YAMS is a form of Contract Net[6], where the nodes in the net represent the distributed computing resources to be managed (i.e the machinery or processes), and they communicate through messages, which are subject to fixed protocols.

In any transaction there are 3 classes of node, the *Manager*, the *Bidders* and the *Contractor*. The *Manager* is responsible for identifying the task to be done and assigning the work to a particular node. The *Bidders* are the nodes which offer to perform the task and the *Contractor* is the node which eventually succeeds in obtaining the task. The protocol begins with the *Manager* broadcasting a *Task-Announcement*, where upon the *Bidders* returns their *Bids*. The *Manager* then evaluates the Bids and sends an *Acknowledgement* to the potential contractor, which then *Reports* it acceptance.

Hence, rather than predicting the assignment of the task, a bidding process takes place which makes use of the local knowledge available at run time. The system uses pre-planned plan segments to decompose the task, "making the planning process trivial" [6] and then uses negotiation to find nodes on which to process them. This allows the system to adapt to external and unpredicted changes. For example, two bidders volunteer themselves at the same time, however, if one of them is busy, the

Manager can then assign the task to the waiting node in preference to the busy one; adapting the state of the system at run time. Modelling the whole system in order to predict this state of affairs at a particular moment would have been a difficult and uncertain process.

However, when this technique is applied in detail, modifications to the protocol are needed in order to maintain an appropriate response time. This tends to produce a complex and specific *Managers* which in turn makes it more difficult to provide an integrated means of detecting and correcting errors. These problems are due to three shortcomings referred to as [6] temporal, loading and spatial ignorance. In general, it was concluded that the problems associated with the approach were the result of a lack of global knowledge in scheduling decisions.

Laboratory

Hierarchical Control (HC)

HC is an approach to the control of large systems. It partitions the control into hierarchical modules. Modules are only allowed to communicate with the modules directly below or above them in the hierarchy. At each level of the hierarchy the implementation of the strategy corresponds to an equivalent level of abstraction in the problem domain. This should lead to a modular and so tractable solution, with all the attendant advantages for design and maintenance of such a system.

Many systems call themselves hierarchical, however, to date only one has a coherent methodology, defined levels and has been implemented in a number of practical roles. This is the National Standards Bureau of America(NBS) approach [8-14].

The NBS's theory of HC incorporates three parallel interconnected hierarchies to model the 3 sources of information within such a system (see figure 2.0)

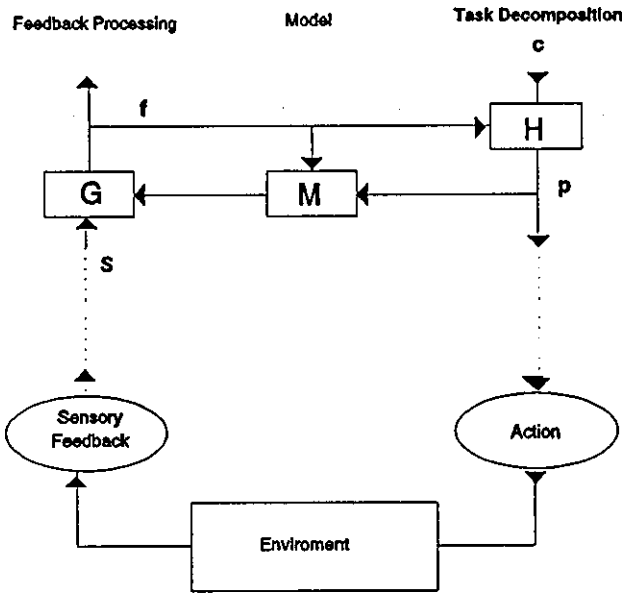


Figure 2.0 NSB Computational Hierarchy.

A behaviour generating hierarchy, which decomposes the task into sub-tasks

A sensory processing hierarchy, which extracts information needed for the goal seeking behaviour.

A world modelling hierarchy, which generates expectations and predictions for the sensory processing modules at each level.

HC is defined as a task sharing or task decomposition technique and can be represented as an AND/OR graph [8].

Each module's function is modelled as a state transition diagram and implemented as a set of forward chaining condition-action rules. However, HC does not allow more than one rule to be triggered for any particular set of input states. This removes the need for any form of conflict resolution. The inputs to these rules can be either the module above it in the hierarchy (i.e. that module's command), from the sensory module at that level (i.e. feedback of the present state), from the module

immediately below it in the hierarchy (i.e. status information of any initiated process) or the present internal state of that particular module. The transfer function (H) of the module, represented as a set of these forward chaining rules is regularly evoked (every 20 msec in [10]), the action must be completed within a multiple of this sampling period. This ensures a predictable and synchronised response. All results are passed between modules via common memory and there is a unique location for each transacted quantity.

HC provides a straight forward methodology and implementation for robot control. However, the major limitation is inherent in the determination of the module transfer function.

> *"H must provide a successful mapping of*
> *S to P, even when there are small perturbations*
> *in F or C from previous modules"[8]*

Small perturbations may be corrected by a low level feedback and may require relatively little sensory information. Larger perturbations might overwhelm a particular module and must be passed to a higher level in the hierarchy to be resolved. This implies that not only explicit programming but explicit error recovery be included in the state transition diagram [14]. For any complex system with incomplete or inconsistent knowledge this will be the case and so leads to very large and unwieldy state transition tables [14].

HC relies heavily on teaching the robot positions, and it's error recovery and detection are provided by the programmer. The programmer monitors the robot's performance of the task and modifies the state transition table accordingly. This is a time consuming process and thus can be considered part of the setting up time.

Task Execution Modelling (NNS)

This project sets out to directly address the problems of on-line decision making, action scheduling, execution monitoring and failure diagnosis and recovery. It is a hierarchical system which for the sake of run time efficiency splits the problem

into off-line and on-line phases. In the off-line state it plans the task, and in doing so, generates what it understands to be all possible evolutions of the on-line task. It represents these as a directed graph of elementary actions. There is a one-to-one correspondence between an elementary action and a NNS state transition. This correspondence allows the system to restart after an error condition from a known and enumerated state. States are described in terms of,

Sites- a place in the cell, intended to support a workpiece.

And

Regions- a specific place in the work cell, which is categorised by the type of effector motion that is available within it.

The workpiece is related to a site using the transforms based upon posture, location and accuracy.

On-line planning occurs either at the end of an elementary action, after a failure or where explicitly stated in the off-line plan network. The planner is based upon a set of plan heuristics which provide approximate solutions to a goal and conflict resolution heuristics which select a path through the enumerated plan network. A node in the network corresponds to a site and an arc corresponds to an elementary action. Each arc has associated with it a set of constraints, which are used to determine whether the action is feasible or relevant at run time. The precise selection of actions is opportunistic, that is it is determined by the perceived state of the world and chosen at run time by the monitor. If the run time system reports an error predefined emergency actions are performed and the state of the system is sent to the failure analysis modules. This module is to establish the likely cause of error and to select corrective action from the plan network.

At present the failure analysis is not performed, so upon detection of an error all sensory information is requested and used to define the actual state within the plan network.

Subsumption Architecture (SA)

This was developed as a means of the control for an autonomous mobile robot. Instead of decomposing the problem of control hierarchically, the subsumption architecture decomposes according to the required behavioural goals (see figure 2.1

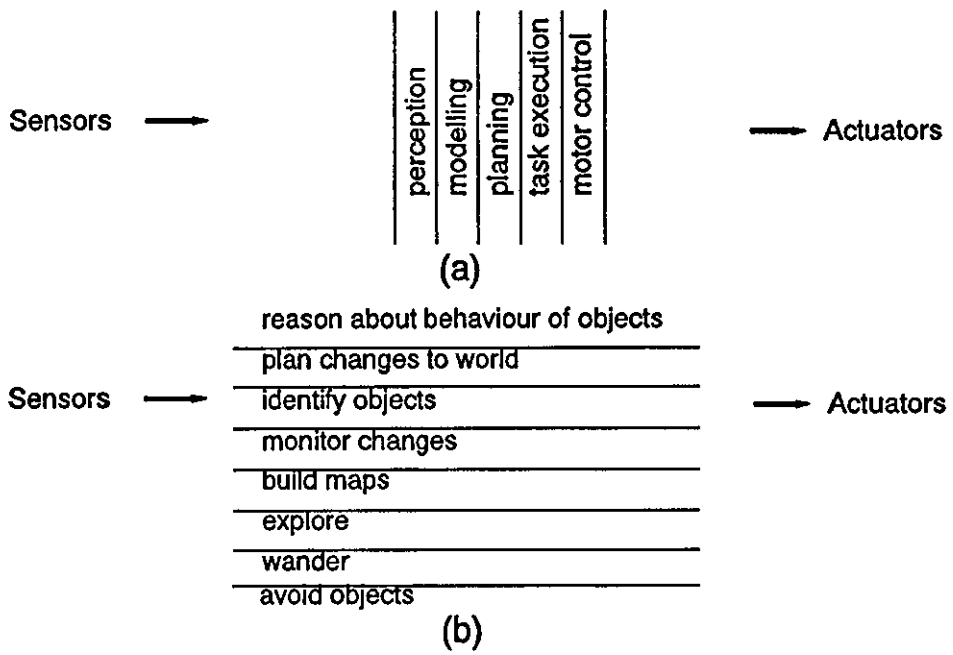


Figure 2.1 (a) Hierarchical Decomposition (b) Decomposition of mobile robot control based upon task achieving behaviours.

for a comparison of the two forms). The lowest level is able to control the robot alone but with restricted behavioural goals, the layer immediately above this modifies the information flow within the first layer to produce more sophisticated behaviour and so on until the required behaviour is produced. Hence, the goals are "hard-wired" in the structure of such a system. A higher layer alters the behaviour of the lower layer by inhibiting or replacing the data to and from modules within the lower layer. The modules are implemented as state machines and the communication paths between modules are fixed. If the higher layers do not produce a course of action within the necessary critical time, this architecture has the advantage of being able to react to an event safely even if that action is not the optimal one.

A modification to the means by which one layer effects another was investigated in [17] using a simulated mobile robot and environment. In this arrangement changes to the data flow were obtained by the upper layer altering a state vector of the lower layer module. The intention of this simulation was to investigate more rigorous means of designing such a system for given behaviours.

Current

Purdue

This is a continuing investigation into sensory systems and sensor-guided motion within a knowledge based robotic assembly cell [1] and its objectives most closely match those of the Fitter problem. The overall architecture of the system is shown in figure 2.2, where the Supervisor coordinates and controls the system. It is the Supervisor which receives the assembly instructions for the product, determines the necessary sequence of operations, initiates and monitors them to complete the task.

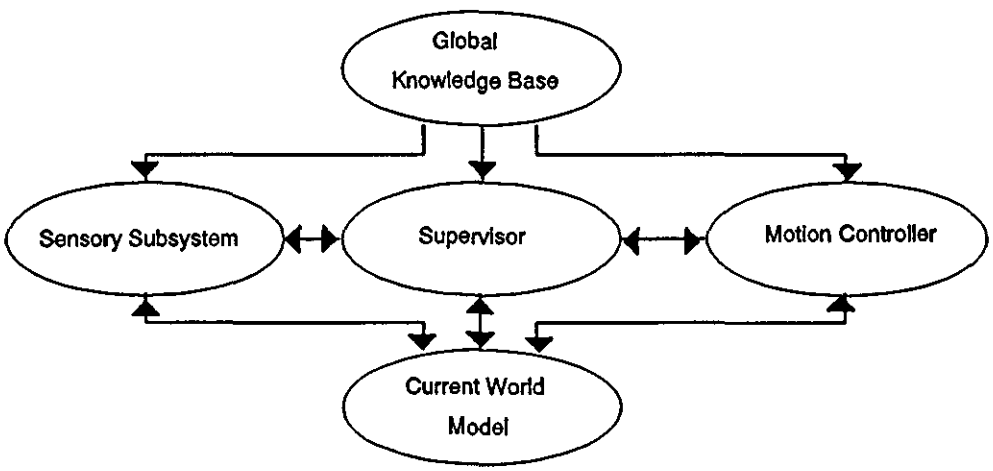


Figure 2.2 Purdue Architecture.

The plan generation uses a slot-filler representation to expand the initial product description by instantiating other frames with the contents of the *Component-Plan* slot to produce a hierarchical set of assembly instructions, the order of the component plans in the slot reflects their execution order. Each frame has a *Verify* field which verifies the correct completion of the plan and an *Error* slot which provides an error handler if it fails. It is through these fields that sensory information is used to guide the plan execution.

During the development of the plan, the planner SPAR [18] uses the modelled uncertainty (the uncertainty calculations are based upon SUF/INF symbolic manipulation [19]) to add sensing operations to the plan in order to reduce the uncertainty, associated with an object after an operation, to an acceptable level.

Problem Analysis and Operator Hierarchy (PAOH)

The work focuses on efficient plan generation for assembly operations and uses as an example the construction of a power supply [20]. The approach is to decompose the assembly sequence using domain heuristics (known as *Assembly principles*) to guide the sub-goal expansion and resolve conflicts by posting constraints with the sub-goal with the intension of choosing the correct solution the first time[21]. Constraints are conditions that must be satisfied before an operation sub-goal can be executed. Constraints can either create new sub-goals or restrict existing operations and are determined during plan generation (Note: these also include *Scheduling constraints*). Each operation is named by the main goal it achieves, and the representation explicitly expresses each abstract operation's refinement alternatives. Thereby avoiding backtracking and eliminating the search for candidate operations.

A plan generator (see figure 2.3) follows the Structure analysis stage and uses simple pattern matching to select the primitive assembly operations based upon the component properties, constraints and the modelled status of the work cell. (Note: The architecture uses a blackboard to hold its decisions and this aspect of the approach is discussed further in Chapter 5).

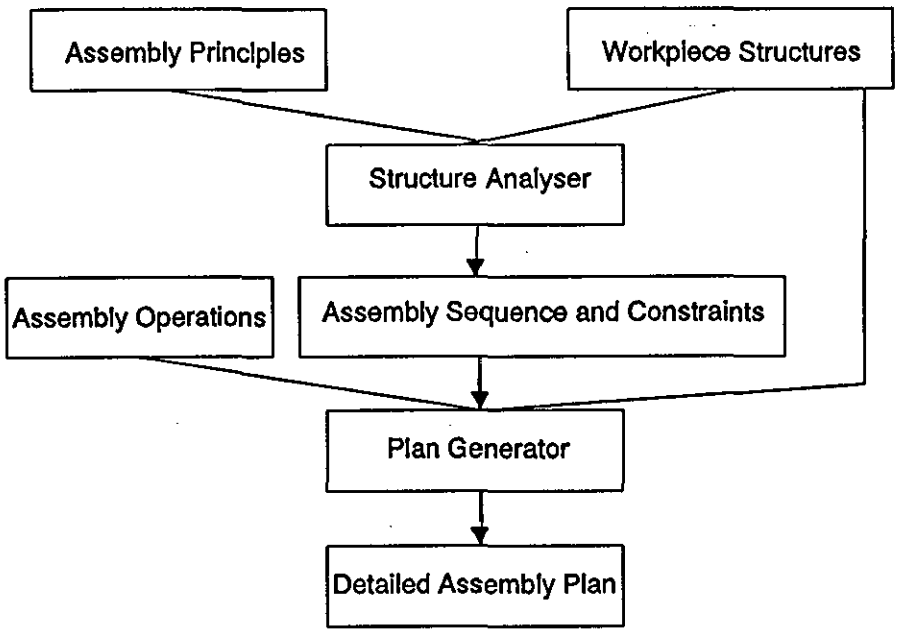


Figure 2.3 PAOH Architecture.

Opportunistic Scheduling (OS)

The approach taken in [22] regards the planning of the assembly operations and scheduling of those operations as two distinct stages. Planning is described using a constraint language [23] which embodies the structural constraints of the system in terms of clauses. The sequencing constraints are expressions composed from the primitive ordering relationship $X < Y$ (indicating that Y must be performed before X), and the logical connectives *AND*, *OR* and *NOT*. The scheduling of the actions uses the constraints and the arrival of parts (which fulfil these constraints) to adapt the choice of next action to the state of the cell and hence, the scheduling is opportunistic.

It was shown in simulation that this approach made more effective use of the time available than traditional sequential scheduling techniques, since the scheduler was able to perform tasks (such as sub-assembly construction or part buffering) when it would otherwise be waiting for the arrival of parts. (Note: the scheduling constraints are implicit and so require expansion during execution)

Discussion

In order that a coherent comparison of the different techniques can be made, the discussion is couched in terms of the stages that are common to each (see figure 2.4) even if some of these are vestigial in a number of cases. However, the discussion

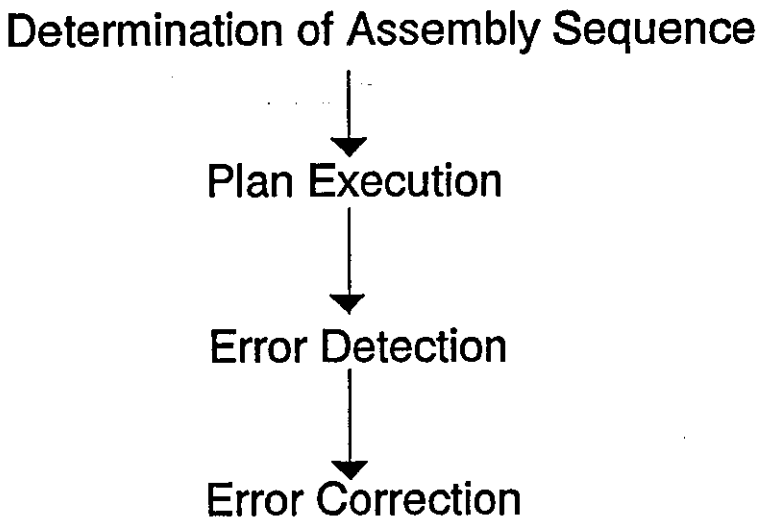


Figure 2.4 Common stages between approaches.

of SA is left until the summary since its architecture does not lend itself to the same stages.

Determination of Assembly Sequences

In the case of the commercial systems a single detailed assembly sequence is produced by the operator which contains predetermined predicted sensory information to verify each action. This immediately limits the success of the assembly to that of a single, but highly likely plan and when errors do occur, the option of graceful termination requiring operator intervention. The plans are produced by decomposing simple task orientated instructions into sequential low level control primitives and so provide the potential for a limited degree of portability.

NNS extends the previous approach by trying to automatically expand a network of all possible sequences and events, to a point where the exact state of the plan could be determined in terms of primitives state values. This provides added flexibility and a means of reasoning about error correction, but begs the questions to what degree of uncertainty and to what level of complexity can this approach be taken. Fox [22] indicated that an exhaustive expansion would not have been feasible in the case of his simple gearbox assembly consisting of 22 pieces.

The hierarchical approaches to sequence generation use hard coded expertise in the form of state machines (HC) or contents of frame slots (Purdue) and so the only means to deal with uncertainty is to increase the precision of its knowledge base. In the case of the Purdue the alternatives are available to reason with in case of error, whereas in HC they are linked directly to the error recovery strategy via the model and sensory modules of the hierarchy.

PAOH makes the knowledge used in the selection of sequence explicit in its *Assembly Principle* heuristics and produces a constrained network of the most likely assembly sequences. This is extensible and allows alternatives to be selected at run time, although there is no method for error correction and detection reported. The use of a precedence diagram by Fox has similar advantages.

Plan Execution

An evolution of sophistication is illustrated by the Thorn, Flymo, OS and YAMS methods of plan execution. The method employed in the Thorn work simply takes the next action from the sequence and executes it, while the Flymo execution is under the control of rules which can be modified. Fox chooses the next action based upon the validity of preconditions and fixed heuristics which allows the plan execution to adapt to part availability. While YAMS delays planning until it is required and hence, not only is the choice of actions influenced by the current state of the world but so is the initial plan generation.

Error Detection

In all systems concerned with error detection the approach is to make use of hard coded techniques attached to a particular level or frame and whose detection and possible action is based upon local knowledge of that operation. Although, this is successful when dealing with a limited degree of uncertainty the use of this technique alone makes the detection of sub-goal interaction difficult to detect and identify precisely, during plan execution.

Error Correction

The forms of error correction available to a system are closely allied to it's representation and the generation of it's assembly sequence. Since in all but PAOH, the reasoning behind the choice of a particular action is implicit and hidden at run time (Note: In PAOH previous reasoning about the plan is not used during the execution of the plan), the systems have only limited forms of dynamic replanning. They are able to reason about sensory information in any other way than is specified by the local error correction routine. For this reason, in the main, error correction takes the form of a graceful termination.

Summary

When comparing these systems with the requirements given in Chapter 1, it can seen that the Purdue and YAMS approaches come closest to achieving the control requirements, since they both are able to generate and schedule plans (*ct1*) and to a limited degree they interpret sensory information (*ct2*). However, they are less successful when coping with errors (*ct3*) and do not address *ct4*. A direct comparison with the knowledge attributes and *kn1-kn4* is not possible as each system was designed to fulfil differing control requirements. YAMS as the only truly distributed system and which had similar communications requirements, can be said to fulfil *cm1* and *cm2*, but as reported, had difficulty maintaining the flexibility of the original design resulting in its failure to achieve *ct4* and *ct3*.

The problems associated with control in the previous approaches derive from the separation of the planning, execution and monitoring processes into discrete stages. This results in useful information generated during one stage being hidden from another (e.g. the intent of a particular action determined in the planning stage is hidden from the monitoring stage, where it could be used to help determine error detection/correction strategies). However, a number of important principles have been highlighted,

Specialised planning techniques (e.g. PAOH) were shown to be more efficient than those based upon domain independent applications and so are applicable to Plethora.

The plan should be a network which includes the reasoning behind the planning decisions, so that during execution more extensive error detection and correction strategies can be performed.

Similarly, decisions taken during execution should be stated explicitly.

Because of imprecise knowledge there must also be a means of reasoning about unexpected events and replanning in the context of the plan.

Selecting the action from the plan network needs to occur opportunistically.

SA has a number of inherent advantages which the other architectures find it difficult to achieve. Firstly, the approach does away with the need for explicit representation of the domain (Chapter 6 discusses the problems associated with such representations) and instead relies upon sensory information and pre-coded simple behaviours to achieve the overall goal. Secondly, its structure naturally allows it to react to external events within the necessary critical times.

The disadvantages are that, at present, the design of such systems are ad hoc and that it would be difficult to introduce the complex goal orientated behaviour required to construct a number of differing assemblies. However, the advantages of SA could be utilised, if it was used as an element in a sensor based strategy and under the control of a goal orientated architecture.

The following chapter describes the approach to control taken in Plethora, which maintains the advantageous characteristics of PAOH and OS while providing a coherent approach to the area of error detection/correction. The approach is based upon incremental opportunism, where a partial plan network, the background reasoning for that plan and sensory information can be used to schedule and monitor the execution of the task. It has the advantage of not enumerating all the possible contingencies and provides a coherent/integrated approach to planning and error recovery.

References

[1] Kak,A.C. Boyer,K.l. Chen,C.h. Safranek,R.j. Yang,H.s.
"A Knowledge-Based Robotic Assembly Cell", IEEE Expert, p63-83, Spring 86.

[2] Williams,A.M. Walters,M. Reay,D.
"A Flexible Assembly Cell", AUTOMAN, May, 85, Birmingham.

[3] Williams,A.M. Walters,M. Reay,D.
"Commercially Available Flexible Assembly Cell", AUTOMAN, May 85, Birmingham.

[4] Selke,K. Swift,G.E. Pugh,A, Davey,S.N. Deacon,G.E.
"Knowledge-Based Robotic Assembly- A Step Further Towards Flexibility", Computer-Aided Engineering Journal, Feb. 87, p62-67.

[5] Selke,K. Shen,H.C. Deacon,G.E. Pugh,A.
"A Strategy for Sensors and Rules in Flexible Robotic Assembly", International Journal of Production Engineering, p100-118, July 88.

[6] Parnack,Van Dyke.
"Manufacturing Experience with Contract Net", Distributed Artificial Intelligence, Chap. 10, ISBN 0-934613-38-9, 87.

[7] R.G. Smith,
"The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver", IEEE Trans. Computers, C-29:12, p1104-1113, 80.

[8] Albus,J.S Barbera,A.J Nageal,R.N.
"Theory and Practice of Hierarchical Control", Proc. 23rd IEEE Comp. Soc. Int. Conf., 87, p18-39.

[9] Barbera,A.J. Albus,J.S Fitzgerald,M.L.
"Hierarchical Control of Robots Using Microcomputers", 15th ISIR, p405-22, 85.

[10] Albus,J.S. Barbera,J.A Fitzgerald,M.L.
"Programming A Hierarchical Robot control System", 12th ISIR, p505-517, 82.

[11] Albus,J.S. Barbera,A.J. Fitzgerald,M.L
"Hierarchical Control For sensory Interactive Robots", 11th ISIR, p497-505, 81.

[12] Shneier,M et al.
"Robot Sensing for Hierarchical Control System", 16th ISIR, p1450-1466, 86.

[13] Albus,J.S. McLean,C.R. Barbera,A.J. Fizgerald,M.L.
"Hierarchical Control For robots in an Automated Factory", 13th ISIR, p13-29 13-24, 83.

[14] Barbera,A.J. Fitzgerald,M.L. Albus,J.S. Haynes,L.S.
"RCS : The NBS Real-time Control System", 16th ISIR, p19-1 19-33, 86.

[15] Chocon,H. R.,Almai
"NNS,A knowledge-Based On-line System For an Assembly Cell", Proc. IEEE Int. Conf. Robotics and Automation, p603-609, 86.

[16] Brooks,R.A.
"A Robust layered Control System for a Mobile Robot",IEEE Journal of Robotics and Automation, Vol. RA-2, No.1, p14-23, March 86.

[17] Booth,C.J.M. Mayhew,J.E.W
"A Sideways Look at Task Decomposition", IEE Colloquium Computing and Control, Knowledge based Environments for Industrial Applications, p6/1-6/3, June 89.

[18] Kak,C.A Hutchinson,S.A
"SPAR: A planner that Satisfies Operational and Geometric Goals in Uncertain Environments", Artificial Intelligence Magazine, vol.11, no.1, p30-61, 90.

[19] Brooks,R.A
"Symbolic Reasoning among 3D Models and 2D Images", Artificial Intelligence, vol.17, p285-348, 80.

[20] Chang,K. Wee,W.G.
"A Knowledge-Based Planning System for Mechanical Assembly using Robots", IEEE Expert, p18-30, Spring 88.

[21] Chang,K. Wee,W.G.
"A Planning Model with Problem Analysis and Operator Hierarchy", IEEE PAMI, Vol.10, No.5, p672-675, Sept. 88.

[22] Fox,B.R Kempf,K.G
"Opportunistic Scheduling For Robotic Assembly", IEEE Int.Conf. Robotics and Automation, p880-889, 85.

[23] Fox,B.R Kempf,K.G
"A Representation for Opportunistic Scheduling", IEEE Int. Conf. Robotics and Automation, p109-115, 86.

Chapter 3
Knowledge Based Control of Resources

This chapter addresses the questions highlighted in the previous chapter which were concerned with the control and scheduling of resources within the system. The chapter describes a knowledge based control technique which tackles many of these problems directly. It proceeds to discuss the drawbacks associated with the technique and the modifications to overcome these difficulties.

The control structure described does not provide a complete solution to the control of resources, but provides a framework within which the requirements of such a system can be met. Exactly how the system will perform depends upon the knowledge and strategies provided by the user. Hence, this framework can be thought of as a real time expert system, which defines the overall architecture and controls the facilities which will be used to plan and execute the action of the cell. Chapter 9 illustrates the use of the framework.

Planning and Control

Planning is defined, by Hayes-Roth [1], as the predetermination of a course of action aimed at achieving some goal. This is followed by a stage which maintains and guides the plan execution to a successful conclusion. This stage Hayes-Roth refers to as control (Fox[2] refers to the same stage as scheduling). Although these stages are conceptually separate they are not necessarily monolithic, and within the domain of the work cell it is advantageous if these stages are interwoven. It will be shown that this helps the system to cope with unpredicted situations and sub-goal interactions which only occur at execution.

Incremental opportunism [1] is a paradigm which can produce this form of interleaving and is the one adopted as the basis for the system's architecture. Incremental Opportunism assumes that planning comprises of the activities of a variety of cognitive "specialists". Each specialist can suggest certain kinds of decisions for incorporation into the plan. These include decisions about : (a) how to

approach the planning problem ; (b) what knowledge bears on the problem; (c) what kinds of actions to try to plan; and (d) how to allocate cognitive resources during planning. The activities of the various specialists are not coordinated in any systematic way. Instead the specialists operate opportunistically, suggesting decisions whenever promising opportunities arise.

Why Incremental Opportunism ?

Incremental Opportunism has the ability to schedule resources based upon past planning action (both successful and unsuccessful), current knowledge and the intent of the plan. Unlike NNS and HC, the reasoning behind the choices made in the planning stages is available to the scheduling mechanism at run time. Hence, these two forms of knowledge which are not available in the previous schemes (i.e. the intent of the plan and original reasoning process) can be used, when selecting between run time actions or modifying the plan during execution. Partially evolved alternatives developed in the planning stage, may become more attractive than the pre-planned course of events when a more complete knowledge of the state is available at run time. The intent, is represented by a record of the reasoning which took place during the plan's generation. This can be used to select the action at run time in the context of the plan's global goals, reducing sub-goal interaction which often occurs when local knowledge is the only basis for selection (as was the case with YAMS).

Thus, the architecture not only provides the system with the advantages of a partial plan network which may be opportunistically executed, but in addition it also has the capacity to incorporate global knowledge into the scheduling decisions.

Fox [2] has already shown that when a form of opportunistic scheduling, based upon the availability of the parts at run time, is compared with simple deterministic strategies (similar to the technique employed in HC) the opportunistic approach results in a higher rate of construction.

The approach here is to extend the opportunistic paradigm to the planning and execution stages and to do this within a single integrated framework, enabling decisions generated at different stages to be incorporated within any other stage. The additional knowledge available to the scheduler will make it possible to adapt to

circumstances other than just part availability. Since, by making use of the knowledge of previous stages it may decide to, abandon the whole assembly based upon global knowledge of the problem; try to patch up the problem by local reasoning about the alternative solutions to a particular goal; or restart a separate assembly with the parts already correctly assembled. These alternatives are not possible if the scheduling is based upon local knowledge alone. In a later example, Fox's technique will be shown to be encompassed as a sub-technique within Plethora's more comprehensive architecture.

The Mechanism of an Incremental Opportunism System

There have been many systems based upon incremental opportunism (e.g. speech recognition [3-5], signal processing [6]) these systems have culminated in the BB1 architecture [7]. BB1 provides a more rigorous definition of its behavioural goals and has been shown to encompass the features of previous systems (e.g meta-planning[8]). Since the aim of Plethora is to extend the traditional incremental opportunism to include the execution of the plan, BB1 is taken as the basis for Plethora's blackboard. What follows is a brief description of the essential details of BB1, how it fulfils its behavioural goals, the problems and limitations of the architecture as a solution to the Fitter problem, and the extensions employed within Plethora. The behavioural goals of BB1 are

1. Make explicit the control decisions that solve the control problem.

2. Decide what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.

3. Adopt variable grain-size control heuristics.

4. Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.

5. Adopt, retain and discard individual control heuristics in response to problem solving situations.

6. Decide how to integrate multiple control heuristics of varying importance.

7. Dynamically plan strategic sequences of actions.

8. Reason about the relative priorities of domain and control actions.

BB1 Architecture

Opportunistic planning is a co-operative scheme comprising of a number of cognitive or Knowledge Specialists (KS), which suggest decisions to be incorporated in to the plan, and a global data structure (known as the blackboard,BB) on which these decisions are kept.

Knowledge specialists

KSs proffer what knowledge they have to the planning process, whenever they recognise a goal or particular conjunction of goals and states. They do this by returning a Knowledge Source Activation Record (KSAR) to a central selection mechanism. This KSAR holds the context within which the KS volunteered its resources. Each specialist suggests decisions at a specific level of abstraction within

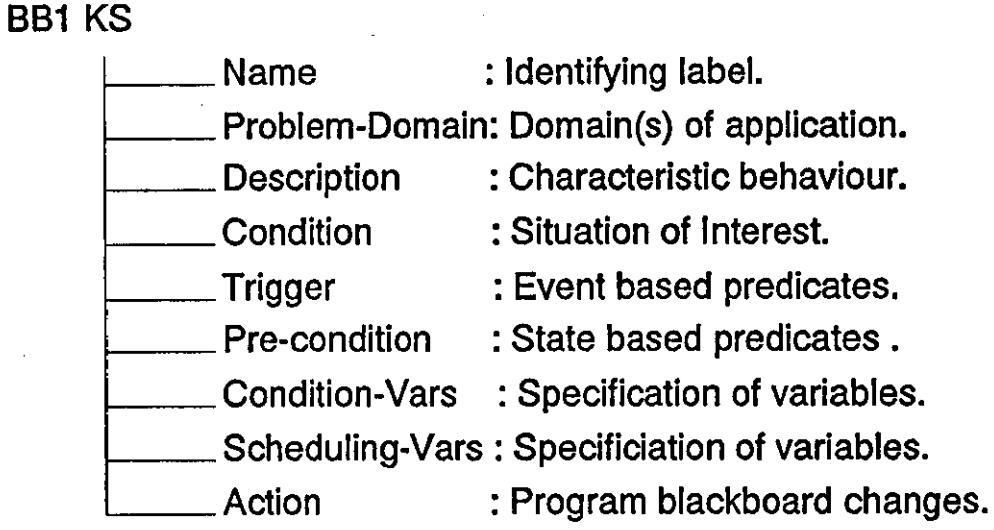


Figure 3.0 BB1 KS structure.

the plan, and they make their suggestions whenever the opportunity arises. KSs take the form of modified pattern-directed condition-action rules (see figure 3.0) and suggest such things as, what action to plan next and how to allocate resources. The Trigger and Precondition must be true before the knowledge is applicable and the Action defines the behaviour of the KS.

The system does not distinguish between the selection of planning or scheduling decisions when choosing its next action. This makes it possible to swap between planning and control stages as the plan progresses. It also makes it possible for the planner to swap between search strategies during a particular stage. It could initially apply a successive refinement (i.e. top down) strategy during the preliminary stages of planning, by preferring gradually less abstract decisions and so allowing constraints discovered in the planning process to guide the selection of the decisions. And then apply bottom-up strategies during the scheduling phases, which allow low-level refinements of the previous plan or expansion of an alternative plan based upon temporal constraints or sensory information. A decision in BB1 has the fields shown in figure 3.1.

Control Decision

- Name : Identifying level and number.
- Goal : Prescribed action.
- Criterion : Expiration Condition.
- Weight : Goal Importance.
- Rationale: Reason for goal.
- Creator : KSAR that created the decision.
- Source : Triggering decision.
- Type : Role in control plan.
- Status : Function in the control plan.
- First-Cycle: First operative cycle.
- Last-Cycle: Last operative cycle.

Figure 3.1 The fields of a decision in the BB1 system.

Blackboard Structure

Most opportunistic systems record the decisions on a common global data structure, known as a blackboard. KSs communicate and interact with the blackboard, and in most systems they cannot communicate with each other (unlike the Contract Net in YAMS). The blackboard is partitioned into a number of planes, each of which contains conceptually different categories of decisions. The planes are then further partitioned into levels of abstraction. This serves two purposes. Firstly, it provides a conceptual taxonomy of decisions. Secondly, it restricts the number of decisions that a KS must examine before making a contribution. The BB1 blackboard has two planes, the Control and Domain planes, each of which is partitioned in a number of levels shown in figures 3.2. The Control plane holds those decisions which control the search, while the Domain plane holds those which directly solve the domain objective of the system, known as the *Outcome*.

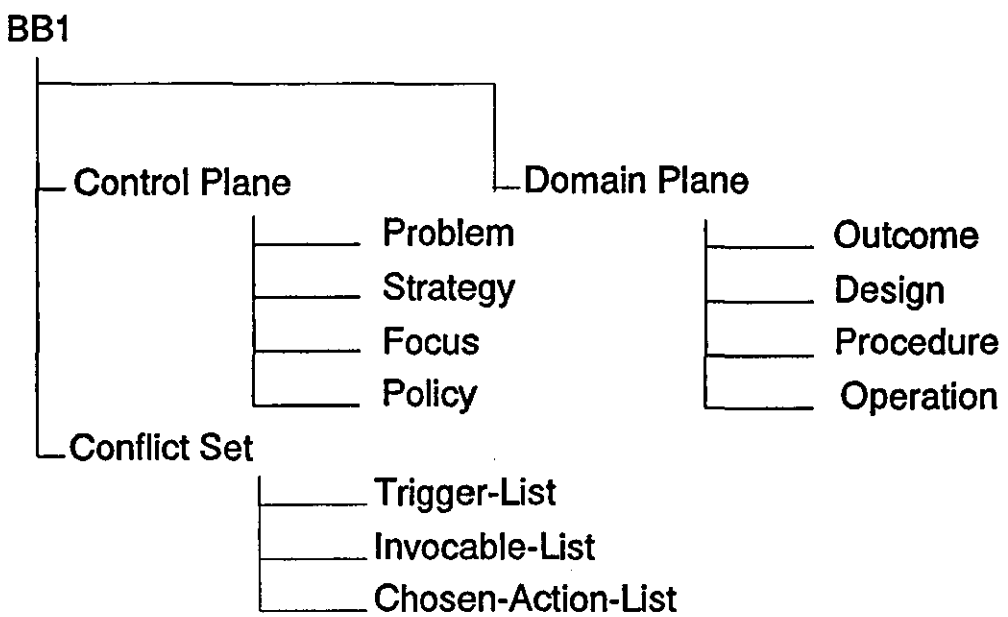


Figure 3.2 BB1 Blackboard Structure.

The categories of decisions contained within the levels of Control and Domain planes are as follows,

Problem decisions represent the problem the system has decided to solve and guides the entire problem solving episode.

Strategy decisions establish general sequential plans for problem solving.

Focus decisions are used to rate KSARs and influence the final choice of KSAR. They are volunteered by strategic KSs.

Policy decisions are long term and general scheduling criteria used to rate KSARs.

Outcome decisions represent the domain task to be solved.

Design decisions determine the general partition or approach to the Outcomes solution.

Procedure decisions are sequences of tasks, which are determined by the design, to solve the Outcome.

Operation decisions describe the means by which a procedure decision is to be achieved.

Under the control of an executive, the planning process proceeds through a series of cycles during which, KSs execute their actions, record their changes on the blackboard and in doing so, trigger new KSs for the next cycle. This continues incrementally until a plan is generated which fulfils the evaluation criteria found in the *Problem* and *Outcome* goals. The executive, which controls the problem solving cycle, is itself a set of modified KSs known as the *BASIC-KSs*. Thus even the operation of the basic inference mechanism can be modified in accordance with the state of the plan.

Conflict Resolution

The selection of which *KSARs* action is to performed, is made through the control decisions entered in the *Strategy*, *Focus* and *Policy* levels of Control plane of the blackboard.

Strategies

Strategies are used to control entries at the *Focus* level. A *Strategy* control decision may encompass a sequence of *Focus* level decisions made as the strategy and plan progress. The *Strategy* decisions therefore do not directly influence the

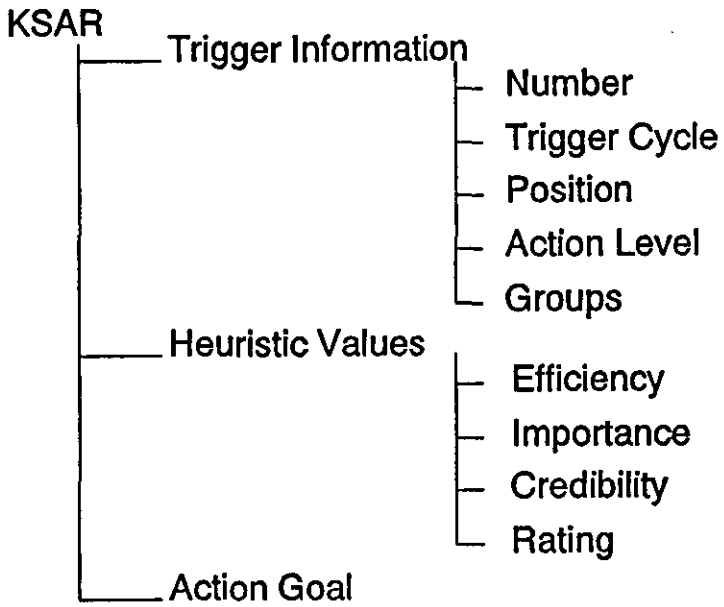


Figure 3.3 KSAR fields.

scheduling decisions, but influence them indirectly through the *Focus* decisions they implement.

Foci

Focus decisions establish local problem solving objectives which prefer the execution of KSARs with particular attributes. They are used to modify the ratings of the KSARs at the beginning of each problem solving cycle. Focus decisions are temporary and several complementary or competing *Focus* decisions may operate simultaneously. *Policy* decisions establish global scheduling decisions, but in contrast to *Focus* decisions, usually remain active throughout the life of the plan.

Conflict Set Generation

In BB1 the set of all pending KSARs (see figure 3.3 for details of KSAR fields) are held within the TO-DO-SET. This is formed of two lists, the *Trigger-List* and the *Invocable-List*. The *Trigger-List* holds all KSARs that have been triggered by blackboard activity. The *Invocable-List* holds those KSARs which have been triggered and whose preconditions are true (i.e. whose original context is still valid and so the

KSs action is still appropriate). If during the problem solving process *Invocable-List* KSARs are no longer invocable, they are transferred to the *Trigger-List*. Hence, the precondition field allows the problem solving behaviour to adapt to the dynamic effects of the plan and perceived state of the world, by continually changing the set of viable actions from which the next action can be chosen.

Attention Focusing

Figure 3.4 illustrates how a simple depth first search strategy could be implemented using the attention focusing technique in BB1 (see [9] for a detailed

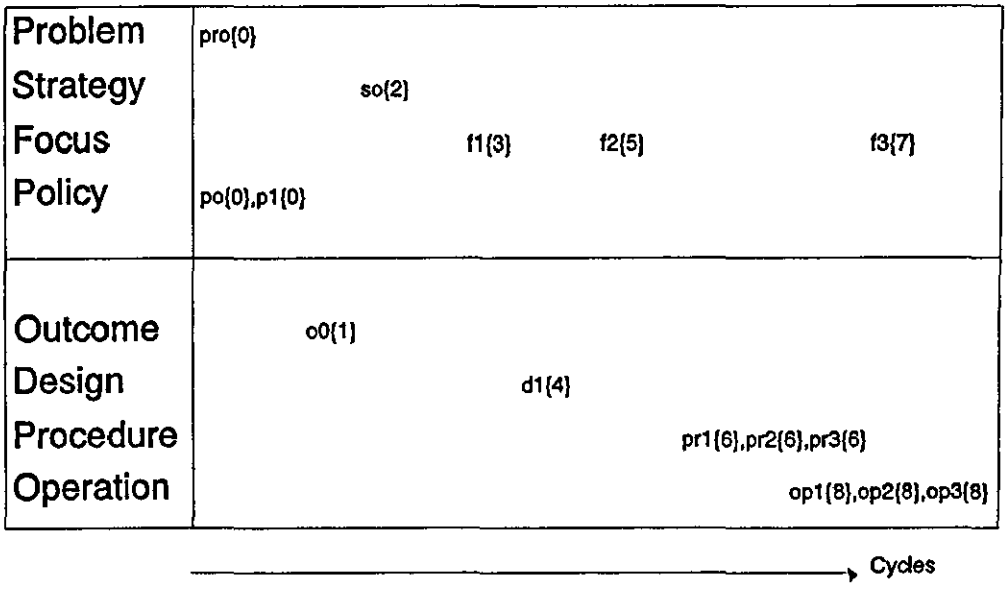


Figure 3.4 BB1 blackboard contents during depth first search.

description of the technique). Assume that initially *Problem* PR0 and the *Polices* P0 and P1 are already on the blackboard. After O0 is entered a number of design KSARs volunteer to solve O0 and S0 volunteers to solve the conjunction of PR0 and O0.

However, S0 becomes the *Chosen-Action* and is executed, since it has the highest rating under the *Polices* on the blackboard. This results in a *Focus* entry preferring KSARs which solve the *Design* level problems of O0. On the next cycle, the previously volunteered *Design* KSARs are preferred and one of them, D1, is selected based upon its efficiency rating. This fulfils the *Focus's* criteria and a new *Focus,* preferring KSARs with action levels at the procedure level, is volunteered by S0 and entered on the blackboard. This alternation between the selection of domain and control decisions continues until the *Strategy* terminates. (Note: the numbers contained in the braces in figure 3.4 indicate the blackboard cycle during which the decision was entered).

At any time during this process S0 could have been replaced or augmented by another *Strategy,* if that *Strategy* had recognised a context within the blackboard for which its knowledge would have been suitable. It is in this manner that the focus of attention can be made to change in tune with each new decision.

Summary

Incremental opportunism appears to be a comprehensive and complex model for planning, since not only can it reason about its outcome, but it can also reason about the strategies and inference mechanisms which are to be used at different points within the plan. The complexity accounts for the wide range of possible planning styles it can accommodate (see [7] for illustrations), and the way it can be made to adapt to the state of the plan.

The ability to swap between planning and scheduling within a single framework is of particular importance in the work cell domain, since sensory data could then be used to influence both the planning and scheduling decisions.

Apparent Complexity

The disadvantage of the adopting a blackboard approach to on-line reasoning appears to be the increased time spent by the blackboard during its conflict resolution scheme, when it is compared with simpler schemes. The following section shows that this concern is more apparent than real. Firstly, by discussing the efficacy of other planning representations and secondly, the mechanism by which incremental opportunism is implemented.

Plan Representations

In simpler control schemes (see [10] for a overview) the state of the world and plan are represented as lists which hold the domain state, changes are made by rules which add and delete states. These rules model the actions made on the world. Selection of the applicable rules is made based upon the contents of the add states and in its simplest form, implicit conflict resolution using fixed heuristics and a depth first search are used to secure a solution. When backtracking to a previous point in the plan, the effect of the rule must be retracted. This is done by adding the properties in the delete list and deleting the properties in the add list of the rule. More complicated control strategies employ general heuristics to guide the selection. These are designed to reduce sub-goal interaction or leave goal ordering to the last opportunity (known as the least-commitment strategy [11]).

However, since only the changes to the current state are recorded, in order to follow a new line of reasoning the old line must be discarded by retracting each of the actions it made on the world. There will be no trace of the reasoning which took place, and the effort expended when solving a particular sub-problem might be expended yet again in the new line.

When such a paradigm is applied to large state space, the cost of retraction and backtracking becomes so significant that it dominates the systems response time [10]. This is the case for the Fitter problem during both planning and execution. Even for the simplest practical assembly, there are a large number of alternative orders in which the assembly may be constructed (see Fox [2] for a quantitative discussion of this point). Also during execution, there is uncertainty associated with each action or

sensed event. Hence, the Fitter Problem is a large state space problem.

In blackboard schemes, the representation and generation of the conflict sets are similar to those used in the simpler schemes, but the determination of the chosen rule for execution is very much more sophisticated. The methods of selection are explicitly represented as *Strategies*, which allows the system to reason about its own reasoning and for this to be changed accordingly (i.e. meta-reasoning). The activity of particular *Strategies* can be made dependent upon the state of the plan, external events or the availability of operators. Since these control *Strategies* are contained within the control plane of the blackboard as decisions, they also can be employed or discarded in response to changes in the planning process or external events. The adaptability of the control *Strategies*, and explicit representation of previous or alternative planning, reduces the search time in large state space problems by tailoring the search technique to the circumstances of the plan.

The blackboard structure also allows the system to make use of the results of competing solutions. Once some item has been discovered and entered on the blackboard it can be used by any KS, this reduces the likelihood of rediscovering or re-evaluating the same piece of information over and over again. Hence, the comparative response time between the blackboard and a simpler approach is not as critical as first imagined. This is particularly so during execution, where the explicit representation of previous reasoning can be used to reduce the computation necessary at run time.

Implementation

It has been illustrated [12] that with a suitable choice of implementation the absolute response of such a system can be improved in two ways. Firstly, if the ratio of conflict resolution time to action execution time is kept to 1\10, then the action time dominates the response of this system. (Note: this is not surprising, since this is also the rule of thumb used in the design of data flow machines [13]). So, the KSs need to be large grained, which suits the domain characteristics.

Secondly, [12] showed how the elements within the basic inference cycle of the blackboard can be made to run asynchronously, by overlapping the conflict set generation, conflict resolution and execution of KS action, and that this resulted in shorter response times.

The "pipelining" of the system's basic cycle can be extended further, given that the system is partitioned so that KSs are distributed across a number of processors, and that changes in the BB are broadcast to the processors. Since this would now allow the generation of the KSARs (which takes place each cycle) to take place in parallel, rather than sequentially as is the case in traditional systems (e.g. BB1). Once again this suits the distributed nature of the domain.

Summary

It has been shown in [7] that the 8 inferential goals are achieved by the blackboard architecture, and also that these goals are sufficient for the planning stages of Plethora. However, the architecture does not deal with the additional problems of execution of the plan in an uncertain environment, where errors and asynchronous events not under the control of the architecture can occur. Thus, since it is argued in Chapter 2 that many of these goals are also adequate for execution, Plethora makes additions to the architecture which maintains them during execution. The following section describes the additions to BB1 found in Plethora.

Plethora's Blackboard

In BB1 there is no means of representing the progress of the execution of the plan, this will be necessary if any of the BB1 behavioural goals are to be supported during execution. Hence, Plethora has an additional plane within the blackboard on which decisions and external events are reported. It uses this plane in conjunction with a slightly modified blackboard execution cycle, to extend the mechanisms which implement the behavioural goals found in the planning to the execution stage. Taking this approach (rather than providing a separate control plane for the execution of the plan) allows Plethora to operate on the decisions made in this area of the blackboard in the same manner as in the other planes.

Thus decisions can be made between whether to reason about the plan or its state, react to changing events, or instigate new actions, in the light of all the knowledge contained on the Blackboard and current state of the external world. This is also performed in a seamless fashion, since the same selection mechanism is used for all phases of Plethora's operation. Hence, the full power of the blackboard architecture found in planning can be applied to the execution phase (so maintaining the behavioural goals during execution).

Execution Plane

The levels within the execution plane are *Event*, *Reason*, *Error* and *Solution*

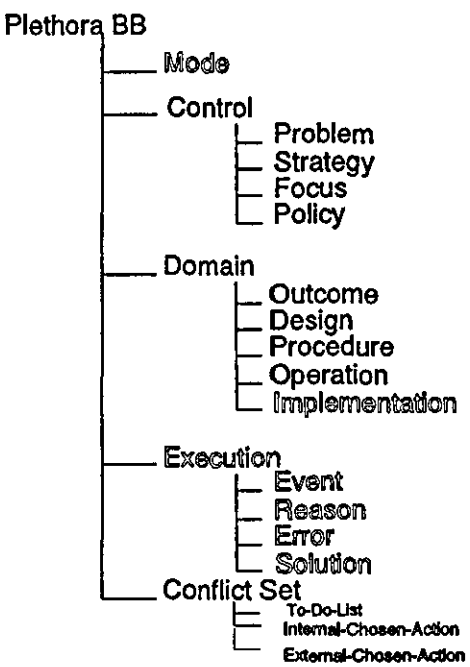


Figure 3.5 Plethora's BB Structure.

forming the overall BB structure shown in figure 3.5, where the additions to the structure are outlined.

Event Level

The event level holds decisions which represent the occurrence of asynchronous events (i.e. external to those directly under the control of the blackboard). They might be reported by the *Chosen-Action* KS during its execution or by another KS which as a result of the *Chosen-Action* detects changing in the cell. An example of two such KSs would be, a KS which controls a gripper and another which monitors the state of proximity sensors attached to the jaws of the gripper. The combined state of these sensors can be used to indicate the presence of an object within the jaws (see [14] for a more detailed description of this technique). Hence, given that the *Chosen-Action* is to grasp, the KS responds by closing the jaws and reports an *Event* confirming this fact. The sensor KS will also report an *Event* based upon the condition of it's sensors, and so may serve to confirm or deny that the object was grasped. Once an *Event* has been entered on the Blackboard it is broadcast in the same manner as other decisions.

The possibility of breaking the conventional feedback loop between sensor and actuator, to include the blackboard has a two fold advantage. Firstly, it allows sensors to be changed without modification to the grasp KS and secondly, allows the system to apply all the knowledge available to Plethora if an error had occurred. (Note: It is initially assumed by Plethora that an *Event* occurs as a result of the current chosen action and so is linked to the *Chosen-Action.* Although, this can be modified by a KS if it has specific knowledge to the contrary.)

Reason Level

These decisions allow KSs to postulate *Reasons* for an *Event* in the form of a goal, which it thinks has been achieved and given rise to the *Event* (i.e. the object has slipped out of the jaws or was never there in the first place).

Error Level

These decisions indicate and describe a possible error which has occurred during the execution of the chosen action. These can be, entered by the chosen action KS itself, identified by other KSs as a result of knowledge of the goal which is to be satisfied by the *Chosen-Action*, or as a result of *Events* and *Reasons* which have

occurred during its execution. A number of possible competing *Error* decisions can be associated with an *Event/Reason* or a single *Error* might encompass a number of *Events*. In the grasping example the proximity sensor KS might have reported an *Event* indicating the gripper was empty, a *Reason* KS proffered that "the object has slipped from the jaws", and an *Error* decision produced which indicates the violation of the grasp goal (i.e. the gripper cannot be closed and be empty and still have grasped an object).

Solution Level

Solutions proffer goals to correct those in *Error decisions*, and are used to trigger *Strategies* in the Control plane to coordinate a solution. This allows the choice of *Solution* and *Strategy* to be made in context of the complete plan rather than in local isolation. As a result it becomes possible to base further action upon previous execution history, the intent of a higher goal or a previously existing constraint (e.g. a more important goal should be achieved, in preference to correcting this error). Examples of solution KSs reasoning might include, all chosen actions associated with the chosen solution to the *Operative* goal have had problems, so choose another method of achieving the implementation goal; or relocate the object before trying again.

Hence, the levels mirror the explicit reasoning for the execution of actions and resolution of external events, found in the Domain plane when planning.

Chosen External Action

However, if this was the complete structure of the BB there would be no means of reasoning during the execution of an action which had its effects within the cell. Since the *Chosen-Action* level would hold the external action's KSAR until it's completion. Hence, there would be no means for the blackboard to make use of the *Events* which occur during the action, as this would require additions to the *Chosen-Action* level.

To this end the *Chosen-Action* level found in BB1 is split into two further levels within Plethora, the *Chosen-Internal-Action* and *Chosen-External-Action*. The

former holds the KSARs which only effect the contents of Plethora and the latter, those which have external effects upon the cell (e.g. moving the manipulator).

Thus during the execution of an external action Plethora can still reason about the progress of the task or plan further actions. Such decisions would be made using the *Chosen-Internal-Action* level, while the external action KSAR would remain undisturbed on the *Chosen-External-Action* level.

External actions can be of the order of seconds and so this pipelining can contribute significantly to the efficiency of Plethora, especially when replanning an action after an error. It also enables Plethora to react to errors before the completion of the external action.

Confidence

In BB1 each decision has associated with it a *Weight*. The *Weight* indicates the importance of achieving a goal and is initially determined by the instigating KS during planning. However, Plethora unlike BB1, reasons about the execution of the plan based upon information provided by sensors. Sensory information is local knowledge and its interpretation is prone to error, and often when determining the success of a goal it may be required to integrate a number of such sources (A similar process when applied to data is known in the literature as sensor or data fusion). Hence, there is a need for a measure of achievement of a sensory goal, which is not found in BB1. This is indicated in Plethora by the *Confidence* value of a decision. *Confidence* represents the KSs percentage faith in the goal, based upon its model of interpretation.

The *Confidence* measure, particular goal and the time of measurement, can be used as criteria when choosing between conflicting sensor estimates of the same property. The source of such measures on the BB comes via the *Event* or *External-Chosen-Actions* and are propagated to other decisions under the control of KSs and the usual conflict resolution mechanism. A number of such propagation methods have been proposed given the availability of a measure of *Confidence,* including the use of error manifolds [15], heuristics [16] and probability theory [17].

Determining Confidence Values.

Integrating sensor data from a number of sources to form a consistent conclusion is a difficult process. This is due in part to the fact that sensory information is always uncertain and usually partial, but also because it is often geometrically or geographically incomparable with other sensory data.

A number of techniques (e.g. [18]) use detailed knowledge of individual sensors and correlation between sensor characteristics, to derive heuristics which guide the fusing of different sources of information. However, as successful as these techniques have been for particular cases, they do not provide a general means of integrating sensor data, which is necessary in Plethora. In order for this to be achieved there needs to be a quantitative measure of sensor response so that data from different sources can be processed within a single framework.

The work undertaken in [19] attempts to do this by determining for each sensor, probability density functions which estimate that a feature in parameter space corresponds to a sensed geometric object (where object here refers to straight lines, planes etc). It uses these, the sensor data, a model of sensor noise and error, a model of its dependence on the observations of other sensors in the system and its internal state, to calculate estimates of geometric objects. It then goes on to use a metric, known as the utility function, to order the likelihood of these decisions. A team of such sensors pass this information between them and eventually arrive at an agreement using Bayesian decision making and a bargaining technique [20].

A related technique [21] creates a distance matrix based upon the separation between the probability density functions for particular sensors and features, from which it then derives a distance measure which is used as the confidence value.

However, this approach has a number of problems when applied to different forms of sensors, rather than a large group of identical sensors. Firstly, the determination of the necessary probability density functions is not trivial, especially in the case of the one-to-many mapping found in the camera. Secondly, the decision making within the team of sensors is based upon geometric features, and so requires

an additional mapping from feature space. Thirdly, since there will be different forms and degrees of uncertainty associated with data derived form multiple sensors, there is a need to know a-priori, the interrelationships between features of different sensor data (making *ct4* and *cm3* difficult to achieve).

The method taken here does not use probability density functions as the basis for decision making, but is based upon inequalities representing features, and a model describing the sensory data and its processing. Figure 3.6 illustrates the typical constituents and data flow within a sensory KS required to produce a *Confidence* value. Here it is assumed that the feature vector $\underline{F}$, determined as a result of the feature extraction process, is a vector of inequalities of the form

$$\underline{F} = \{f1_1 <= x_1 <= f2_1, \ldots\ldots, f1_n <= x_n <= f2_n\}^T$$

where each scalar value x_n is bounded by two limits (f_{n1}, f_{n2}) and that each vector corresponds to a detected object. The predicted feature vector $\underline{G}$ is also assumed to be of the same form, but is calculated from predicted data on the assumption that the goal that it is to verify is correct. For each feature vector element x_n, a confidence value is calculated against the corresponding predicted value, resulting in a vector of confidence values. From this confidence vector an overall confidence value (C) for the vector is calculated. The vector with the highest confidence is used to determine the satisfaction of the goal and the KSs final confidence value.

In keeping with the goal directed behaviour of Plethora, integration of information takes place on the BB in terms of goals and *Confidence* values. This has a number of advantages. Firstly, inequalities representing possible variation are readily determined from sensor data and a method to manipulate such information already exists (see [22] which derives and manipulates such data for scene analysis derived from camera photographs). Secondly, the use of a goal language and the posting of the results on the BB, allows the use of other than purely statistical techniques (see [23] for a simple and effective technique). Thirdly, it provides a natural means of integrating information with other KSs in Plethora for which inequality information already exists. Fourthly, it does not rely on a-priori knowledge

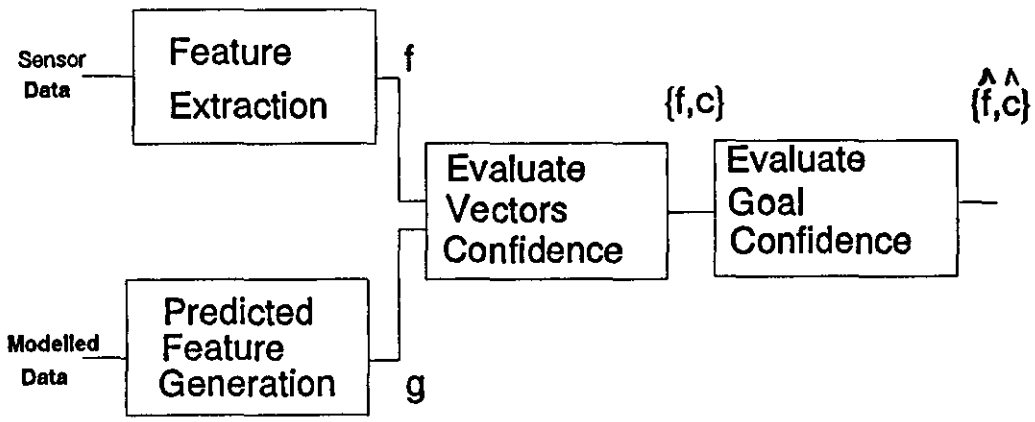


Figure 3.6 Schematic of data flow in a typical sensory KS.

of the statistical correlations between sensor characteristics (maintaining the modularity required by *ct4* and *cm3).*

However, there still remains the requirement for a measure of the agreement between an individual sensor scalar value and its predicted value.

Confidence Function

The derivation of the confidence function can be best explained by considering single features generated by the feature extraction and feature generation processes in figure 3.7. Where f_1 and f_2 indicate the range of the extracted feature, and f_{hat} its nominal value. Since there is no indication of the probability of the actual value, it is assumed that any value within the range is equally likely. Thus an important constituent of the *Confidence* value is the degree of overlap between actual and predicted ranges. In order to cope with cases where the overlap is large but is a small portion of either range, two quantities indicating the degree of overlap are used, the overlap normalised to the predicted range and the overlap normalised to the feature range. Since either case will reduce the *Confidence* in the match, these are equally

weighted expressions in the final *Confidence* expression.

A further distinction between cases is provided by an additional term, based upon the separation of the nominal values normalised to the predicted range. This leads to a maximum *Confidence* value of *1*, which occurs when the ranges match exactly and the nominal values coincide, and a minimum value of *0* when there is no overlap and the nominal values are separated by a predicted range. The algorithm used to calculate the Confidence value is given below.

Overlap:

No overlap	: Overlap=0
G enveloped or equal to F	: Overlap=g_2-g_1
f_2 within G but not f_1	: Overlap=f_2-g_1
f_1 within G but not f_2	: Overlap=g_2-f_1
F enveloped within G	: Overlap=f_2-f_1

Goal Normalised Overlap:

g_2-g_1=0 : Gno= 0

g_2-g_1< >0: Gno= Overlap/(g_2-g_1)

Feature Normalised Overlap:

f_2-f_1=0 : Fno=0

f_2-f_1< >0: Fno= Overlap/(f_2-f_1)

Nominal Normalised Distance:

g_2-g_1=0 : Nnd=0

g_2-g_1< >0: Nnd=min{ 1, (1-(abs(f_{hat}-g_{hat})/g_2-g_1)) }

Confidence = { Gno + Fno + Nnd }/3

Figures 3.9 and 3.10 illustrate how the value of the function varies for different degrees of separation and position of *fhat* within the range F (i.e. *a* and *b* respectively, as shown in figure 3.8).

The process of determining the overall *Confidence* value of a vector of *Confidence* values is similar to determining the overall reliability of a system which has a number of parallel elements each with individual probabilities of success.

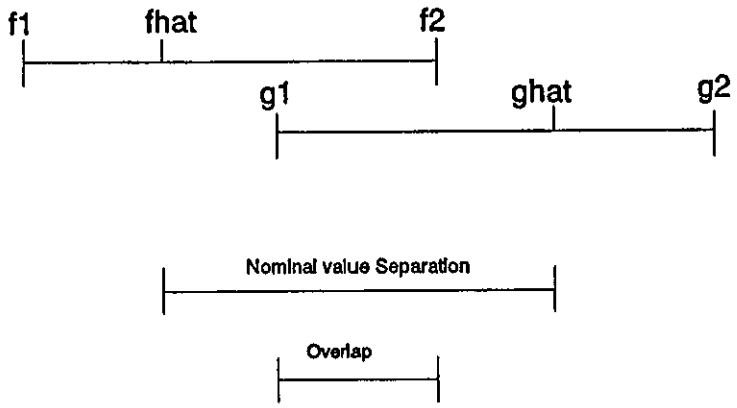


Figure 3.7 General case of overlapping ranges.

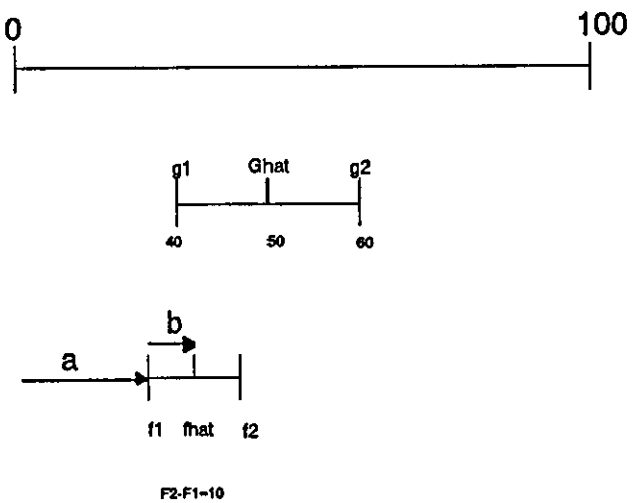


Figure 3.8 Confidence variation with Feature parameters.

Hence, the overall *Confidence* value (C) of a vector's individual confidence values $\{c_1....c_n\}$ is defined to be

$$C = \{ 1 - \{(1-c_1).(1-c_2)....(1-c_n)\}\}$$

The *Confidence* value of a vector can be used to distinguish between a number of features detected by the sensor. The feature vector with the largest *Confidence* corresponds to the best match and whose features will be used to determine the value for the final goal. Examples of its use are given in Chapter 9.

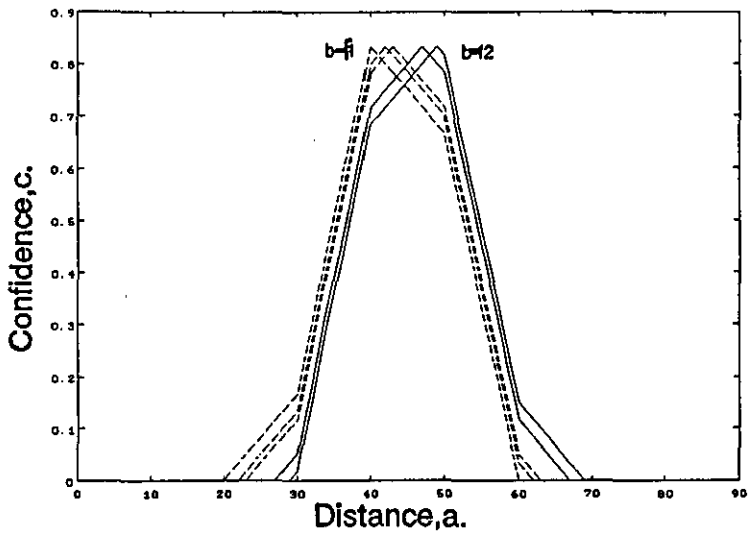


Figure 3.9 Variation of Confidence with range position and nominal value position.

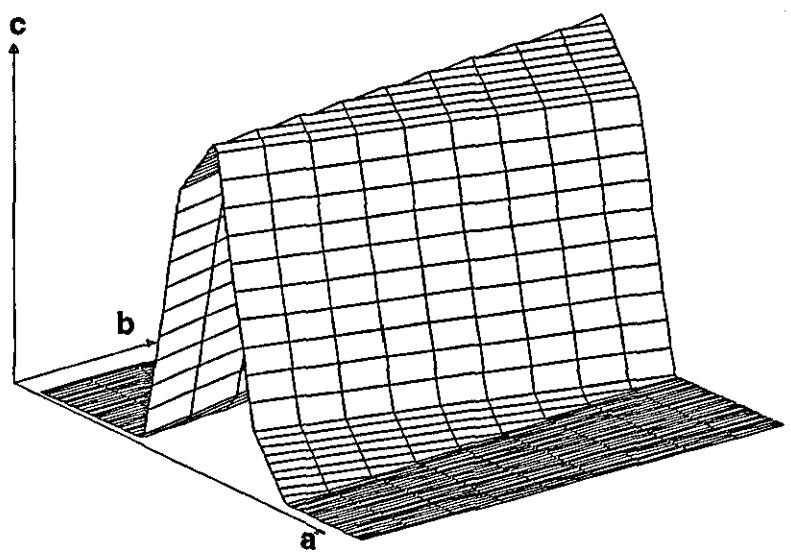


Figure 3.10 A mesh plot of Confidence variation with range position and nominal value position.

Pre-Condition/Action

In the modified BB of Plethora, the preconditions are held within the KS itself and not transmitted to the blackboard along with the KSAR. This arrangement simplifies the communication between the blackboard and KS. It also allows all such tests to be carried out in parallel, and removes the need for an *Invocable-List.* Within Plethora's KSs, there is an extra predicate which is used at execution time, it

combines the operation of a precondition test and the action and is called *Precondition+Action*. This is used to ensure that the action is still valid immediately before its execution. Its use removes those problems which occur when there is a delay between proposing to execute an action and the actual time at which the action takes place (during which interval the state of the cell may have changed making the action invalid).

Although the validity of the *Action* and *Precondition* fields must be checked dynamically, it is advantageous that the *Trigger* of a KS maintains a history of interested events. It can then use this as a context in which it can interpret each new changes of state when determining whether it should volunteer a KSAR. The use of this locally held information radically reduces its need for access to the BB, which would otherwise be necessary if the KS were to check the validity of states each time an interesting change of state occurred. Thus KSs in Plethora are more independent than those found in BB1, whose trigger evaluation can take in parallel. This is of particular advantage in a distributed environment where the amount of communication traffic plays a dominant role in the system's performance.

The action variables of BB1 are replaced with instantiated goal statements that are interpreted by the *Action* of the KS.

Groups of KSs

To increase the efficiency and provide a form of taxonomy for the KSs, each KS can belong to one or more groups. A group consists of KSs with related objectives or requirements. A broadcast to a group ensures only the interested KSs are informed of a particular event. An example in the use of a group is the *Groups* field in a KSAR. This field holds the groups of KSs which might be interested in the resulting changes to the blackboard. When its action is executed, the BB informs the members of these groups of the changes to the blackboard. Groups of KSs can also be associated with activity within particular levels of the blackboard.

Implementation Level

An extra level, below the operation level, has been added in the Domain plane of the blackboard. It holds the decisions which indicate exactly how an *Operation* goal is to be implemented within this cell (i.e. What particular method of identification is to be used?). Hence, the level is filled with KSARs extracted from the *To-Do-List,* as potential *External-Chosen-Actions* to fulfil a particular *Operational* goal.

Modes

The modified version of the blackboard has *Modes*. These *Modes* indicate the form of present behaviour undertaken by the blackboard, and reflect the state of the *Basic-KS* state machine. Modes are changed by the *Basic-KS* which recognises the end of a particular phase. The mode of the system is used as a debugging aid and can be interrogated by *Focus* goals to adapt the behaviour of *Strategies* to the current *Mode* of operation (e.g. planning to scheduling).

Temporal Ordering and Dependence

Temporal ordering and goal/sub-goal dependence are indicated by separate fields in the decision, and are regarded as orthogonal sets of relationships. Temporal ordering is provided by unidirectional *Before/After* links between decisions and is used to indicate the set of *Viable* goals.

A goal is said to be *Viable* if three conditions hold. Firstly, all the goals immediately *Before* it must be *Completed* (i.e. their status is *Completed).* Secondly, its parent goal is *Viable* and thirdly, the goal is not already *Completed* (see figure 3.11). The viability of a goal is indicated in a separate field and updated by the BB whenever a decision changes status and dynamically changes with the state of the work cell.

Dependency links describe which goals must be or might be used to satisfy a parent goal. The plans goal dependency is represented as a conventional AND/OR tree where the nodes of the tree are the decisions on the blackboard. Decisions are said to be *Planned,* if all dependent decisions within an OR branch are *Planned* or there is a complete tree from the decision with terminal nodes in the *Implementation* level.

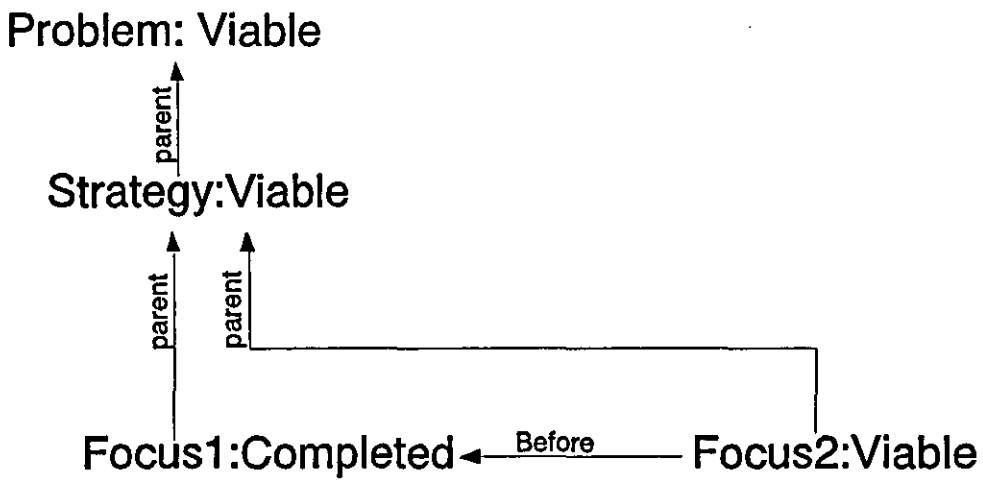


Figure 3.11 A Viable decision.

Focusing Attention

The Strategy/Focus mechanism has also been modified in order to operate more efficiently within a distributed environment, and uses the *Viability* of goals to control the sequence of *Foci* in a *Strategy*.

Each *Focus* and *Strategy* decision has a termination condition associated with its goal, at the beginning of every cycle these termination conditions are evaluated. If the *Focus* has terminated, the *Basis-KS* marks it *Completed* and updates the *Viable* decisions. If a *Strategy* has terminated, the *Strategy* and all its *Foci* have their status changed to *Completed*, and once again the *Viability* of decisions is updated.

In the modified implementation of Attention Focusing, when a *Strategy* is implemented it enters all the *Foci* that will be necessary for its fulfilment, onto the BB in one go. The order of these *Foci* is indicated by their *Before/After* links, the first *Focus* will have no *Before* links and so is *Viable*. Only *Viable Foci* are used to rate the KSARs, thus the others have no effect until the first *Focus* terminates, when it does so it makes the next focus *Viable*.

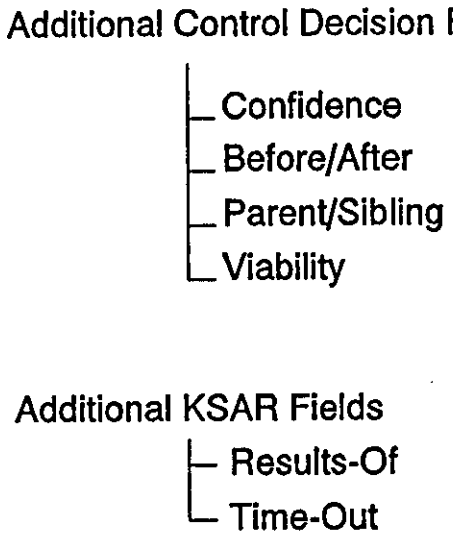


Figure 3.12 Additional KSAR and CD fields.

In BB1 the next *Focus* decision is determined as the result of the evaluation of a function contained within the *Strategy* decision. When this method was tried in Plethora, it caused significant delays. The present approach reduces the time taken to implement attention focusing and provides a unified means of representing temporal constraints on the Control, Domain and Execution planes.

The *Focus* goal and *Strategy/Focus/Policy* termination conditions are tokenised and evaluated by the *Basic-KS* locally during each cycle. The BB also recognises a number of tokenised values (e.g. *Very-High, High*), which are used in *Foci* and *Policy* goals and interpreted by BB as signed values when generating ratings.

For the sake of efficiency, unlike BB1, Plethora's BB does not apply an *Integration-Rule* to the individual ratings of the KSAR generated by the *Foci* and *Polices.* BB1 maintained a list of separate ratings attached to each KSAR during conflict resolution, and allowed a user defined *Integration-Rule* to determine the most eligible KSAR based upon the contents of this list. The house-keeping, involved in maintaining this list and the interpretation of the *Integration-Rule,* was found to be time consuming. Since this occurred in each BB cycle, the added flexibility that it provided was considered to be out-weighed by the increase in the time taken to complete a BB cycle.

Basic KSs

The iterative problem solving process of the BB is controlled by the Basic-KS, which can be specified by the user, but unlike other KSs it does not appear on the blackboard. The default *Basic-KS* cyclically invokes the action of the chosen KSAR, checks for the termination conditions of any control decisions on the blackboard (including checking the time out of the current chosen action) and controls the conflict resolution; informing the rest of the system of each change of state.

The default *Basic-KS* (Note: this is the one used in all experimentation) implements a synchronous agenda based control. It instigates the *Chosen-Action* by returning the *Action* goal to the KS, and marks the *Chosen-Action's* status *Pending.* During this time any KS may read information on the BB and *Events* can be entered. If the *Chosen-Action* is internal, only its KS is permitted to change the BB contents. This continues until the *Status* of the *Chosen-Action* changes (e.g. to *Completed, Failed* etc) or the time-out is exceeded. If the time-out is exceeded the *Basic-KS* instructs the KS to terminate the action, marks it *Timed-out* and the cycle begins again.

Figure 3.12 summarises the differences between the control decisions and KSARs found in BB1 and Plethora.

Blackboard User Interface

Plethora's BB has a user interface which helps in debugging and monitoring progress (see figure 3.13). It enables the user to display, edit and save any item on the blackboard and via the use of the *Monitor* window, describe each operation it performs. The output of the *Monitor* can be used to form a log file of the BB activity. The contents of the BB can also be displayed in a pseudo-graphical manner, where each item of the blackboard is represented by its level, its event number and where the colour of an item indicates its status. In this case, the chosen KSARs status is *Completed,* and during its action it had entered an *Outcome, Problem* and three *Policies* on the BB. The yellow lines indicate the contents of the *Results-Of* field within the KSAR.

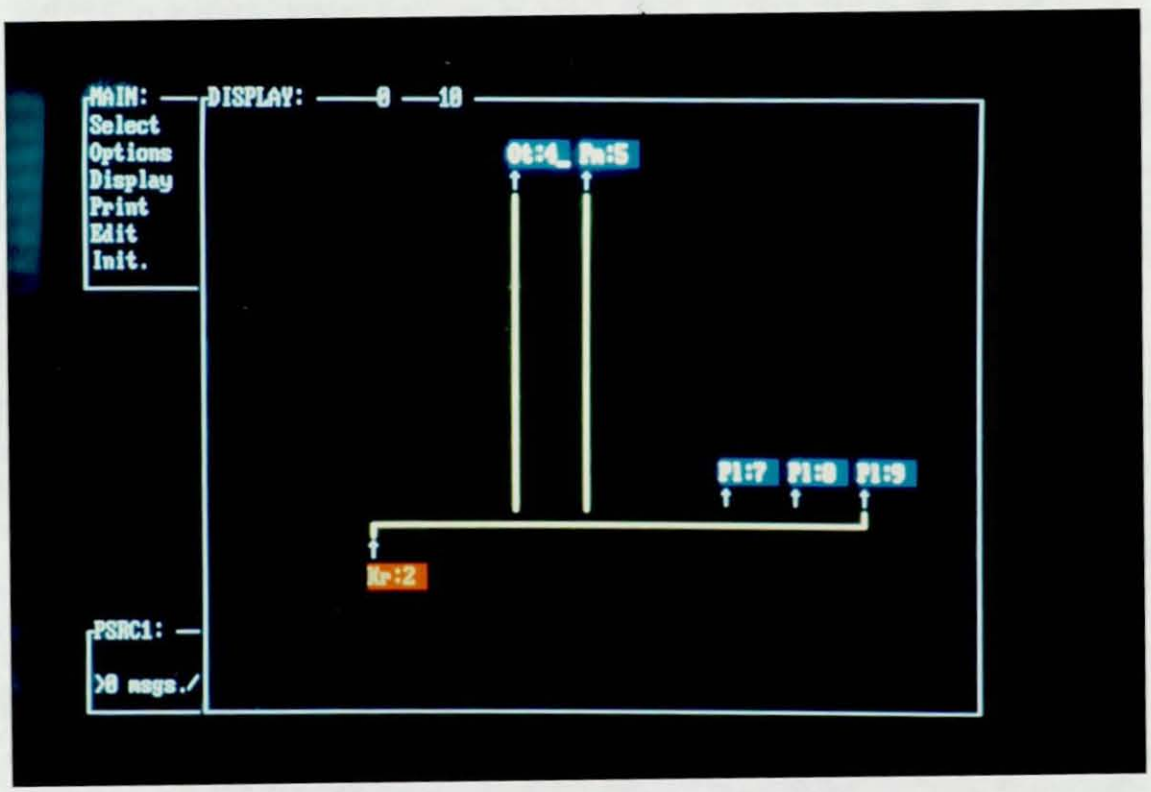


Figure 3.13 A photograph of the BB user interface.

Previous Blackboard Architectures

The use of BB architectures for work cell control is limited. A proposed architecture [24], which mixed Contract Net and Hearsay [4], highlighted some of the advantages found in Chapter 2 for the use of incremental opportunism in work cell control. However, the proposal contained no details of exactly how the problems of error recovery, critical times, blackboard structure etc were to be tackled. To the authors knowledge there has been no further published development of the ideas.

More recently, a multi-blackboard approach, Plato-Z [25], has been proposed and development begun upon the initial ideas. Plato-Z has a separate blackboard for each of the modes of operation found in Plethora's BB (i.e. Scheduling, Monitoring and Error handling), no mention is made of the structure of the blackboards or methods of operation. Plato-Z is still under development, and so there are no results to allow a comparison with Plethora. Plato-Z is written in Lisp and runs on a Symbolics 3645 Lisp machine. Before the date of publication of Plato-Z the author published an outline of the Plethora in [26] (see Appendix A).

Summary

Plethora's BB extends the behavioural goals of BB1 to include execution of the plan in an uncertain environment by a distributed system. It does so by the addition of the *Execution* plane, which allows the term "Action" in the original behavioural goals of BB1, now to include external actions. The *Event list* and *External-Chosen-Action* provide a means of reasoning about asynchronous events in terms of time, *Confidence* and goal directed behaviour, during the execution of an action. The use of the same conflict resolution scheme extends goal 8 of BB1, to include the relative priorities of execution actions in a uniform manner.

The use of independent agents to implement KS, rather than the procedural elements found in BB1, allows Trigger evaluation to take place in parallel reducing the cycle time of the BB.

References

[1] Hayes-Roth,B. Hayes-Roth,F.,
"A Cognitive Model of Planning", Cognitive Science, 3, p275-310, 79.

[2] Fox,B.R. Kempf,K.G.
"Opportunistic Scheduling For Robotic Assembly", IEEE Int. Conf. Robotics and Automation, p880-889, 85.

[3] Lesser,V.R. Erman,L.D.,
"A Retrospective View of Hearsay-II Architecture", 5th IJCAI, p790-800, 77.

[4] Balzer,R. Erman,L. London,P. Williams,C.
"Hearsay-III: A Domain-Independent Framework for Expert Systems", Proc. 1st National Conf. on AI, p108-118, 80.

[5] Erman,L.E London,P.E. Fickas,S.F.
"The Design and an Example Use of Hearsay-III", 7th IJCAI, p409-415, 81.

[6] Nii et al
"Signal to Symbol Transformation: HASP/SAIP case study", Artificial Intelligence Magazine, part 3, p23-35, 82.

[7] Hayes-Roth.B,
"A Blackboard Architecture For control", Artificial Intelligence Journal,no.26, p251-321, 85.

[8] Stefik,M.,
"Planning and Meta-planning (MOLGEN: part 2)", Artificial Intelligence 16, p141-169, 81.

[9] Hayes-Roth,F. Lesser,V.R.
"Focus of Attention in Hearsay-II Speech Understanding System", 5th IJCAI, p27-35, 77.

[10] Rich,E.
Artificial Intelligence, Mc Graw-hill,ISBN-0-07-052261-8, 83.

[11] Sacerdoti,E.D.
"The Nature of Non-Linear Plans",IJCAI 4, p5-15, 75.

[12] Fennell,R.D. Lesser,V.R.
"Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II", IEEE Trans. on Computers, vol.c-26, no.2, p99-111, 77.

[13] Tanenbaum,A.S
"Computer Networks", Prentice-Hall, ISBN 0-13-164699-0, 81.

[14] Coiffet,P.
"Interaction with The Environment", Kogan-Page, ISBN 1-85091-402-8, 83.

[15] Brooks,R.
"Visual Map making for a mobile robot", IEEE Conf. Robotics and Automation, p824-829, 85.

[16] Crowley,J.
"Navigation of an Intelligent Mobile Robot", IEEE Journal of Robotics and Automation, RA-1(1), p3-41, 85.

[17] Chatila,R. Laumond,J.P.
"Position Referencing and consistent World Modelling for mobile Robots", IEEE Conf. Robotic and Automation, p138-145, 85.

[18] Allen,P.A.
"Integrating Vision and Touch for Object Recognition Tasks", The International Journal of Robotics Research, vol.7, no.6, p15-34, 88.

[19] Durrant-Whyte,H.F.
"Sensor Models and Multi-Sensor Integration", The International Journal of Robotics Research, vol.7, no.6, p114-138, 88.

[20] Nash,J.F.
"The Bargaining Problem", Econometrica,50.

[21] Luo,R.C. Lin,M. Scherp,R.C.
"Dynamic Multi-Sensor Data Fusion System for Intelligent Robots", IEEE Journal of Robotics and Automation, vol.4, no.4, p386-396, 88.

[22] Brooks,R.A.
"Model Based Computer Based Vision", UMI Research Press, ISBN 0-8357-1526-4, 81.

[23] Harmon,S.Y. Bianchini,G.L. Pinz,B.E.
"Sensor Data Fusion Through a Distributed Blackboard", IEEE Conf. Robotics and Automation, p1449-1454, 87.

[24] Paul,R.P Durrant-Whyte,H.F. Mintz,M
"A Robust, Distributed Sensor and Actuation Robot Control System",6th ICAR, p93-100, 85.

[25] O'Grady.P. Lee,K.H.
"A Hybrid Actor and Blackboard Approach to Manufacturing Cell Control", Journal of Intelligent and Robotic Systems, no.3, p67-7, 90.

[26] Sillitoe,I.P.W.
"Towards Knowledge Based Control of a Flexible Work Cell", IEE Colloquium on Knowledge Based Environments for Industrial Applications, Savoy Place, London, p1-4, 89.

Chapter 4
Communication

The work cell is an inherently distributed domain. This chapter describes the design and implementation of a distributed communication mechanism based upon message passing. It's design provides a unified means of communication and coordination between all elements of Plethora.

Message Passing

The traditional view of software systems is that they are composed of data and procedures which describe how to manipulate the data. However, procedures make implicit assumptions about the form of the data on which they operate. In traditional systems, this natural functional inter-relationship between a particular set of data and its procedures is lost, and the binding of these two elements is left to the programmer.

In a distributed message passing system, where in general the procedural element would be a parallel process, these elements are combined to form an agent and changes to the agent are made by sending it a message. A message embodies the semantic content of the manipulation. The encapsulation of data and procedures allows changes to be made within the agent, while the external action of the agent remains unchanged (see [1] for a more complete description of message passing and object orientated systems).

Why use the message passing paradigm ?

Message passing has four major advantages for robotic systems,

An intelligent robotic system will be large and complex. The message passing approach makes design, testing and maintenance a modular and so a more manageable task [2].

Robotic systems are inherently distributed. Many techniques will need to be run in parallel in order to obtain the required response. Message passing provides a means of distributing such a system while maintaining a formal framework.

Message passing can be used to explicitly synchronise tasks, be they physical or computational. Hence, the same approach can be used to synchronise co-operating KS's or physical resources.

Message passing can be used to encapsulate all aspects of a device within an agent and provides a well defined interface between it and the rest of the system and a degree of portability.

Issues in Message passing systems

Four design issues dominate the form of a message system [3]

Are the primitives, which manipulate the messages, to automatically block processes?
(i.e. force the agent to wait for the next communication event)

What addressing mechanism is to be used to specify the agents involved in communication?.

What is the format of a message?.

What is the approach to exception handling, such as communication failure?

A brief discussion follows of the different solutions available to the problems above.

Blocking or Non-Blocking

Blocking primitives provide an elegant means of synchronising agents. If a SEND blocks until the receiver is ready to receive, and the receiving agent also blocks until the message is available (i.e. rendezvous [4]), then no other mechanism of synchronisation is required. This is simple to visualise and there is no need to buffer messages.

However, a process blocked in sending cannot do useful work and when the communication concerns a number of agents this approach can inhibit the inherent parallelism that the technique was to promote. This is particularly so in large grained systems where such delays will be greater, and in real-time environments where unlimited blocking of an agent might be disastrous.
Other options include,

> A queued send, which queues messages for the receiving process, but does not block. It is used with a blocking receive and allows the sending process to run arbitrarily ahead of the receiving process. However, it raises the possibility of unbounded message queues and deadlock.
>
> Conditional primitives, the send only transmits if the receiving process is blocked and waiting for the message, otherwise it returns failure. Likewise the receiving agent receives the message only if it is already waiting. This can lead to excess polling by the transmitting agent and possible bandwidth limitations in a common channel.
>
> The use of the reply [3] primitive. In this arrangement the transmitting agent remains blocked until it is sent a special message (i.e.its *Reply*) to unblock it. The reply primitive is non-blocking. The use of a blocking send and receive and non-blocking reply removes the need for other non-blocking primitives and additional synchronisation schemes.

Addressing Mechanism

The communicating agents could specify the message destination implicitly, through an input/output declaration or explicitly within the body of the message. However, if the communicating agents are specified explicitly it is possible to allow agents to accept a message from any sender. This has three major features useful to Plethora,

Non-deterministic reception of messages from different sources.

An agent can then be used as a shared data base
(i.e. geometric information about the workpieces).

It leads to a straightforward implementation of a broadcast mechanism.

Message Format

Two separate questions arise within this issue. Firstly, should the message size be limited in some manner. Secondly, whether the user section of the message be defined completely by the user or selected from a set of system defined possibilities.

In practice, messages within the work cell and between modules, tend to be short and have a fixed maximum length. Messages with a maximum fixed length simplify and quicken message storage allocation. They also help in the prediction of the critical time for response of an agent to a transaction over the network. This is an important factor in the detection of exceptions and communication failures.

System defined message formats and data types provide a formal framework for the basis of communication systems. However, they tend to be verbose and inflexible in complex systems, where there are a large number of agents acting in a non-deterministic manner [3]. Hence, while system defined messages may provide a possible solution to the real-time aspects of the problem, they do not provide an ideal solution to the communication between the knowledge based agents.

Communication Failure

Causes of communication failure can be categorised as below

Deadlock [4] between communicating agents.

Untrustworthy correspondents, a process might violate the protocol in progress and block the transmitting process forever.

Physical communication failure.

Time-outs are a common form of solution for such potential failures. These techniques raise an exception, if after a specified time, the next communication event has not occurred. Two typical techniques are outlined below

An assassin process is started at the beginning of a transaction, but its action is delayed for the time-out interval. If the transaction is completed within the interval the sending agent destroys the assassin otherwise the assassin destroys the sending agent.

Administrator [3] form, the transmitting agent is blocked on a "receive-any-message" primitive, awaiting an input. A time manager sends it a message indicating that the last time interval has expired. The transmitting agent either decides to handle an exception or send a new time interval to the time agent.

Both techniques rely upon a third agent, this is a disadvantage in a real-time heterogeneous network, where many of the simple control processors cannot afford the extra overhead that is required to support the third agent.

Requirements

The requirements of the communication system can be obtained by analysing the data flow during a typical blackboard cycle,

Remote invocation of the action part of a KS.

Asynchronous reception of data (i.e. an agent reporting the status of an event).

Access to globally shared sources of information (e.g. geometric modeller).

A means of ensuring coherent responses from globally shared sources of information, given a number of competing requests.

Synchronisation of two or more agents.

Detect and process timing exceptions.

Terminating an action prematurely, in response to an error condition.

Broadcasting blackboard changes.

Previous Message Passing Systems

No distributed message passing system used in robotics (to the author's knowledge) has tried to satisfy real-time and knowledge-based requirements within a single system. The following describes two groups of related work, which employ the message passing paradigm to solve one half of these requirements. The first illustrates the concepts within systems which were designed to support general distributed programming. These are applicable to the knowledge-based requirements of Plethora. The second group describes systems that have been primarily designed to control the physical processes within the work cell. These have a direct bearing on the use of message passing for the coordination of real-time processes.

General Distributed Programming

PLITS [5] was a language which described a methodology for general purpose distributed programming based upon message passing. It introduced the notions of the MODULE and TRANSACTION KEY (known in other systems as a transaction identifier (TID)). The ideas used in PLITS formed the basis for many other message passing systems (e.g. [6],[7]).

In PLITS, a kernel maintains a single message queue for each module, which serves as the only means of communication with the module. A message (or A-set) comprises a set of name-value pairs called SLOTS. Unique transaction keys are included in the message to identify a particular message. These are used to solve the problems of selective reception, message validity and the implementation of multiplexed servers.

The message passing primitives consist of a blocking send and receive, each of which has an associated local time-out period (the actual length of this time-out period seems to be a default value). The preservation of the send-transaction-receive sequence can be used to guarantee that the messages arrive in the same order as they were transmitted by the sender.

The ability for a module to handle messages with unknown slots was said to allow several modules to take part in task sharing, while maintaining the integrity of the individual module structure.

Message Passing Applied to Industrial Robotics

IMRS

IMRS [8] is a proposed set of application level primitives used to support a real-time distributed architecture. The problem, and hence control, is assumed to consist of the indivisible sub-processes, forming a hierarchical n-ary tree module structure. They foresee that it will use task-sharing as a means to synchronise the processes within the work cell, and as a means to resolve the interaction between the computational processes. The interaction between computational processes is categorised as follows,

Independent processes

Where 2 robots exist on the same shop floor, but their work is independent.

Loosely coupled processes

Where the work of the robots is independent, but where the individual actions are not (e.g. tool sharing, collision avoidance).

Tightly coupled processes

Where 2 robots co-operate to perform the same task "Picking up a steel beam from a conveyor".

Serialised motion processes

Where the operations of one robot must be finished before the next can begin its work.

Communication between modules is further categorised into vertical and horizontal communication. Vertical communication is between a module and its sibling processes, horizontal communication is between the children of a common parent. The basis for all inter-task communication is via *ports*. This logical structure is used to limit the communication primitives to the proposed set. A time-out exception can be raised if the communication is not completed by a designated time, on either the one-way message or two-way rendezvous.

They concluded that remote procedure calls, message passing with a variety of non-blocking, blocking and interrupt forms, and signal/wait operations would be necessary to provide for the 4 types of interaction.

MAP message specification

Message passing appears at the application layer of the ISO standard. The MAP [9] application layer is to have 2 sets of facilities, file transfer (FTAM) or the manufacturing message communication (MMFS).

GEC [10] have found problems when implementing systems using MMFS.

"There has been considerable pressure from the user for simplified, rapid communications within an island of automation or process cell" [10]

Such in-cell data is often restricted to small amounts and needs to be transferred rapidly. This is better suited to a connectionless protocol, which does not contain the large overheads associated with virtual connections when transferring small amounts of data. This has led to the development of in-cell networks (known as mini-map/micro-map), while the full blown MAP is used for communication between islands of automation.

A contender for the micro-MAP is ERA [11] which is based upon MIL-STD-1553b, and is used for communication between sensor/actuator systems. This is a much simplified network, which was originally used to communicate between embedded computer systems in aircraft. Once again, it defines a message passing approach, but it is of a form which is more applicable to real-time control. Messages are of a fixed size, and there is a minimum acceptable delay between transmission and the reception of the answer for a given message. Allowable extensions within the proposed standard include, broadcasting a message and direct internode communication.

Summary

Each of the approaches have advantageous attributes, but fails to provide an overall solution to Plethora's communication requirements. The module and language structure of PLITS could satisfy communication between knowledge based agents, but there is no means of making their behaviour dependent on the time constraints of the system. Similarly, MMFS's rigid internode language is suited to simple data exchange but less appropriate for a task or results sharing.

MMFS and IMRS make a-priori assumptions about the structure of communication during the task, which in turn is predicated upon limited uncertainty within the cell. Whereas it has been argued that for the control of a truly flexible work cell, the structure of communications needs to be determined by its current state, so that it may adapt to unpredicted and unmodelled events which will undoubtedly occur.

Plethora's Message Passing System

Players, Prompters, Companies and Groups

Plethora is a distributed collection of autonomous Actor [12] like agents made up of *Players* and *Prompters* which communicate through a single mechanism, that of buffered message passing.

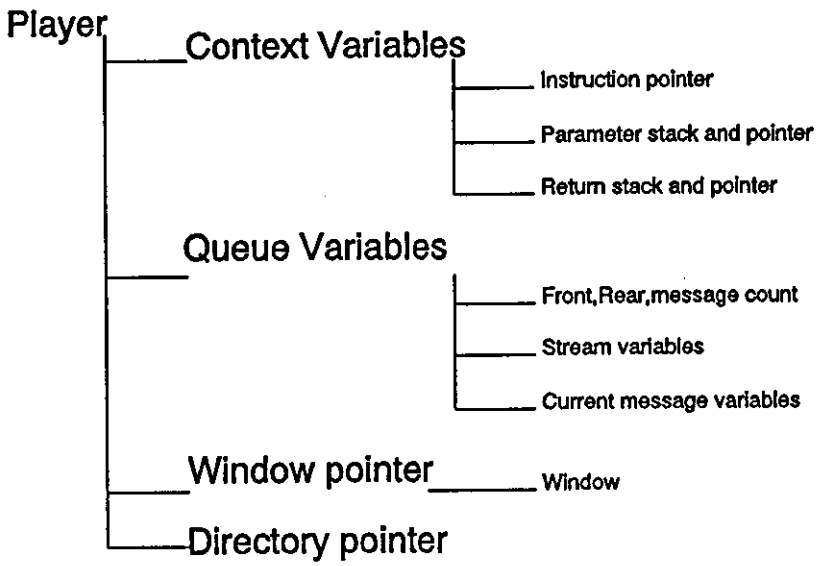


Figure 4.0 Groups of fields within a Player.

Players are large data driven agents which are scheduled according to the attributes of the messages in their message queues. Hence, the *Player* which the scheduler regards as having the highest priority message is scheduled first. *Players* are used to form the knowledge sources within Plethora and have *Entrance*, *Part* and *Abort* fields. The *Entrance* contains the initialisation code and is run each time the *Player* is restarted. The *Part* constitutes the nominal activity, and the *Abort* that which needs to be undertaken to safely bring the *Part* to a conclusion when a fatal fault occurs. *Players* are independent agents and hence have their own context. Figure 4.0 shows the groups of fields within the context of a *Player*. The context consists of 300

bytes of return stack, 500 bytes of parameter stack and 80 bytes for the remaining variables.

Prompters are small, simple, event-driven agents which synchronously sink and source information in and out of Plethora with the occurrence of physical events. Since, they are event-driven and deal with the time critical operations of the Plethora, a *Prompter* will interrupt the action of a *Player* whenever its associated event occurs. More than one *Prompter* may be attached to an *Event* (see figure 4.1), such as a timer, and events can have differing priorities. An *Event handler* (one per processor) controls the activation of *Events* and their *Prompters,* and so hides the interrupt

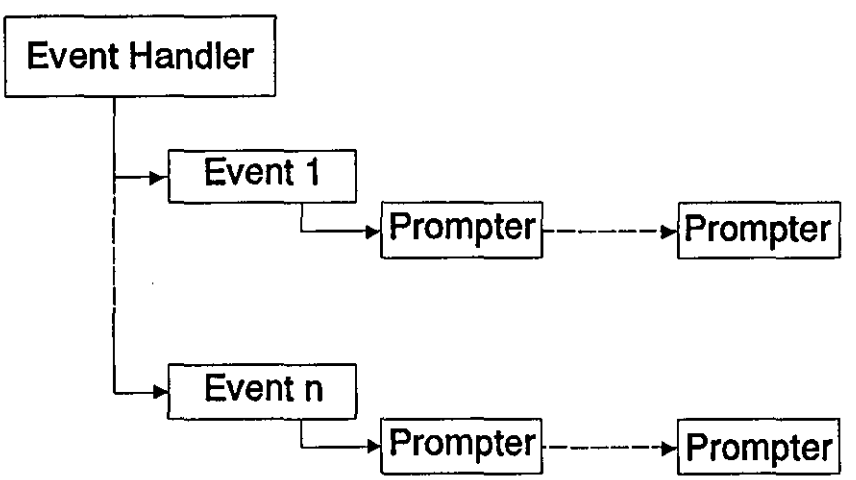


Figure 4.1 Event handler, Event and Prompters.

details of a particular processor from the message passing system. The structure of a *Prompter* is similar to that of a *Player*, except that it does not have its own stacks but uses those of the *Event,* neither do they have queue variables but have instead *State vectors* (this is explained in greater detail in the section concerned with message primitives). Disregarding the memory space required for it's code, each *Prompter*

occupies 32 bytes and each *Event,* disregarding its stacks, 16 bytes. The depth of the *Event* stacks are chosen by the user. However, because of the simplicity of a *Prompter's* action these can be very much smaller than those necessary for a *Player* (typically 32 bytes each). *Prompters* form the real-time interface to the devices in the cell.

Each agent can make *Assumptions* about the existence of other groups or agents. These *Assumptions* allow the agent to specify messages to the assumed agent or group without prior knowledge of it's address.

Players and *Prompters* can be formed into *Companies. Companies* are functionally interdependent groupings of agents which collectively perform a single activity. Other agents in the system interact with a *Company* as if it were a single entity.

Groups of agents allow the linking of independent agents with similar interests or functions (e.g. the group of all the players which can solve a particular type goal). An agent can belong to a number of groups.

Plethora extends the concept of encapsulation, from that of program and data structure, to include the physical attributes of a mechanism and its function. Each *Player* representing a device, not only encapsulates all the computational attributes, but it also includes it's geometrical description (in terms of geometric primitives found in the geometrical modeller). Whenever Plethora is restarted the agents are interrogated and their attributes included in any shared data bases. In this way, as long as a device conforms to the message passing protocols, it can be added or removed without intervention by a programmer. (fulfilling requirement *ct4)*

Message Passing Primitives

As a result of the diversity of the requirements, it was decided that the choice of message passing primitives and message format should not be made so as to enforce a single solution (i.e. such as the rendezvous). Rather, that they should be chosen in order to produce a flexible basis for the implementation of structures most appropriate for a given requirement. With this in mind, protocols based upon a combination of primitives which reflected the data flow within Plethora were developed, in this case a blocking *Send* and a number of blocking and non-blocking *Receives*. The agent's received messages are buffered and the *Receive* primitives selectively remove messages from the agents queue, based upon keys contained within the message.

The message record is partitioned into two groups of fields, the message itself, which is user defined, and the header (see figure 4.2). The header contains all of the

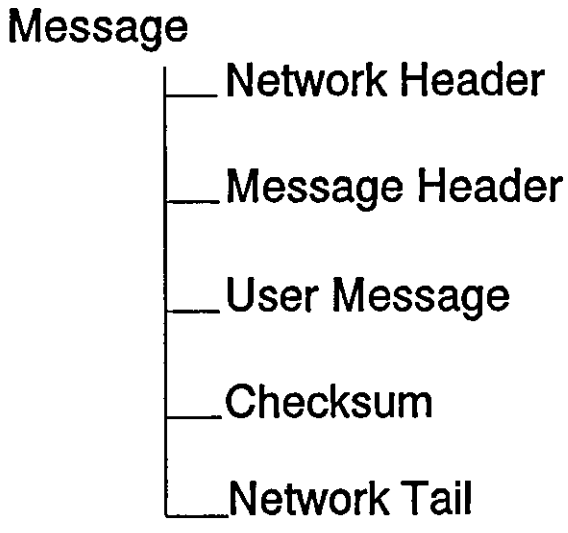


Figure 4.2 Message Structure.

system information for the selective removal of the message from the queue and

consists of the fields shown in figure 4.3. What follows is a description of the use of the header fields and resulting protocols. A discussion of the user portion of the message is delayed until Chapter 5.

Header Format

The header contains all the information required to send or broadcast a message, perform a form of time-out, provide various levels of acknowledgement and

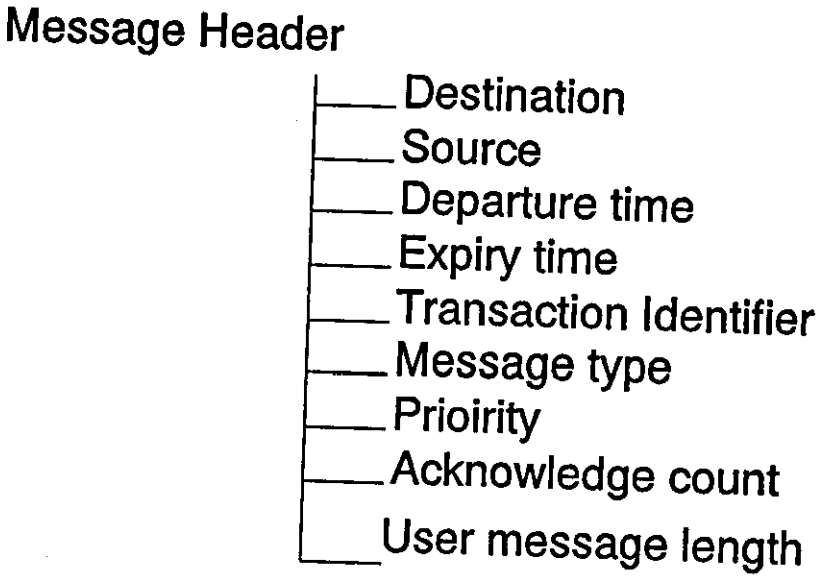


Figure 4.3 Header Structure.

implement transaction protocols.

Message Addresses

The source and destination addresses are formed in two parts, the processor identifier and the agent identifier. Together they form a unique address for an agent within Plethora. A broadcast to a group is indicated by a processor identifier of zero

followed by the group identifier.

The host processor translates from the system address to the local address with the use of a data structure known as the *Agent-Directory*. This is a table which contains the local addresses and forms of the resident agents. Entries for a Group are similar to those for an agent, except that they also contain a counted list pointing to the members of the group which reside on the particular processor. Groups form the basis of the interested groups of KSs described in Chapter 3. The address [0,0] is undefined and used by the *System-Kernel* when verifying the operation of the network.

Message Types

All messages within Plethora have types, these are used to simplify the interpretation of messages and ensure consistency within the various protocols. The message types used within Plethora are described below.

Requests and Reply

The *Request* type is similar to remote procedure calls [13], in that it allows an agent to access remote data structures and knowledge to which it is not privy. However, it differs from a remote procedure call in that it a part of an object orientated system and so manipulates messages.

The *Reply* type is used to return the result of a *Request*. A *Request* transaction continues until a *Termination* message is sent or the end of the request is indicated in the user portion of the message. Chapter 5 expands upon *Request-Reply* transaction and the representation and management of objects within Plethora.

State Changes

The *State Change* message type differs from the *Request* in that it makes changes to the state of the agent, rather than changes within the scope of a local transaction. Changes within the scope of a local transaction are not permanent and so disappear at the end of the transaction, such as the request for sensory data. While state changes remain after the conclusion of the transaction and so affect the further

operation of Plethora. *State Changes* include messages to actuators which will change the physical state of the cell, and messages which alter the contents of shared knowledge sources (e.g. the geometric modeller). Before a *State Change* message is accepted extra checks must be made based upon its TID and the source of the message. For example, whether the agent "sending" controls the chosen action on the blackboard and so has the right to make state changes.

Trigger, Precondition, Action and Event

These describe the task sharing activities within Plethora (as described in Chapter 3). *Trigger* messages inform the system of occurrences validated and entered on the blackboard, and are used by the KS to update the local trigger state. *Action* and *Precondition* hold the context of a particular action volunteered by the KS to the BB in a KSAR.

Acknowledgement, Error and Abort

These form the responses from the recipient to the sending agent indicating the validity of the contents of the received message. An *Acknowledgement, Error* or *Abort* message is made in response to every *State Change, Action* and *Terminate* message, depending on its validity. An *Error* differs from an *Abort* message in the severity of the fault that gave rise to it. An *Error* would occur if the syntax of a message is violated (since this is recoverable), while a robot control agent would report an *Abort* if it were asked to move its manipulator to a point which was outside its reachable work space.

Priority

The priority of the messages in a *Player's* queue, determines the priority assigned to it by the local player scheduler and hence, when the message is to be serviced. Messages with high priorities are sent over the network and attached to the destination agent's queue, in preference to lower priority messages. This ensures that high priority messages are serviced first wherever their destination in Plethora. *Event* messages usually have high priorities, since they indicate what is actually happening within the cell and so may require immediate action. The value of priority is assigned by the sending agent and so can be a function of it's state.

Departure and Expiry Times

Time-outs can be enacted using the expiry time in the header of a message in conjunction with the group of *Receive* primitives. The expiry time interval corresponds to the time interval after which a response to the message will no longer be appropriate. The receiving agent can check if a time-out has occurred, if the current time is greater than the messages's departure time plus expiry time. The transmitting agent uses the same interval as the period required in the *Receive's* time-out, allowing it to independently detect a time-out and take the appropriate action. This mechanism removes out-dated messages from the system and provides a means to graceful recovery after a communication error.

Acknowledgements

The acknowledgement field within the header is an integer and can be used to indicate the number of agents which have responded to the message. This is particularly important when an agent is performing certain types of broadcast, since this field returns the actual number of replies the agent is to expect.

Transactions

The transaction identifier is an integer generated by the initiating agent and is used as the basis of all transaction protocols within the system. A transaction comprises of an atomic interaction by the agent with the rest of Plethora, and can include communication with a number of agents (e.g. initiating of a number of parallel actions).

The TID can be used in a number of ways. For example, consider an agent which provides a service to the rest of Plethora (i.e. the geometric modeller) and contains globally accessible information. At any particular time, it will have a number of pending *Requests* or *State Changes*. In order for it to return coherent responses to each *Request*, we need to ensure the "serializabilty" of each transaction (see [14] for a complete discussion of the point). The simplest means of achieving this is to complete the transaction under way before beginning the next. This can be accomplished by limiting the response of the service agent to only those messages which contain the current TID and agents source address, delaying the other requests. More extensive protocols to control common resources are described in Chapter 5.

Primitives and Protocols

Sending messages

```
Co-Send <Message Type>
            <User Message 1> |
            <User Message 2> |
            ..............
            <User Message n>
End-Co-Send
If
      <If all messages were successful>
Else
      <If a message failed>
EndIf
```

Figure 4.4 Co-Send Structure.

At the completion of any *Send* there is a context switch performed by the local player scheduler. For groups of related messages a *Co-Send* structure is available (see figure 4.4) in which there is no context switch between sends and each message sent has the same TID. The structure then waits for the reception of all the responses and returns either a false flag indicating a time-out or a true flag and a list of the responses. At present all messages must be of the same type.

The *Co-Send* can be used to treat a number of related parallel operations as a single operation, synchronise the player to the completion of that operation and take coordinated action if it fails.

Receiving Messages

The receive primitives can be categorised into three groups, according to the effects they have upon the execution of the agent. The *Wait* group halts the operation of the agent (effectively removing it from the scheduling process), until it receives a message with the appropriate header field. The *Polling* group do not cause a context switch, but poll the message queue for an appropriate message once, and then return a flag and the message, if successful. And finally the *Receives*, which employ time-

Primitives	Attribute		
	TID	Source	Time
Cwait\creceive\cpoll			
Twait\Treceive\tpoll	X		
Stwait\streceive\stpoll	X	X	X

Figure 4.5 Message Reception Primitives

outs and cause a context switch if the appropriate message is not found in the message queue.

Whenever there is a choice of acceptable messages in the queue, the message with the highest priority is selected; or if the acceptable messages are of equal priority, the message with the shortest deadline to expiry is removed. The three groups are summarised in figure 4.5.

The *Wait* group make efficient use of the available processor time by forcing the agent to act in a message-driven manner and reduce the scheduling overhead.

The *Polling* group allow a real-time or event-driven procedure to continue under the control of an external agent. Thus, the procedure once started can be terminated or its action modified, in the light of changes that occur during the time of the procedure.

The *Receive* group implement a form of simple time-out which is only dependent upon those agents concerned with the communication, and allows each agent to take

independent action to correct the fault. This maintains a low overhead, in terms of the extra agents which would be required for other forms of time-out, and becomes important when mapping agents on to a simple processor. Such processors are often used for the control of real-time processes and require the use of time-outs, but cannot support the extra burden of a more sophisticated arrangement.

Streams and Pipelines

Pipelining is a method by which data is processed in a series of stages and where each stage is implemented as a parallel process. As the processed data is passed on to the next process in the pipeline, the current process begins to operate on the data passed to it. The flow of data is known as the data Stream [4].

Many fundamental problems in robotics can be formulated in this way (i.e. the process of generating feature vectors from raw sensor data) and so pipelining provides a natural means of introducing parallelism into Plethora.

Plethora provides the following message passing primitives to support pipelining.

Begin-Stream, End-Stream
These initiate and terminate a pipeline.

Generate-Stream
Generate a stream of replies to an Initial *Begin-stream* request by application of the given function. The next item in the stream is sent once the acknowledgement has been received from the agent up-stream. The stream can also be terminated by the agent up-stream using a *TRM* message.

Map-Stream
This maps the incoming replies through a given function.

Filter-Stream
This applies a predicate to the stream.

Merge-Streams
Takes two sources and maps the pairs of messages through a function to produce a new Stream.

(Chapter 7 illustrates the use of a Stream within a Company when processing vision data)

Player Scheduler

The player scheduler is a *Prompter* attached to a timer. Context switching between players occurs when the timer expires or under the direction of one of the message passing primitives. The scheduler supports two lists of players those with messages with and those without zero priorities. When a switch takes place the agent with the highest priority message is started. If two messages have the same priority the agent is chosen which has the message with the shortest expiry time. Otherwise, an agent is chosen from the zero priority list using the shortest expiry criteria. At any time during this process *Prompters* may interrupt the *Players*, eventually returning control to the scheduler.

Message passing and Prompters

Prompters are designed to be small and fast and so cannot afford the overhead required of sophisticated message interpretation. Thus, the action of a *Prompter* to incoming messages is different to that exhibited by the *Player*. To simplify the operation of message passing the *Prompter* does not contain code to explicitly process incoming messages, but relies on the message passing system to recognise that the destination of the message is a *Prompter* and instead of attaching the message to a message queue, loads the contents of the message into the appropriate *Prompter vector*.

The *Prompter* maintains three state vectors, its *Command* state, *Internal* state and *External* state. The command state is loaded with the user portion of a valid *State-Change* message, while the *Internal* state is copied to form the *Reply* to any *Request*. The *External* state is the sink or source of information which it controls and is periodically updated by the *Prompter*. The most usual role of a *Prompter* is to form a simple state machine.

Network Interface

The message passing system is a connectionless architecture where the message passing primitives provide functions at the Transport layer in the ISO Reference model. The Network and Data link layers (below this) are implemented by three *Players* and two *Prompters*, copies of which reside on all processors which contain *Players*. They are,

Network Layer

Net> and >Net
These and the *Error-Handler* form the Network layer of the network and so provide the network interface seen by the message passing primitives. Out-going messages are attached, by the *Send* primitive, to *>Net's* message queue.

Error handler
This logs and tries to repair system level errors. (i.e. one of its functions is to create and send an Error message to the instigator of a message which has used an unknown address).

Data Link Layer

(>Net) and (>Net)
These are block oriented network device drivers created from two *Prompters*. Together they form the Data Link layer.

A further single agent known as the *System-Kernel* completes Plethora's interface to the network. *System-Kernel* performs all management functions for the communication system within Plethora and it also provides a user interface, through which remote agents can be controlled via their local *Kernels*. During the initialisation stage of Plethora it also has the job of resolving the *Assumptions*. It does this by requesting from each of the local *Kernel* agents a list of their residents, addresses and *Assumptions*. Once the list is complete, it returns the necessary addresses for each of the unfulfilled *Assumptions*.

Network

The experimental network of processors consisted of a number of IBM AT clones and 65F12 microcontroller boards. The physical layer was formed by connection of their RS232 serial lines, using a conventional asynchronous byte orientated protocol (at 110K baud and 62.5K baud respectively). An odd parity bit was contained with each byte, and a vertical parity check-sum byte included after the user defined message (see figure 4.6).Two topologies were used, a star and a token ring [15]. The star had a central message switch created from another IBM AT which

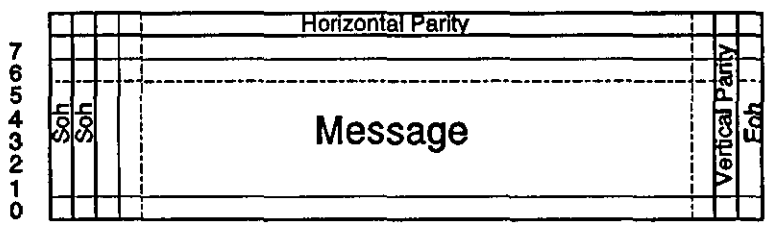


Figure 4.6 Network Message structure.

contained a message routing agent called *Aether*. *Aether* routes messages and provides a limited debugging and monitoring facility. The ring allowed more processors to be included into the network, but gave a slower response. Both topologies enforced deterministic access to the network using a naive protocol. Each processor could only receive and pass on messages around the ring until it had access to the network. In the case of the ring, this was indicated by the arrival of the token, and by a change in state of a status line in the case of the star. The processor had access to the network for one message transmission before it passed on control. Messages were acknowledged as they passed through by incrementing the *Acknowledge-Count*, by the number of agents in each processor which corresponded to the destination address (i.e. piggyback fashion [15]). Failure to receive the return message caused a re-

transmission, a further failure caused the *Error-Handler* to return an *Abort* message to the originator of the message.

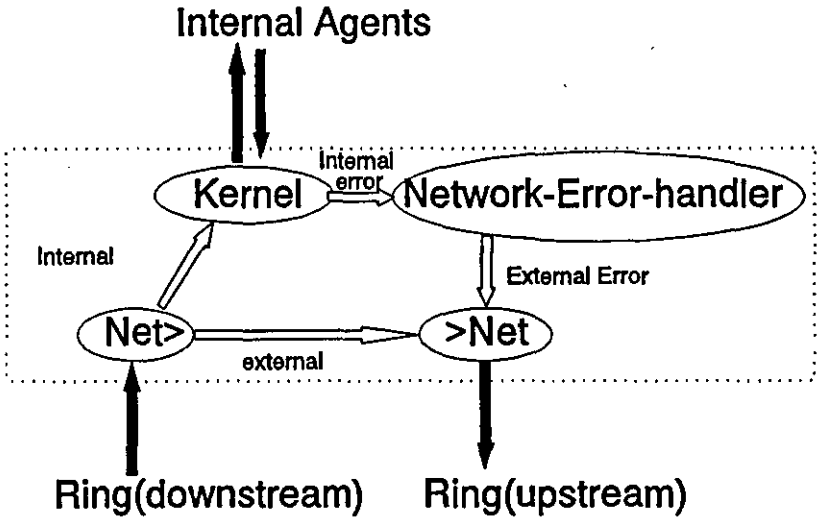


Figure 4.7 Network System Agents.

Summary

The structure of the message passing system fulfils a number of the requirements in Chapter 1 by providing,

> a uniform mechanism for control and communication for both knowledge and real-time agents. *(cm2)*
>
> in the *Assumption* mechanism and extension of the concept of encapsulation, a means by which devices can be added or removed without affecting other agents within the Plethora. *(ct4)*
>
> both data and event-driven agents of an appropriate form (i.e size, speed and method of activation) for a real-time heterogeneous network. *(cm2)*
>
> a mechanism by which the data driven agents are scheduled according to real-time constraints (i.e priority and expiry time). *(cm2)*

References

[1] Ramamoorthy,C.V. Sheu,P.S.
"Object-Oriented Systems", IEEE Expert Systems, p9-15, Fall 88.

[2] Sommerville,I.
"Software Engineering", Addison-Wesley, ISBN 0-201-14229-5, 85.

[3] Gentleman,M.W
"Message Passing Between Sequential Processes: The Reply Primitive and Administrator Concept", Software-Practice and Experience, vol.11, p435-466, 81.

[4] Ben-Ari,M.
"Principles of Concurrent Programming", Prentice/Hall International, ISBN-013-701078-8, 82.

[5] Feldman,A.J.
"High Level Programming For Distributed Computing", Comm. ACM, vol.22, no.6, June 79.

[6] Cheriton,D.R
"The V Kernel: A Software Base For Distributed Systems", IEEE Software, p19-42, April 84

[7] Boggs,P. Shoch,J.F. Taft,E.A. Metcalfe,E.
"PUP: An Internetwork Architecture",IEEE Trans. Comm., vol.28, p624-631, April 80.

[8] Shin.K.G. Epstein.M.E
"Intertask Communications in an Integrated Multirobot System", IEEE Journal of Robotics and Automation, vol.RA-3, no.2, April 87

[9] "Industrial Automation Systems - Systems Integration and Communications: Manufacturing Message Specification- part 2. - Protocol Specification", 2nd Draft Proposal, 1987-5-21, ISO/TC 184sc5.

[10] Platt.E.
"MAP and GEC", GEC Review, vol.3, no.2, p6-11, 87.

[11] Burton,P.
"A Proposed Industrial Field Bus For MAP Networks Using MIL-STD-1553B", ERA Technology, Cleeve Rd, Leatherhead, Surrey, England, 84.

[12] Hewitt,C.
"Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence, No.8, p323-364, 77.

[13] Birrell,A.D. Nelson,B.J.
"Implementation of Remote Procedure Calls", ACM Trans. on Computer Systems, vol.2, no.1, feb. 84.

[14] Bernstien,P.A. Shipman,D.W. Wong,W.S.,
"Formal Aspects of Serializability in Database Concurrency Control", IEEE Trans. Software Engineering, vol.SE-5, p203-216, May 79.

[15] Tanenbaum,A.S
"Computer Networks", Prentice-Hall, ISBN 0-13-164699-0, 81.

Chapter 5
Objects, Rules and Inter-agent language.

The previous chapter dealt with the communication mechanism and representation of the agents within Plethora. This chapter extends the description of Plethora to include the related questions of representation within agents, and the language of discourse between the agents.

Objects

The message passing system described in the previous chapter can be classified (using the classification given in [1]) as *object based*. This means the agents form encapsulated descriptions and communicate through messages, but there is no class structure or any form of inheritance between the agents. The concept of *class*, categorizes related objects together so that each member of the class automatically inherits the methods or procedures of that class (a discussion of inheritance, its various forms and uses is found in [2]).

This has a number of advantages. Firstly, it allows much of the implementation detail to be hidden from the user of the object, in turn making creation and manipulation of objects simple. Secondly, such a system has the ability for two different objects to react differently to the same message. This leads to a compact form of language when specifying object manipulations (see [3] for a more detailed illustration of this point).

Definition of an Object

Objects can be modelled as automata whose state represents the object's and whose input symbols represent operations with their accompanying arguments. Figure 5.0 illustrates an object with operations $\{f_1.....f_n\}$. An operation f_i with arguments x in state s, results in output $f_i(x,s)$ and state transition $s'=g_i(x,s)$. Thus, the operation symbols f_i are associated with symbols on the input tape, internal actions are associated with the state transition function $g_i(x,s)$ and outputs are associated with operations $f_i(x,s)$.

Objects can be described with a variant of the *let* notation

$$
\begin{aligned}
&\textit{let } x_1=a_1,\ x_2=a_2 \ldots\ldots \textit{ in}\\
&\qquad f_1(p_1) = \textit{body of } f_1\\
&\qquad f_2(p_2) = \textit{body of } f_2\\
&\qquad \ldots\ldots\ldots\ldots\ldots\\
&\textit{endlet}
\end{aligned}
$$

where the variables $\{x_1 \ldots . x_n\}$ correspond to the instance variables, and $\{p_1 \ldots . . p_n\}$

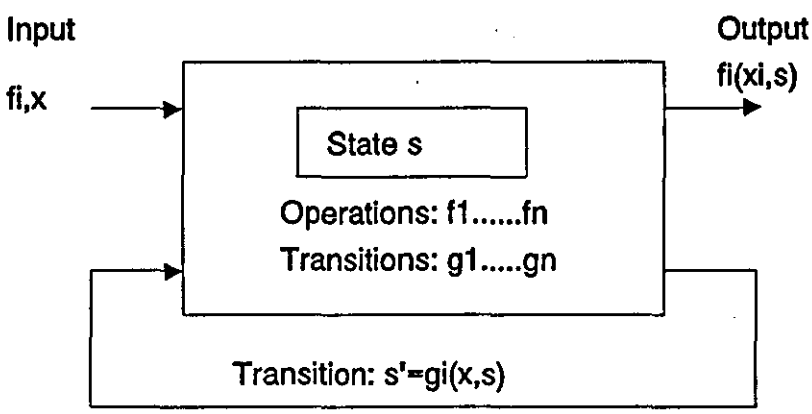


Figure 5.0 Diagrammatic representation of an Object.

correspond to parameters of the messages $\{f_1 \ldots . . f_n\}$. The semantics of the *let* clause differ from the *let* notation in functional programming [4] in that an operation can modify the instance variable.

A system which supports class and inheritance is categorised as *object-orientated*. Plethora is object orientated within its agents and these objects are slot-based (i.e. defined in terms of the contents of their instance variables) hence, agents communicate in a form of object orientated programming language. However, unlike conventional object orientated systems which are sequential, the objects within Plethora are distributed and their management is necessarily more sophisticated. The approach taken here was first described in a paper by the author [5] (a copy of which is found in Appendix B) and called ROOP, remote object orientated programming.

ROOP

The following section describes ROOP and its application to the vision server KS.

Vision Server

The camera provides the necessary sensory data for a large number of KSs which can be used to identify objects and verify actions. During the problem solving cycle, a number of these KSs make concurrent access to the server in order to evaluate heuristics. These will be used to select the most suitable technique for the current state of the plan. Therefore the design of the vision server must not only be able to cope with the problems of large data sets (Image sizes range from 64-256 kbytes) and those problems associated with the particular domain, but it must also maintain coherent responses under the influence of competing requests (see [6] for a full discussion of such problems) while minimising the housekeeping required of a KS to obtain a service.

Representation

The elements of a transaction are represented as polymorphic objects within the camera KS, that is the *Class* of a particular object may change during its lifetime. The polymorphic nature of the objects is necessary because of the size of the data structure involved and the limited memory available. All objects are held and managed within the server (see figure 5.1 for an illustration of the fields found within an object), reference to an object is made via it's object token which is supplied by the server. Objects are owned by the KS whose transaction created them, and they are protected from destructive operations by other KS's. They form an inheritance hierarchy in the usual manner (see figure 5.2) via their *Class* slots, while the subclass field acts to modify the methods. Changes in this field made by the server, give the object its additional polymorphic characteristics.

All objects are also part of a dependency hierarchy which is used by the implicit memory management and garbage collection (GC) of the server. The implicit nature of GC reduces the complexity required by a KS when requesting a service, while also endeavouring to make maximum use of the memory available to the server. At the top of this hierarchy is *All-Objs* (see figure 5.3) which is a permanent object

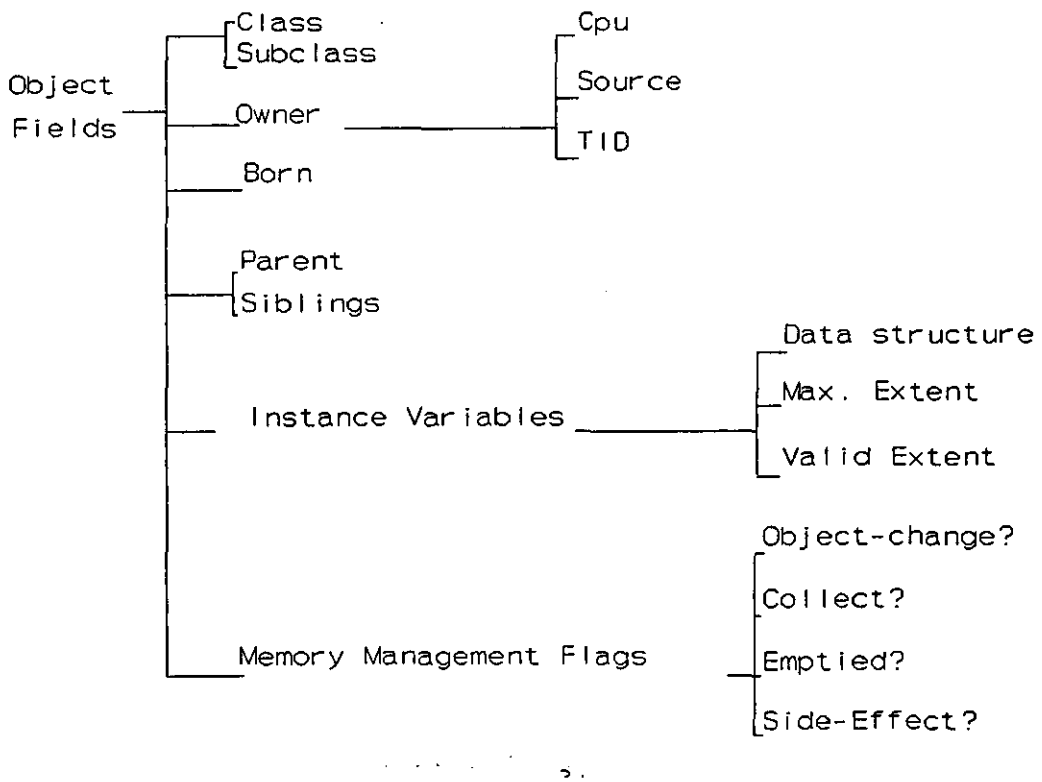


Figure 5.1 Object Fields.

of class *Scope* of *Scopes*. A *Scope* holds the objects created within a particular transaction. A number of *Scopes* can exist at the same time, allowing separate KSs to carry out what seem to the KSs to be, concurrent operations. A new *Scope* is created or an existing *Scope* is made the current *Scope*, at the commencement of processing a new message. If an object makes reference to data outside it's *Scope* (such as where a number of objects share the same source image), and if the method to be used is destructive, a copy of the original object is made in the local *Scope*.

The dependency hierarchy indicates which particular object was generated as a result of which parental data and is used to limit the possible interactions between competing KSs. The function of the GC is to maintain the truth of such relationships during the progress of a transaction, that is at any time, **the same method should be able to be applied to the parental data and still produce the same siblings.**

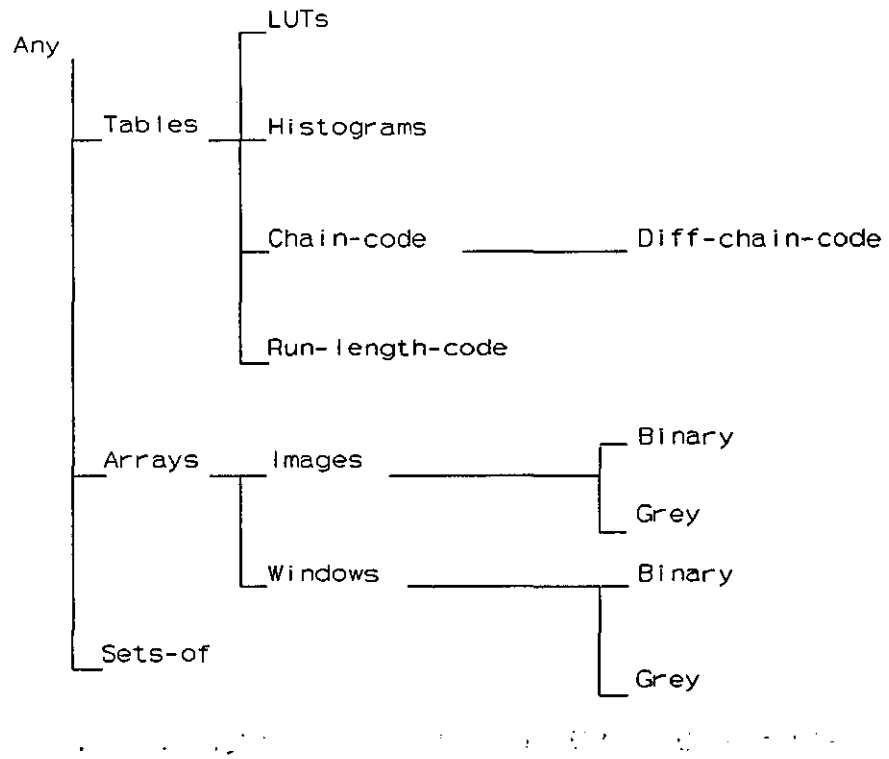


Figure 5.2 Method Inheritance between Object Classes.

The local instance variables (see figure 5.1) are used to define the maximum dimensions of the object data structure, and the valid region of that structure on which the methods can operate. For example after the completion of a number of methods (e.g. convolution with a 3x3 mask), the extreme pixels of the processed image are no longer defined and so as a result the valid region of the image is smaller than the maximum extent of the data structure. The automatic adaptation of the valid region reduces the amount of house-keeping that is required to be undertaken by the requesting KS.

Garbage Collection

The server provides automatic garbage collection to simplify programming and ensure maximum usage of the memory available. Its action is triggered by a change in state of the memory management flags within an object. These flags are set by the methods and indicate the change in the objects data structure or subclass which may allow garbage collection to take place. Garbage collection takes place when,

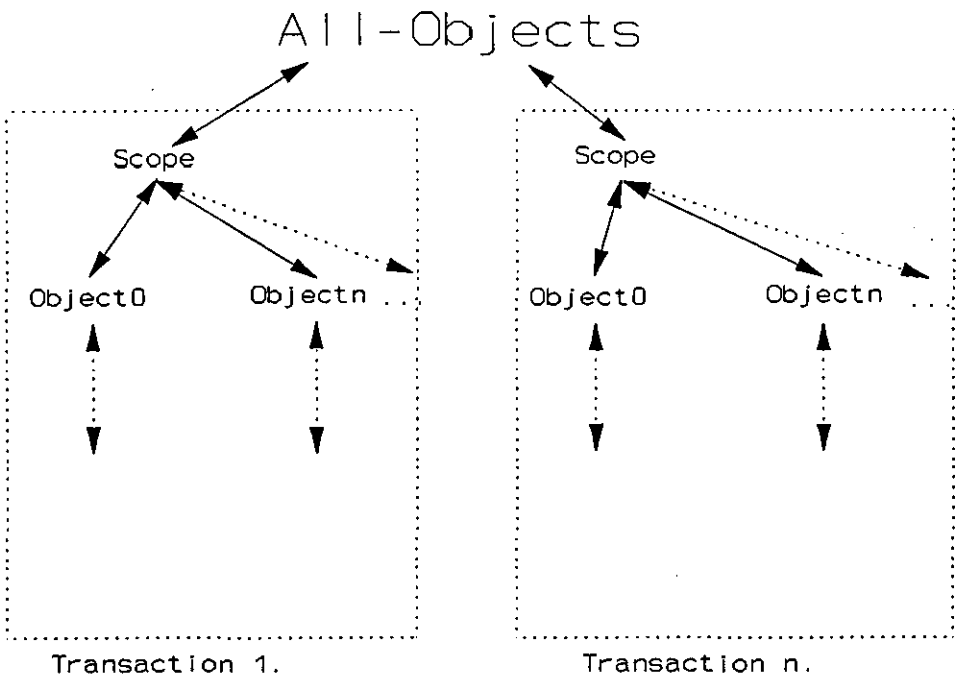


Figure 5.3 Scopes and Transactions.

1. A KS requests the contents of a particular object to be sent to it, indicating the end of the objects useful life. When this occurs the siblings are also no longer valid and so the object, all its siblings and their siblings are collected. This situation is indicated by the method setting the *Collect?* flag.

2. A transaction is completed and the contents of the corresponding scope are removed.

3. An object's subclass changes (say grey scale to binary image) the contents of the object remains valid, however, the validity of siblings is dependent upon the previous subclass of the object and so they are removed.(see figure 4)

4. Under certain circumstances the data structure can be deleted by the action of the method (i.e Certain forms of chain encoding algorithm), when this occurs it is indicated to GC by setting the *Emptied?* flag within the object. When the garbage collector is called it removes this object and moves its siblings to the parent of the emptied object (see figure 5.5).

However, there are still opportunities during a transaction when garbage collection could be performed, but which are not detected by the rules above. These situations arise when objects are formed as side effects of the eventual goal object. Such is the case when enhancing the contrast of an image by the use of histogram

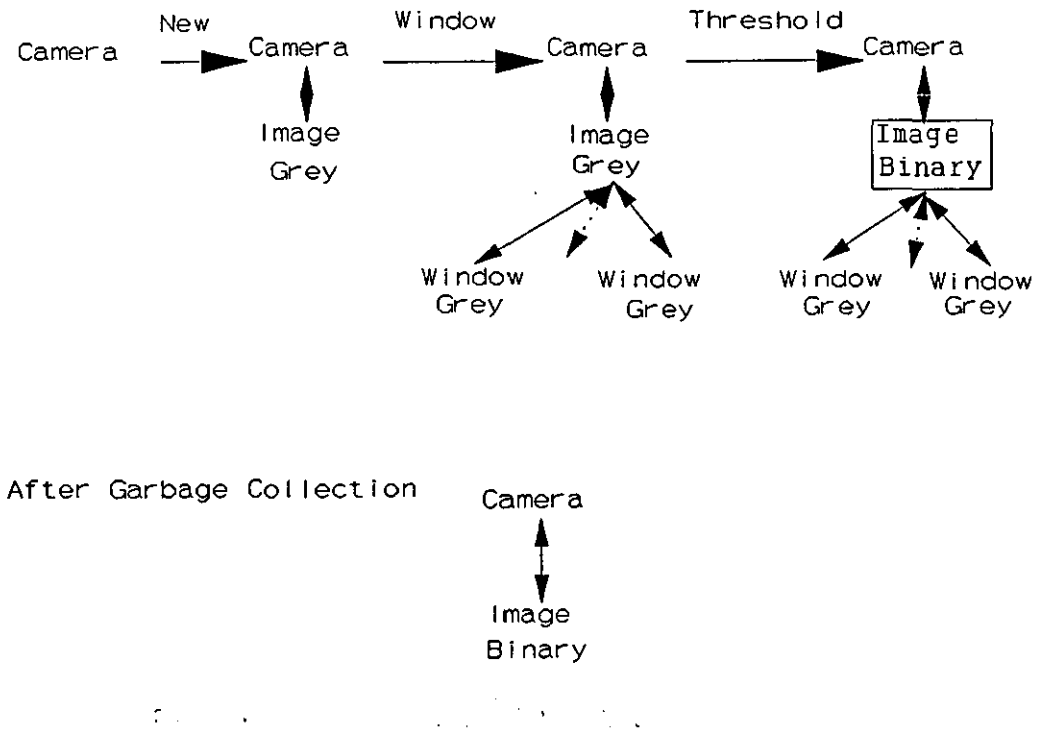


Figure 5.4 Changes in Object Class.

equalisation. During this process a look-up table (*LUT)* and *Histogram* are generated which have no further relevance once the contrast of the image has been modified. Since the detection of these situations would require the server to have an understanding of the intention of the KS's action, these cases cannot be implicitly processed by the GC.

Thus, in order to cope with these eventualities, there is a construction recognised by the message interpreter within the server known as *FORM*. *FORM* takes as its arguments an object and an expression, during the evaluation of the expression it marks all objects generated as side effects, except for the goal object specified in its arguments. When GC is called it removes all the objects and siblings marked as side effects, leaving only the goal object within the scope. In this way the calling KS can ensure that only necessary objects are maintained throughout a transaction, and the syntax of *FORM* underlines the intention of the transactions code.

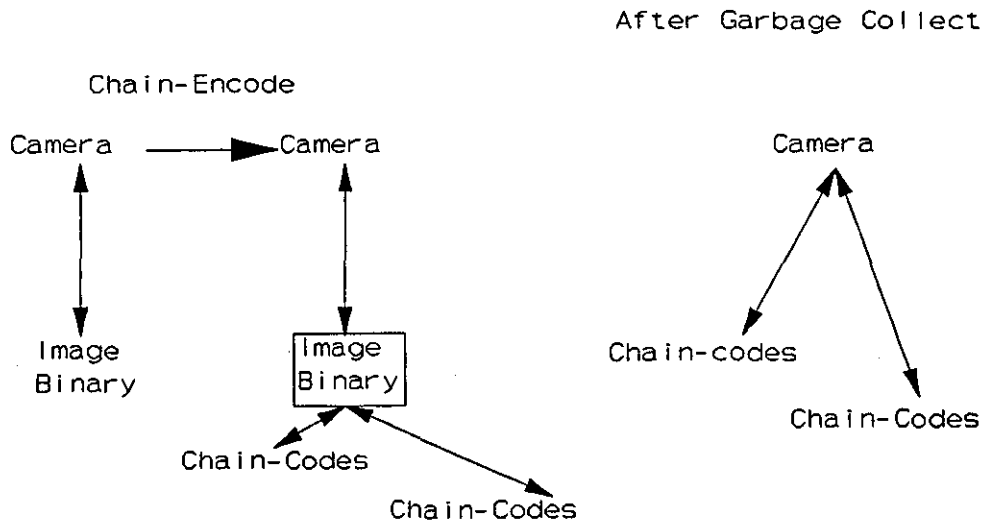


Figure 5.5 Emptying an Object.

Language

The main function of the language is to provide an efficient and flexible means by which combinations of methods can be specified, while keeping the message traffic to a minimum. The compromise solution employed within Plethora, uses a set theoretic approach to the language. This provides the unambiguity and flexibility that would be associated with the use of first predicate calculus but leads to a more efficient method of message interpretation (see [7] for a discussion of the equivalence of these two forms.)

The reason for the increased efficiency of the approach is two fold. Firstly, the *extensions* of the most commonly used predicates already exist as the contents of the slots of objects. Secondly, the set theoretic notation can be simply interpreted using a stack architecture and without the need for the computationally expensive unification processes.

Operations within the language can be categorised as those which change or interrogate the state of an object and those which change or interrogate the relationships between objects. For the sake of interpretation efficiency these are separately specified as *Assert*ions and *Make* respectively. Both categories of operations can be applied to single object or sets of objects, referred to by an *object token* supplied by the host agent. Methods are also tokenised, in order to reduce the length and the time taken to interpret the message. The "tokenising" procedure treats the ascii name string of the method as a bit pattern $\{x^n....x^0\}$ and applies the generator polynomial [8]

$$g(x) = x^{16} + x^{12} + x^5 + 1$$

to produce a 16 bit token. This function is a built-in and allows reference to methods without the need to enumerate them, while still maintaining a suitable dispersion between the method tokens (The present system uses 100 methods and as yet there has been no duplication of method token values). An absolute value, rather than a token to be evaluated, is preceded by "%".

The language also supports local variables which simplify the manipulation of the remote objects and so are used in those expressions applied to sets. The scope of the local variables and hence their contents, are valid for the duration of a transaction. General methods applied to sets include *For_all_members* and *For_one_member* as well as the more usual *Include, Remove, Union, Difference, Equal* etc.

Figure 5.6 shows the results of the vision server Request below.

```
&y Form
        new '' grey-scale-image $x Assert
        &x  '' rats
        &x  '' thin
        &x  '' Chain-encode     $y Assert
End-Form
0 $z Assert
&y For_all_members
        &l length  &z Gt
        *If
                &l length $z Assert
                &l        $r Assert
        *Then
End-Forall
&r Return
**
```

A new image is taken by the camera and thresholded to produce a binary image using a variable thresholder (based upon a technique known as RATS [9]). The image is thinned to produce an outline and then chain encoded. The blob with the largest chain code length is returned, the transaction is ended by **.

When the results of a *Request* are to be returned they are packed into the *Reply* using an agent independent format, and then unpacked by the *Requesting* agent on receipt. The packing and unpacking are recursive procedures which rely on the common representation of a limited number of primitive data structures (which at present include single variable, double precision variable, one and two dimensional arrays, lists and counted strings). This procedure allows complex objects to be sent between agents and simplifies result sharing. The structure of the Blob *Reply* is indicated in figure 5.7.

In a *Prompter,* where the *Reply* is formed by copying the state vector, the state vector acts as a message template (which contains the necessary object identifiers) and the *Prompters* action updates the contents slots of the message.

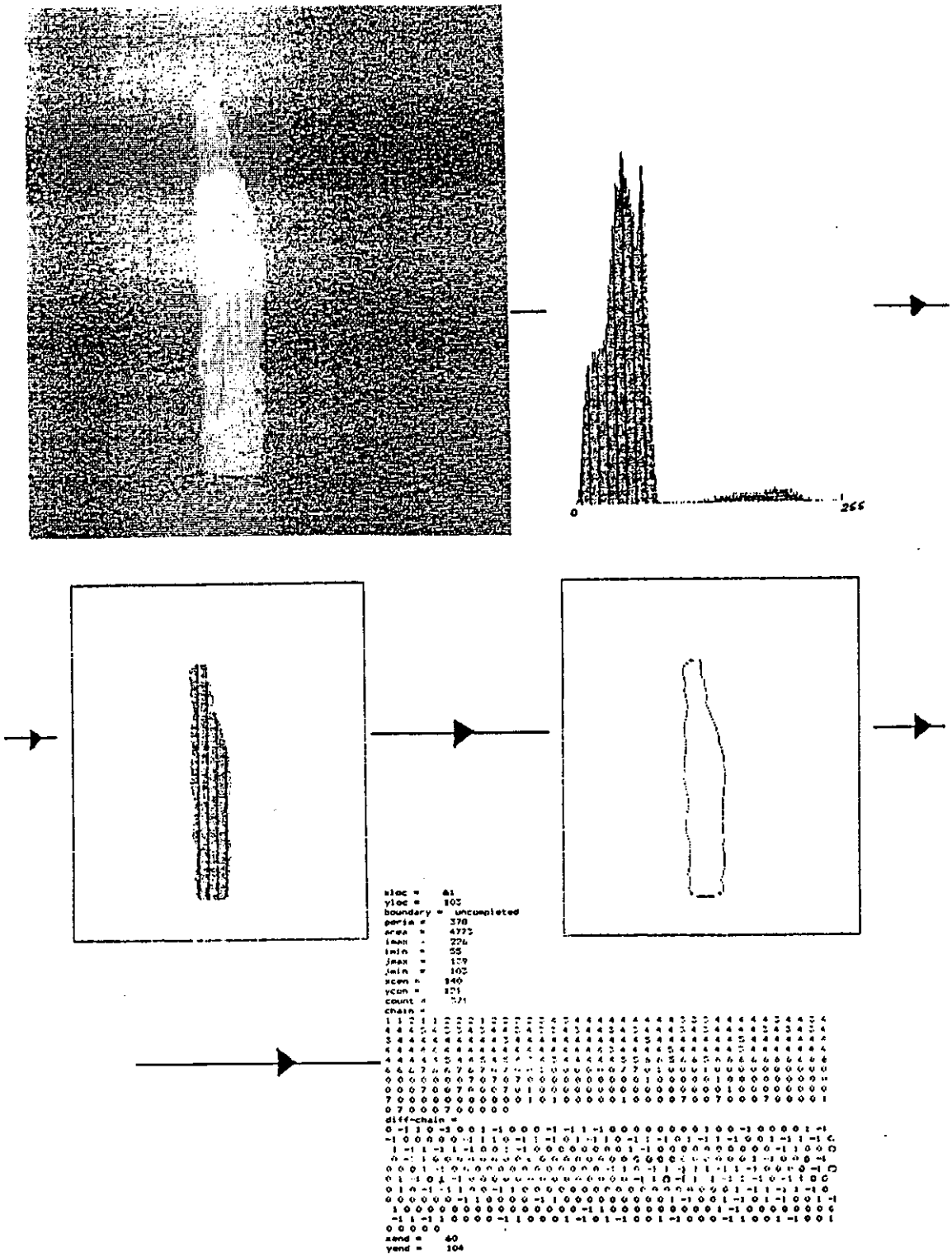


Figure 5.6 Images illustrating the progress of the message above.

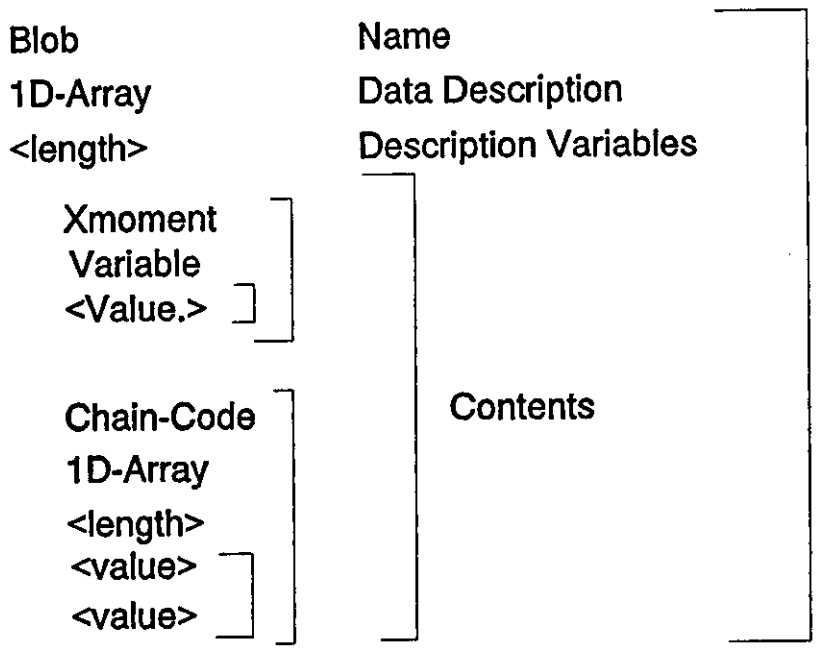


Figure 5.7 Reply Format for Blob.

Rules

An important class of object is the rule table. A rule table contains forward chaining production rules of the form,

Rule-Table <*name*>
Initialisation <*code*>
    Rule <*optional name*> <*Predicate*> ***If*** <*Action*> ***Then***
    Rule <*optional name*> <*Predicate*> ***If*** <*Action*> ***Then***
    .....
End-Rule-Table

a rule interpreter and local instance variables. They provide a formal means of specifying the procedural elements of an agent's behaviour and are used extensively to implement state machines and complex message interpretation mechanisms. A rule table in Plethora also has an initialisation section of user code, this is run prior to handing control of the table to the rule interpreter. The choice of rule interpreter determines the sequence of control through the rule table.

Rule Interpreters

HC State Machine (HCSM)

The simplest rule interpreter is the HCSM, which mirrors the implementation of the HC control found in [10]. It begins at the top of the table of rules and tests the predicate expression of each rule, the action of the first true predicate is executed and then the cycle begins again from the beginning of the table. This does not require conflict resolution, since only one predicate can be true at any time within the HC state machine.

Typically these rule tables are under the control of a *Prompter*. Figure 5.9 shows a part of the state transition diagram used to control an electrically actuated

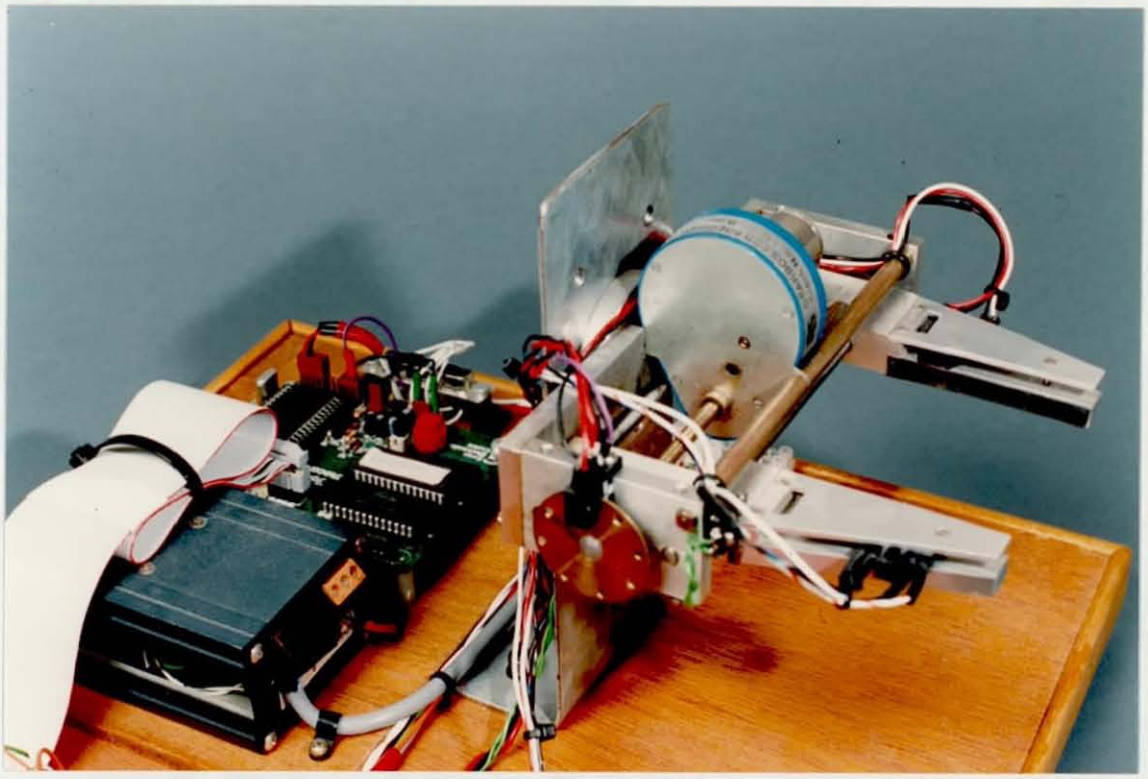


Figure 5.8 Photograph of experimental electrically actuated gripper.

gripper using hybrid force-position control and proximity sensors (see figure 5.8 for photograph of the gripper). It implements a simple form of Bang-Bang control, which was found necessary to overcome the stiction of the screw thread used to drive the fingers. The *Internal State* of the *Prompter* contained,

Error
The Difference between the demand and actual value of Command attribute.

PWM
The current Pulse width demand to DC motor control circuitry.
Full Reverse = 0 ; Full Forward = 255 ; Stop = 128;

Position
Jaws are either Fully open (fo), Fully closed (fc) or Midway

Motion
The jaws are either moving in a positive direction, moving in a negative direction or stopped.

The *Command State* contains two variables the command attribute, either Distance or Force and the demand value of the command. In the example given in figure 5.9, state 1 represents fully open and stopped, state 4 fully closed and stopped, while 2 and 3 are the Bang-Bang states between which it cycles. The final *Prompter* was able to control the gripper to perform the ubiquitous egg grasping exercise.

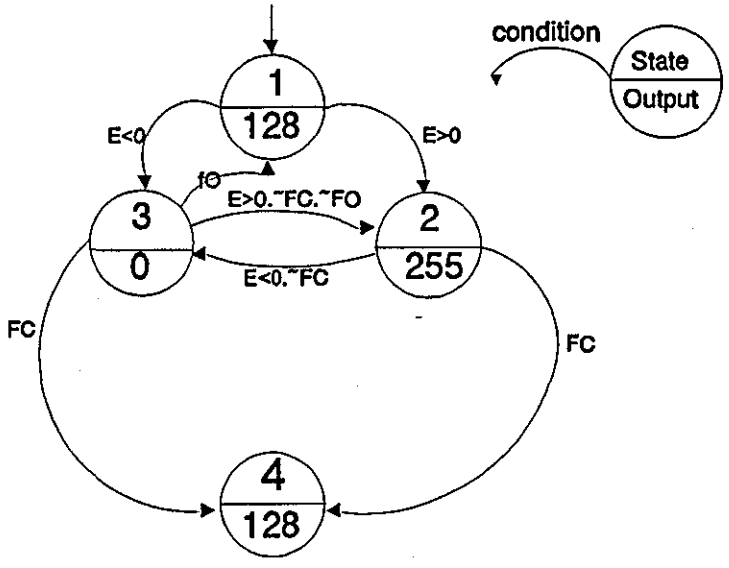


Figure 5.9 State transition diagram for hybrid force/position control of the gripper.

Procedural Interpreter

A less efficient rule interpreter, but one more suited to complex behaviours, such as message interpretation within a server agent, leaves the majority of the flow of control to the action of the rules themselves. This allows the rule table to recursively call itself, return from the current level of recursion, return from all levels of recursion or restart the cycle of rules from the beginning.

Rule tables can call other rule tables from within their action fields and each table has its own local instance variables. These characteristics can be used to increase the efficiency of a rule based implementation. Firstly, it is possible to reduce the search time of a large rule table by partitioning its search space into smaller tables of related rules. The smaller tables then can be invoked from a master table. Secondly, the called and calling rule table do not have to have the same type of rule interpreter. Thus the most suitable form of interpreter can be chosen for each operation, increasing the overall efficiency.

Implementation

Rule tables are implemented as an array of pointers to segments of code (see figure 5.10). Each segment holds the predicate or action of a particular rule (in a similar manner to that of FORPS [11]) and so provides efficient access to the necessary parts of the rule. Each rule table also has a local context (i.e. its instance variables, such as a pointer to current message, current position within the message etc) which can be manipulated from within the rule table.

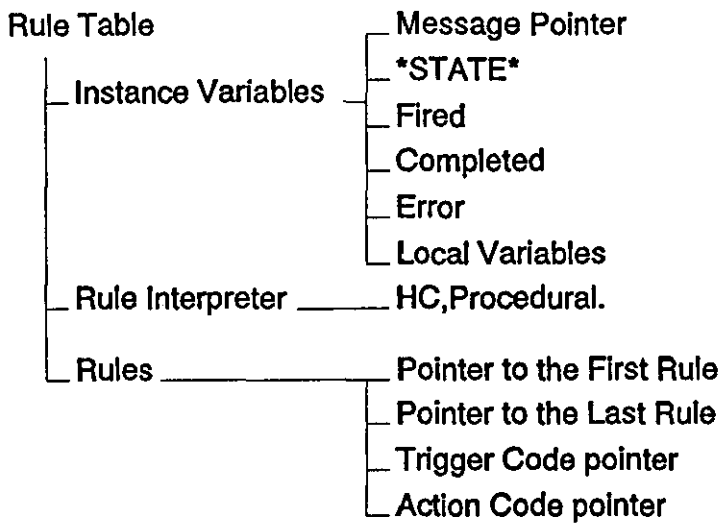


Figure 5.10 Rule table Attributes

Summary

Given a suitable selection of rule interpreters, the ability for tables to call one another, their local contexts and the additional initialisation field, rule tables can be used to tackle large state space tasks more efficiently than is possible with FORPS, while still satisfying applications with short critical times.

Rule tables provide a uniform means (given a suitable form of interpreter and agent) to represent the procedural element of both the real-time and knowledge based elements of Plethora. ROOP and the form of the inter-agent language, reduce the necessary housekeeping required of a KS and so contribute to the modularity of the system.

Although, neither of these elements fulfil a requirement in Chapter 1, they provide the underlying structure by which approaches to satisfy requirements in Chapter 1 are possible.

References

[1] Wegner,P.
"Learning the language", Byte, p245-253, March 89.

[2] Wegner,P. Zdonik,S.
"Inheritance as an Incremental Modification Mechanism, or What Like is and Isn't Like", Proc. ECOOP 88, Springer-Verlag, LNCS No. 322.

[3] Cardelli,L. Wegner,P.
"On Understanding Type, Data Abstraction, and Polymorphism", Computing Survey, p2-27, December 85.

[4] Henderson,P.
"Functional Programming: Application and Implementation", Prentice-Hall
ISBN 0-13-3331579-7, 80.

[5] Sillitoe,I.P.W
"An Approach to the Programming of Distributed Shared Resources within an Experimental Robotic Work Cell", EuroFORML'90, Southhampton, Oct. 90

[6] Bernstein,P.A. Goodman,.
"Concurrency Control in Distributed Database Systems", Computing Surveys, Vol 13., No.2, p185-221, 81.

[7] Grey,P.
"Logic, Algebra and Databases", Ellis Horwood ISBN 0-85312-803-0. 84.

[8] Tanenbaum,A.S.
"Computer Networks", Prentice-Hall, ISBN 0-13-164699-0, 81.

[9] Kittler,J. Illingworth,J.
"Threshold Selection Based upon a Simple Image Statistic", Computer Vision Graphics and Image Processing, vol. 30, p125-147, 85.

[10] Barbera,A.J. Fitzgerald,M.L. Albus,J.S. Haynes,L.S.
"RCS: The NBS Real-Time Control System", 16th ISIR, p19-1 19-33, 86.

[11] Matheus,C.
"The Internals of FORPS: A Forth-Based Production System", The Journal of Forth Applications and Research,Vol 4,No.1, p7-27, 86.

Chapter 6
World Modelling

The contents of this chapter are concerned with the on-line modelling of physical objects within the work cell and representation of facts about the objects. The chapter begins with a look at previous work in the area. It then goes on to analyse a simple construction task, which is used to identify the form and types of information that will be required. This is followed by a detailed description of the knowledge and representation within the geometric and relational networks.

Previous Work

Very little work has been aimed directly at the problem of on-line robotic world modelling. In the main, specific sub-problems have been addressed such as modelling for visual verification [1], automatic solid modelling [2], off-line path generation [3] and geometric databases for inclusion within task level languages [4]. An exception is the work of SMGR [5] and to a lesser extent NBS [6] and Purdue [7].

SMGR

SMGR splits the problem into two by employing separate geometric and relational models. The relational model is implemented as a semantic network, where nodes correspond to objects and the arcs represent constant relationships or physical transformations between objects. Multiple redundant structures are used to represent the object's relationships and are chosen to simplify the application of specific operators. A hierarchical organisation is employed to represent spatial relationships between objects, a tree structure to represent kinematic chains, and a graph of objects for the representation of rigid and permanent structures.

Multiple representations are also used within the geometric portion of the modeller. Objects are represented as generalised cones for use in spatial decision making, spherical representations for rough collision avoidance and a boundary

representation for all other uses, such as surface intersections. It was reported that it has been successfully used in a number of experiments to model on-line assembly processes. Only limited details of the exact representations and operators used in SMGR are given.

NBS

The dominant concern of the work in [6] is the provision of information from a 3D modelling system, to verify and predict sensory data for predicted scenes. Multiple geometric representations are used to simplify particular operations but they are all based around a central boundary representation. These extra representations include explicit representation of quadratic surfaces, and the use of rational bicubic splines for more complex surfaces.

Object representations are hierarchically organised in singly linked lists. Each element in the list can have an associated feature/value pair. The pair is chosen so as to best differentiate the object from the other objects in the system (given the set of extractable features available from the sensory system).

Purdue

Once again this work is concerned with sensory prediction, but rather than generating a 3D data base, it extracts an intermediate representation of the object features from a constructive solid geometry based modeller. It generates attributed relational graphs in which both nodes and arcs have attributes, where nodes represent surfaces and arcs the relationships between surfaces. The attribute graph is referred to as sensor-tuned and forms an intermediate level of representation. It can be derived from model based or sensory information and is created to facilitate the matching of partial object descriptions with the "complete" object model.

Which Knowledge?

In order to determine the necessary actions to complete the simplest of tasks, the system must have information about the shape and present position of the objects in the workspace. For the system to be able to plan that action it must also have information about the spatial constraints which must exist within such a construction. This information is contained within, what is known in the literature, as a *world modeller.*

The requirements of a robotic world modeller differ from current conventional CAD modellers in a number of fundamental ways [8]. Most conventional modellers are concerned with single objects in static situations. In robotic applications the modeller must deal with multiple objects, moving objects, relationships between surfaces of different objects and uncertainty in the object's position and shape. (Note: At the end of this chapter and in chapter 10, the functions of a world modeller given in [8] are discussed in the light of Plethora's real time requirements. This leads to a slightly different formulation of necessary requirements). For example consider the problem "Place the peg in the hole", during the planning and execution of the goal the modeller will be called upon to provide information in order to,

1. *Determine the final positions of the pieces in the construction.*
2. *Determine the expected freedom of movement of an object in a particular direction.*
3. *Update the modeller using sensory information.*
4. *Help predict sensory information based upon the physical properties of objects.*
5. *Identify collisions and provide information for the generation of collision free manipulator paths.*
6. *Determine physical properties, such as weight.*
7. *Determine the pose of elements within kinematic chains.*

Few of which can be derived from a conventional solid modeller. However, to achieve these efficiently, the information must also be in a form which is directly applicable to the function to which it is to be put. These characteristics are referred to as the Representational Adequacy (the ability to represent all the forms of knowledge that are needed in a domain) and Inferential Adequacy (the ability to manipulate the representational structures in such a way as to derive new knowledge inferred from old) of the modeller[8] respectively. They form part of the more general Frame problem in artificial intelligence[8].

Relational and Geometric Information

From the list of requirements, it can be seen that the information in 1-7 can be partitioned into strictly relational and strictly geometric parts. (i.e. 1 requires relational information to predict the constraints on the structure and geometric information to determine the parameters of those constraints). For the sake of efficiency these forms of knowledge are treated as two related planes of a frame network [9]. Hence the question of their representation is discussed separately.

Geometric Modelling

Geometric representation schemes can be broadly partitioned into wire-frame and solid representational schemes[10]. Wire-frames are a collection of curve segments that represent the object's edges (see [11] for a rigorous definition). They have several disadvantages, namely

The representation can be ambiguous.

They cannot be used to compute mass properties

They do not have an interference operator and so collision detection is not possible.

Solid Modelling

Solid modellers overcome these disadvantages and are distinguished by their unambiguous representation of the object. In general, they model the object using the following techniques or combination of techniques.

Swept volumes
Where an object is represented by a 2D outline swept along an axis. The shape of the axis determines the complexity and practicality of the representation. This technique was employed in ACRONYM[12] to represent it's volumetric primitives.

Boundary
An object is defined by a set of bounding polygons, which are themselves defined in terms of their edges, vertices, surfaces and normals[10].

Cellular/spatial occupancy methods
These methods assume that all space can be divided up into a large numbers of discrete cells which are either occupied by an object or not. It can be implemented simply as a 3D array, or a more sophisticated k-tree, in which the space is hierarchically ordered (e.g. Octree[10]).

Constructive Solid Geometry
Combines simple primitive volumes under set operations, in a hierarchical manner. These primitive volumes are themselves represented as any one of the other forms. The result is a set of simultaneous inequalities which must be solved to determine physical properties[10].

Cellular methods have been used in a number of related techniques, but only to specify single stationary objects. They have the disadvantage that their storage requirements must be traded against the resolution of the representation. Tips-1[13] is based upon a cellular strategy, it uses an array to form a coarse mesh and has special techniques at the level of the cell, to overcome the resolution problem.

Pure CSG schemes are not efficient when used to "pick an edge from" an object, and are inadequate for analysis of spatial interferences which require an intersection operator[10].

Swept representations are a natural means of describing solid volumes and they have been applied to a restricted class of volumes in APT and ACRONYM. However, at present there are few algorithms for use in computing object properties. Growing object's, which is the basis of a number of path planning schemes, is also difficult.

Hybrid scheme's, which use multiple representations, can exploit the particular advantages of a given representation for a particular function, and so provide a solution to the diverse requirements of the robotic world modeller. The following section describes the hybrid scheme used within the geometric portion of the world modeller (GeoMod).

GeoMod

The geometric plane of the network is linked to the relational via the geometric attributes of an *INDV* (through its *physical-property* slot). This points either to a *CHAIN*, *BODY* or *COMPOUND-BODY* frame. A *BODY* describes the volumetric and surface properties of the body which the *INDV* represents. A *COMPOUND-BODY* is used when a combination of bodies is necessary to describe an *INDV's* physical attributes. A *CHAIN* frame indicates that the *INDV* is an open linear kinematic chain and it forms the head of a list of *BODY* frames.

BODY Frames

The slots of a *BODY* frame are shown in figure 6.0. They can be partitioned in to those which represent the spatial relationships within the *WORLD* (i.e. *Parts* and *Subparts*), those which describe it's variable ones (i.e. the variable relationship group) and the fixed relationships between the component volumes within the *BODY or COMPOUND-BODY.* The variable relationships indicate the *BODY* is part of a kinematic chain.

The *BODY* frame forms a hierarchy of other *COMPOUND-BODY/BODY* frames through the *PART-OF* and *PARTS* list. It's position in the *WORLD* is defined with respect to the *COMPOUND-BODY/BODY* which it is a *PART-OF,* via that

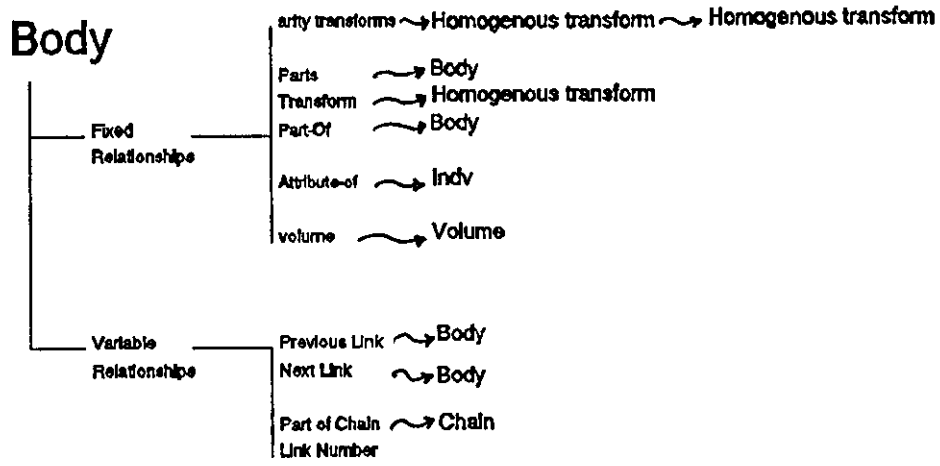


Figure 6.0 Body Frame.

BODY's coordinate *PART-TRANSFORM*. At the top of the hierarchy is the *WORLD*, which is a *PART-OF* itself. The hierarchy terminates in *BODY* frames (i.e. frames with a single *VOLUME*). These *VOLUME* frames contain the quantitative information about the *BODY*.

A *COMPOUND-BODY* is represented as a list of *BODYs* or other *COMPOUND-BODYs*, each of which belongs either to the sets of holes (*HLES*) or solids (*SLDS*). Hence, each separate piece of a *BODY* has an associated frame and a name. This makes referencing parts much easier and more efficient than referring to a group of surfaces within a single monolithic representation.

The distinction between *BODYs* and *COMPOUND-BODYs* and the explicit separation of *SLDS* and *HLES* in the *COMPOUND-BODY*, although not absolutely necessary (e.g. Whether a *BODY* is a *HLE* or *SLD* could have been determined via the inheritance mechanism) increases the efficiency of the methods. It does so by

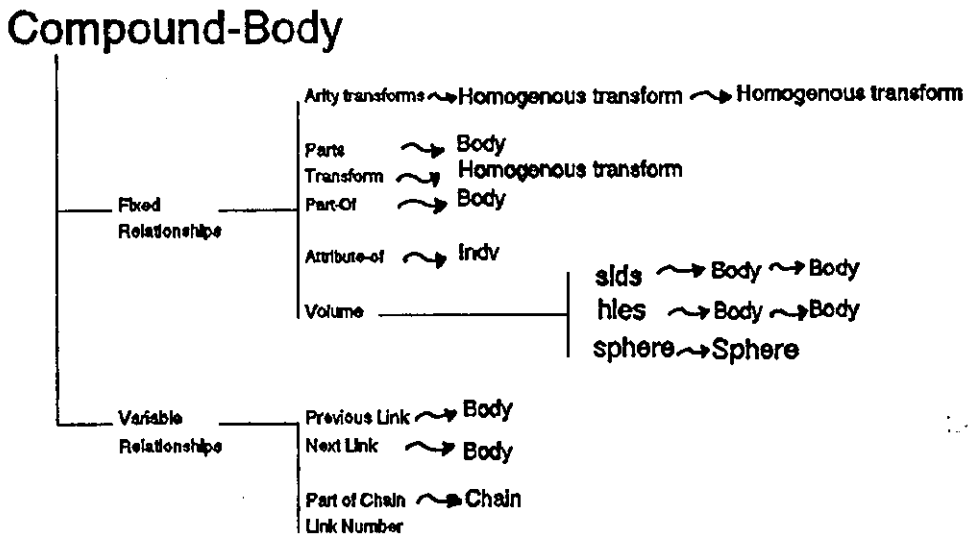


Figure 6.1 Compound-Body Frame.

reducing the number of coordinate compound transformations and use of the inheritance mechanism (which would have been required by a more uniform representation).

VOLUME Frames

The *VOLUME* frames are a set of primitive volume descriptors (e.g. hemisphere, cone, rectangular parallelepiped and cylinder. See figure 6.2) or user defined descriptors which when used in conjunction, build up a description of a particular *COMPOUND-BODY* through its *BODY* frames. An example in composition of a *COMPOUND-BODY* is given in figure 6.3, which models the volume of a screw as two cylinders and a cone.

VOLUME Hierarchy

Each *VOLUME* (see figure 6.4) node is composed of a number of multiple representations which form a hierarchy. At the top are bounding volumetric approximations to the object, the first is a single sphere, beneath which there are a number of smaller overlapping spheres enclosed by the first sphere. Both

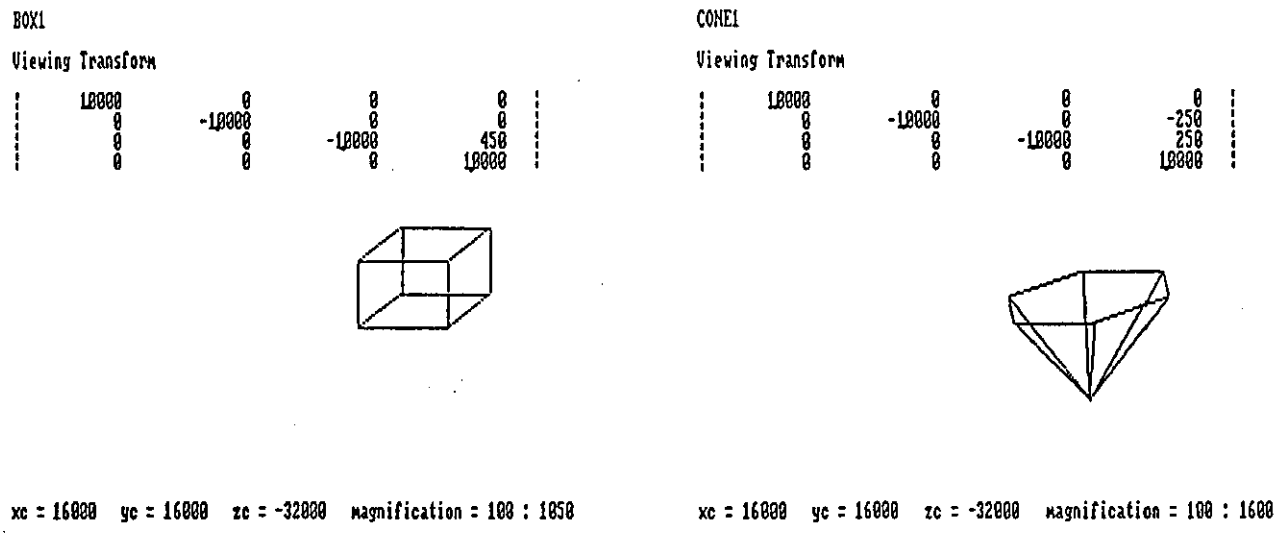


Figure 6.2 Screen dump of primitive volumes taken from GeoMod.

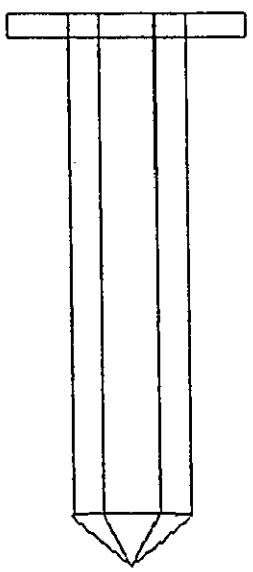


Figure 6.3 Compound-Body Screw.

representations are derived from an imaginary rectangular parallelepiped (RP) which bounds the boundary representation of the volume. Below these volumetric representations is a conventional boundary representation[4], comprising a hierarchy

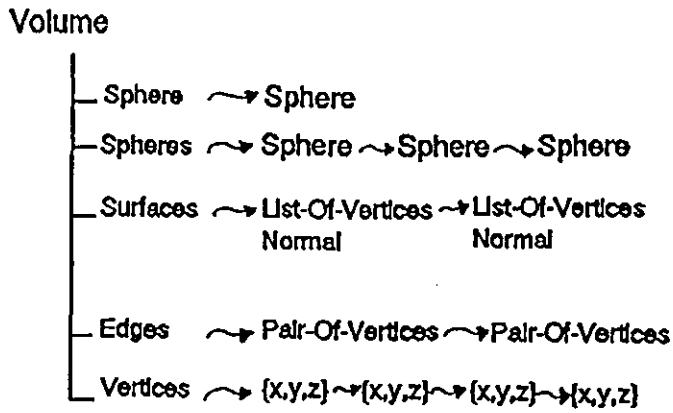


Figure 6.4 Volume Frame.

of planar polygon surfaces, their normals, edges and vertices. Both forms of representation are specified with respect to the *VOLUME*'s local frame of reference.

There is necessarily a trade off between the number of smaller spheres used to bound the RP and the accuracy of this representation. The size and position of the spheres are chosen according to a minimum volume error criteria and they are positioned along the major axis of the RP.

The procedure begins by splitting the RP into a number of smaller equally sized RPs along it's major axis and enclosing these with spheres of radius, a, which ensure minimum volume error (see figure 6.5). The derivation of the expression for a in terms of the RP's end face is given over the page, where E is the error in volume.

The procedure continues by fitting at one end of the RP, a sphere at a distance a from the end face, the next sphere is placed at a distance of $2a$ from the centre of the previous sphere and so on until the final section of the RP. The final sphere is placed at a distance a from the far end face, so as to enclose the volume left over and

$$E = \frac{4\pi r^3}{3} - 8xya$$

$$E = \frac{4\pi}{3}\,[x^2 + y^2 + a^2]^{\frac{3}{2}} - 8xya$$

$$\Rightarrow \quad \frac{dE}{da} = 4\pi a\,[x^2 + y^2 + a^2]^{\frac{1}{2}} - 8xy$$

$$When \frac{dE}{da} = 0 \quad \Rightarrow \quad a^4 + (x^2 + y^2)\,a^2 - 4\left(\frac{xy}{\pi}\right)^2 = 0$$

$$Let\ b = a^2 \quad \Rightarrow \quad b = -\frac{1}{2}(x^2 + y^2) \pm \frac{1}{2}\left[(x^2 + y^2)^2 + 16\left(\frac{xy}{\pi}\right)^2\right]^{\frac{1}{2}}$$

$$But\ a^2 > 0 \quad \Rightarrow \quad b = \frac{1}{2}\left[(x^2+y^2)^2 + 16\left(\frac{xy}{\pi}\right)^2\right]^{\frac{1}{2}} - \frac{1}{2}(x^2 + y^2)$$

$$Since\ a > 0 \quad \Rightarrow \quad a = \left[\frac{1}{2}\left[(x^2 + y^2)^2 + 16\left(\frac{xy}{\pi}\right)^2\right]^{\frac{1}{2}} - \frac{1}{2}(x^2 + y^2)\right]^{\frac{1}{2}}$$

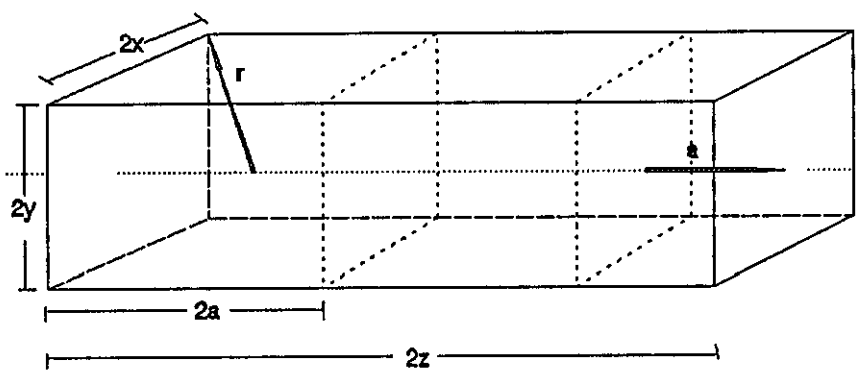


Figure 6.5 Partition of the Bounding RP.

still fulfil the minimum error criteria (see figure 6.5).

A *COMPOUND-BODY* also contains a bounding sphere, however, these bound the single sphere volumetric representations of its component *BODYs* (see figure 6.6, where the dotted spheres represent the individual *BODY* spheres, solid ones the *COMPOUND-BODY* spheres and r, the radius of the final bounding sphere for the entire body). Here

$$r = \{ ((z_{max}-z_{min})/2)^2 + ((y_{max}-y_{min})/2)^2 + ((x_{max}-x_{min})/2)^2 \}^{1/2}$$

$$c = \{ (x_{max}-x_{min})/2,\ (y_{max}-y_{min})/2,\ (z_{max}-z_{min})/2 \}$$

The hierarchy formed by this representation provides a means of increasing the

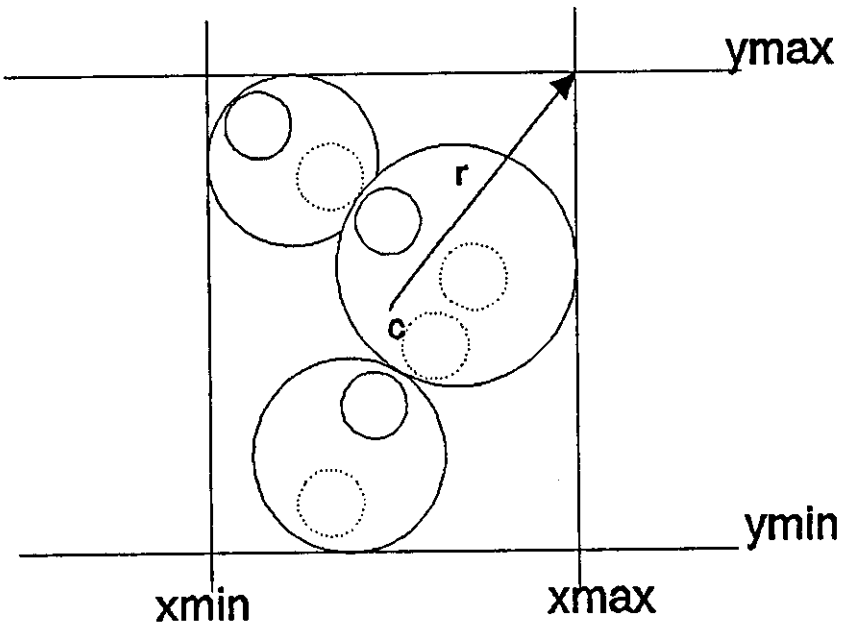


Figure 6.6 Sphere bounding a Compound-Body.

efficiency of many of the fundamental operations of the modeller. It allows the modeller to apply efficient tests based upon the volumetric representations, to prune the search before applying the more computationally expensive boundary operations.

Interference Test

Interference tests between objects are the basis of a number of methods employed within Geomod including collision detection. An interference test is usually computationally expensive and indicates whether two objects overlap or touch. The method employed within GeoMod takes advantage of the volumetric hierarchy to reduce the complexity of interference tests. It does so by allowing simple approximate tests to remove the obvious cases, before proceeding to make more detailed and complex tests on the more likely candidates. For example in figure 6.7, when determining whether a *BODY(a)* intersects with another *BODY(b)*, the spheres at the top of *COMPOUND-BODY* hierarchy (or in the case of a *BODY* with only *PARTs*,

the *PARTS list)* are tested for intersection using the **necessary** intersection predicate,

$$r_a + r_b \geq \{ (z_a-z_b)^2 + (y_a-y_b)^2 + (x_a-x_b)^2 \}^{1/2}$$

If there is an intersection, then the sphere *BODY(b)* is expanded and *BODY(a)*'s sphere is retested against the more detailed model of *BODY(b)*. This

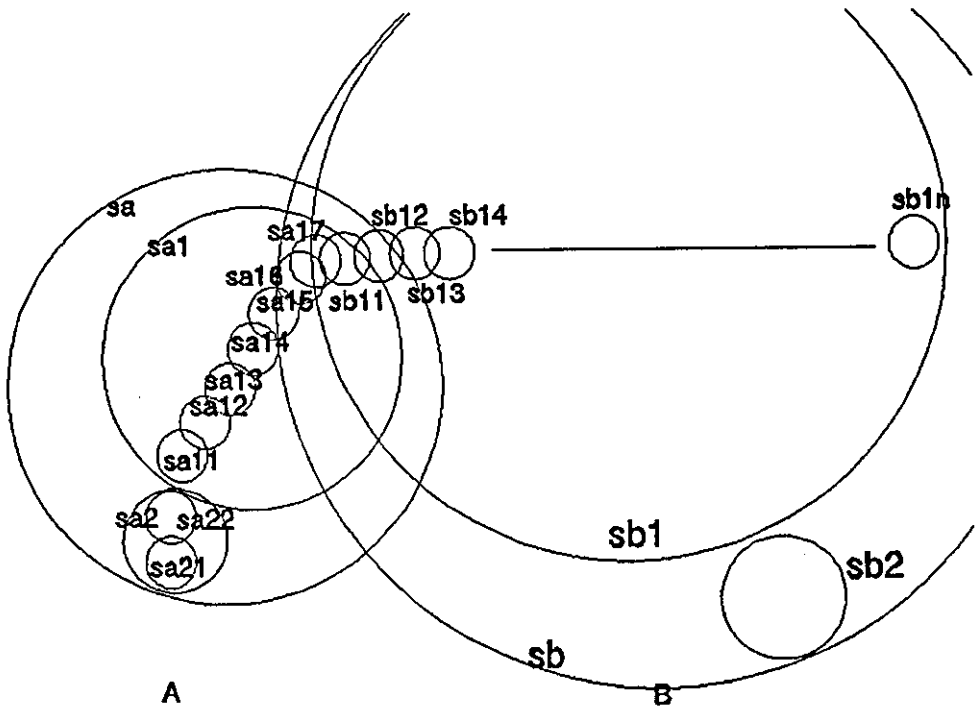


Figure 6.7 Hierarchical Intersection test.

process continues in a breadth first fashion until the spheres which represent the primitive *VOLUME*s are reached. If there remains an intersection then BODY(a) is expanded and the sphere tests are performed in the same manner until the bounding spheres of *BODY(a)* are reached. By this point the potentially interfering parts have been identified and it is now necessary to make the more expensive boundary interference tests.

These are well known and are performed in the usual manner. Firstly, the groups of facing surfaces are determined using the cross product of their surface normals, then vertices/surface interference tests are performed (details are to be found in [14]).

The steps below describe the order of the intersection tests between the spheres in figure 6.7, where $\bigcap$ is used to indicate the intersection test and a starred expression, an intersection.

1. ${}^{*}S_a \bigcap S_b$

2. ${}^{*}S_a \bigcap S_{b1}$; $S_a \bigcap S_{b2}$;
3. ${}^{*}S_a \bigcap S_{b11}$; ${}^{*}S_a \bigcap S_{b12}$; ${}^{*}S_a \bigcap S_{b13}$; $S_a \bigcap S_{b14}$; $S_a \bigcap S_{b15}$; ..... ;

4. ${}^{*}S_{a1} \bigcap S_{b11}$; ${}^{*}S_{a1} \bigcap S_{b12}$; $S_{a1} \bigcap S_{b13}$;
   $S_{a2} \bigcap S_{b11}$; $S_{a2} \bigcap S_{b12}$; $S_{a2} \bigcap S_{b13}$;

5. $S_{a11} \bigcap S_{b11}$; $S_{a12} \bigcap S_{b11}$;..... $S_{a16} \bigcap S_{b11}$; ${}^{*}S_{a17} \bigcap S_{b11}$;
   $S_{a11} \bigcap S_{b12}$;.....;

(Note: Other more efficient sphere intersection tests, based upon the sufficient D_4 and D_8 distances metrics, were investigated. However, the extra expansions required as a result of the approximate nature of the tests out-weighed the increase in their efficiency).

CHAIN Frames

Open kinematic chains are modelled using the modified Denavit-Hartenberg formalism found in [15]. This encodes the coordinate transformations between joints, using four scalars α_{i-1}, Θ_i, a_{i-1} and d_i. For a prismatic joint the joint variable is d, for a revolute joint, Θ. The motion of the joint is limited to translation along or rotation about the z-axis, such that the coordinate transformation between two joints i and i-1 is given by the compound transformation,

$$ {}^{i-1}_{i}T = \mathrm{ROTX}(\alpha_{i-1}).\mathrm{TRANSX}(a_{i-1}).\mathrm{ROTZ}(\Theta_i).\mathrm{TRANSZ}(d_i) $$

or when expanded explicitly,

$$\begin{bmatrix} c\Theta_i & -s\Theta_i & 0 & a_{i-1} \\ s\Theta_i.c\alpha_{i-1} & c\Theta_i.c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}.d_i \\ s\Theta_i.s\alpha_{i-1} & c\Theta_i.s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}.d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $c\Theta_i$ is $\cos(\Theta_i)$ etc. To allow for more complex arrangements (such as spherical joints) fictitious parts each with a common origin are used to perform the extra transformations.

The formalism is implemented using *CHAIN* frames and the variable relationship slots of the *BODY* frames. A *CHAIN* (see figure 6.8) contains a pointer

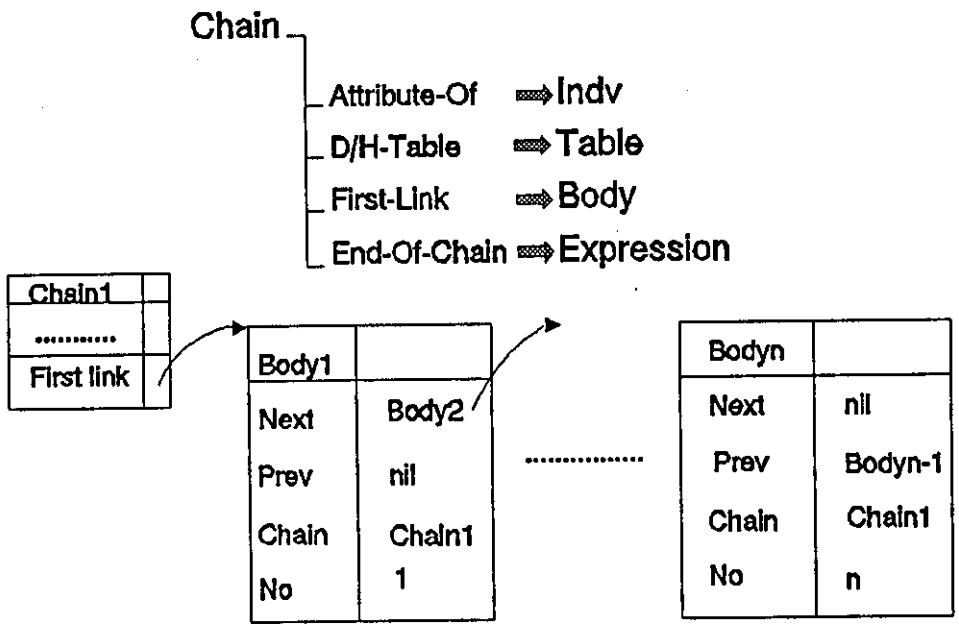


Figure 6.8 Chain Frame.

to the *INDV* whose physical properties it represents, the Denavit-Hartenberg (D/H)

parameter table and a pointer to the *BODYs* which constitute its links. The *Link No.* describes the forward relative coordinate transform to the *Next-Link* in terms of the entries in the D/H table. Pointers back along the chain (indicated by *Previous-Link*) are used to determine the position of a link with respect to the world coordinates. This is achieved by evaluating the inverse of the D/H transforms along the chain until it reaches the base (indicated by a Nil *Previous-Link*).

An optional slot within the *CHAIN* descriptor holds an executable function which returns the position of the end of the chain. Often there is a simple geometric solution to the forward kinematics of a manipulator and its use reduces the cost of

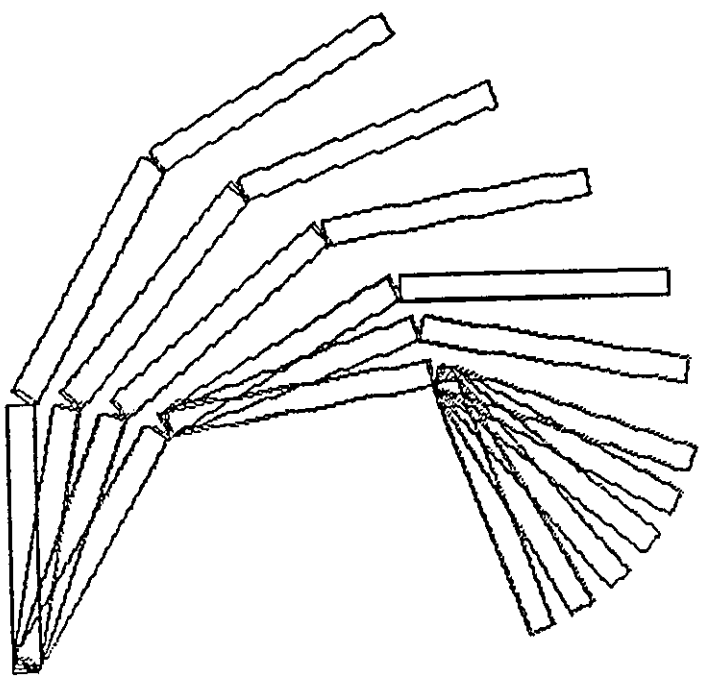


Figure 6.9 A screen dump of example Chain.

what is otherwise a computationally expensive operation. However, the link list is still necessary for the determination of the manipulator limbs in collision avoidance.

A *CHAIN* does not have a bounding sphere, since updating such a sphere would effectively require the re-calculation of the forward kinematics for each change

in joint variable. This is extremely computationally expensive and so sphere representations of the chain are calculated when required.

Collision Detection

A KS will often need to determine whether a potential collision is about to take place (e.g. when planning a manipulator path). The description of GeoMod so far does not include any reference to the modelling of motion. This is still an area of current research and no conclusive technique has been found which has sufficient generality and the necessary efficiency. The following section discusses current approaches and indicates how limited modelling of motion is achieved within GeoMod.

Solids In Motion

Collision detection, and the more general area of modelling solids in motion, is difficult. There are three fundamental techniques used at present. The first determines the movement of the object as a swept volume and checks for intersections between it and other objects. The second incrementally changes the position of the object and after each interval checks for intersections. The third creates a model of the shape and motion of the object in four dimensions or higher, and looks for intersections in that space.

Swept Volume Representation

A swept volume technique has been proposed by Boyse[16]. This was restricted to single bodies moving under translation or under rotation only, and may be a practical solution to a number of work cell situations. The technique parameterises the equations for the paths and objects and tries to solve them simultaneously to determine a collision. Noname[17] uses a similar technique, but represents the path as a swept sphere which increases the speed of the process. However, these techniques do not include time as a variable and are limited to the movement of a single object.

Incremental Motion

Incremental techniques are computationally expensive but are simple to implement and also cope with a large class of problems.(Note: Although they have difficulty with modelling sliding contacts) Techniques exist to improve their efficiency over that of the most naive approach. They rely on looking ahead to a time when there might be a potential collision. Coarse steps are made up to that point, when higher resolution steps are made. However, great care has to be made in the choice of method when determining the point of

potential collision. Cameron[18], used a divide+conquer technique which while increasing the response of the algorithm, he noticed that the determination of the potential collision point took five times as long as the interference test itself.

Algebraic Representations

Robomod[19] included a time dimension in the equation representing the normal to the surface of an object.

$$ax + by + cz + dt + e = 0$$

This was used to solve the collision problem for two objects in linear motion. This technique is not general and it is time consuming and difficult to represent[19].

Summary

It is clear that modelling solids in motion is both difficult and time consuming. For this reason it must be restricted to the off-line sections of Plethora's behaviour. Techniques which rely on modelling motion during the on-line sections (such as many path planning techniques), need to be reformulated so as not to need this form of modelling.

An alternative efficient path planning KS, which does not rely upon the direct modelling of motion is provided in Chapter 8. However, for those occasions where modelling motion is necessary Geomod uses an incremental technique in conjunction with the hierarchical interference checker to provide a limited means of modelling solids in motion.

Collision Detection in Geomod

The technique uses a swept volume of the path of the object (where the object is modelled as a sphere), a simple prediction of where the likely collisions are to occur, and the use of the hierarchical interference test for the detection of collisions for a single object. The technique is limited to simple translational paths of solids, but could be extended to more complex paths using [15]. The prime purpose of the technique is modelling fine motion during the determination of assembly sequences (this is described later in the chapter).

The workpiece is modelled using it's single bounding sphere and hence, it's path is modelled as a cylinder. The likely areas of collision between the path and

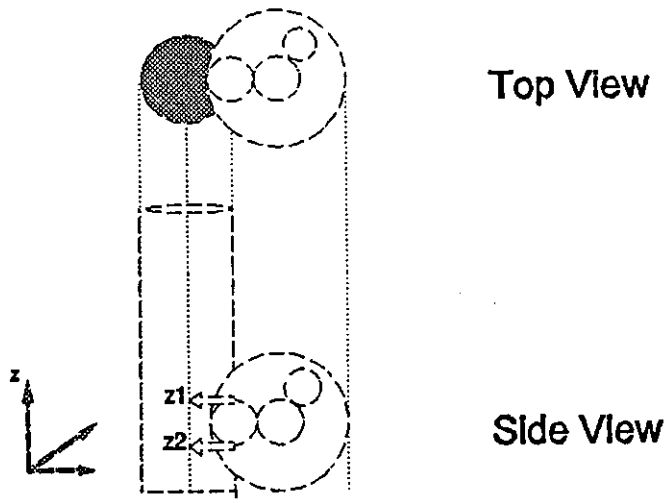


Figure 6.10 Simplified Collision test.

other workpieces are highlighted by intersections in the x-y plane. These are between the cross section of the cylinder and the x-y circles of the pieces (see figure 6.10). The extent of the workpiece in the z direction, which forms the potential collision, is used to determine the zone over which the incremental interference tests will be performed. The workpiece is then moved in steps of 1 mm through the region of possible collision (z_1-z_2 in figure 6.10) and at each point the hierarchical interference test is performed to determine whether or not a collision has occurred.

Relational Information

The relational plane of the world modeller is used to provide relational information about constructions and the relationship between physical objects and their functions. Three types of frame are used to accomplish this, they are the individual (*INDV*), the set (*SET*) and the set descriptor (*SETD*). *INDV*'s inherit all the attributes of the *SET* to which they belong. Each member of the *SET* provides a mutually exclusive alternative to its fellow members. *SET*s can have *SET*s as their members and so form a set-superset hierarchy. The attributes of a *SET* are described by it's *SETD*. This contains it's *FUNCTION* (i.e. the context in which the *SET* is used) and the *Role* it plays within that context.

The possible forms of inheritance between frames are shown in figure 6.11(a). This precise structure and definition of inheritance avoids the problems of intuitive schemes. The structure of the frames interconnection is shown in figure 6.11(b), although *SET*s may also share a *SETD* and *INDV* may also have a number of *EQU*ivalent *INDV's*.

Relationship	Function	Roles	Context	Members
Isa/Isamemb	✓	✓	✓	INDV-SET
Type/Tmemb	✓	✓	✓	SET-SETD
Spec/Spmeb	✓	✗	✗	SETD-SET
Sub/Smemb	✗	✓	✓	SET-SET

Figure 6.11(a) Forms of allowable Inheritance.

The hierarchy culminates in the *SET UNIVERSE*, which is a member of itself. Immediately below the *UNIVERSE* the network is partitioned into *INFORMATION* and *THINGS*. *THINGS* take part in the construction process and may have *INDV*

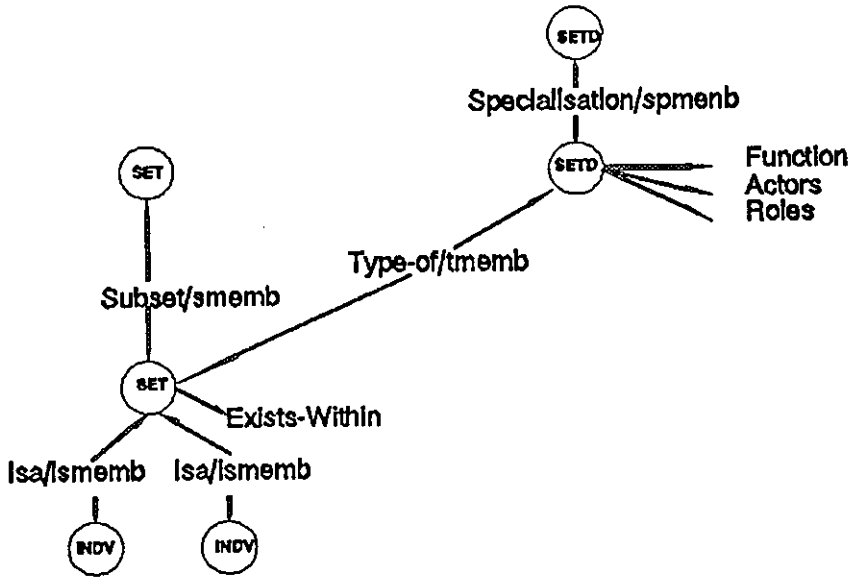


Figure 6.11(b) The inheritance mechanism within the relational network.

terminal frames with physical properties. While *INFORMATION* frames terminate in descriptions of primitive relationships between *SETs* or in *SETs* themselves. A diagram showing part of the relational frame network is given in figure 6.12. (Note: Whenever a new item is added to the network its relationships are checked to ensure consistency with figures 6.11(a,b)).

One of the important functions of the relational plane of the network is to describe assemblies and fixtures. The names of the frames provide the vocabulary for the system when discussing the construction (i.e. goal and *Role* names etc). The next section describes how the network can be used to interpret declarative descriptions of assembly tasks (completely fulfilling requirement *kn2* and partially fulfilling *ct1*).

Descriptions of Spatial Constraints

A major function of the *INFORMATION* branches of the network is to provide declarative descriptions of *CONSTRUCTIONs*. This information is analysed by KSs and used to enter the goals on the BB. A *CONSTRUCTION* is described in terms of *ASSEMBLYs* and *SUB-ASSEMBLYs*, which in turn are described by *FIXTURES*

between *WORKPIECES* or *SETs* of *WORKPIECES*. *FIXTURES* are domain terms which describe constraints between the *WORKPIECES*. The constraints are described in terms of the *ACTOR* and *ROLE* fields, *ACTOR's* describe the *SET's* of allowable participants in the construction and the *ROLES* the constraints between those participants. A *FIXTURE*'s *ROLES*s are described in terms of *PRIMITIVE* relationships which hold between two coordinate frames. An *ASSEMBLY*'s *ROLEs* are a mixture of *FIXTURES*, *PRIMITIVES* and other *SUB-ASSEMBLY* nodes. In this way a hierarchical description of an assembly task can be built from other simpler descriptions.

Each *SET* of *WORKPIECES* has a transform arity, which specifies the number of ordered coordinate frames which each member must have. The transforms

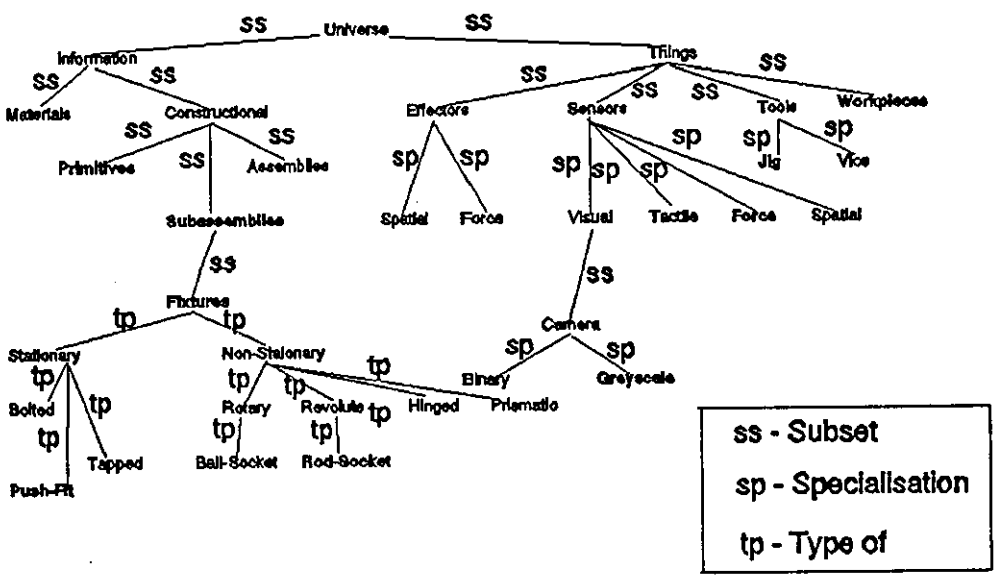


Figure 6.12 An outline of the relational network.

correspond to the position of the coordinate frames affixed to the *WORKPIECE*, each of which corresponds to it's use in a *FIXTURE*.

The *PRIMITIVES* within the *FIXTURE* specify relationships between these coordinate frames and so describe the constraints on a *FIXTURE*. The simultaneous

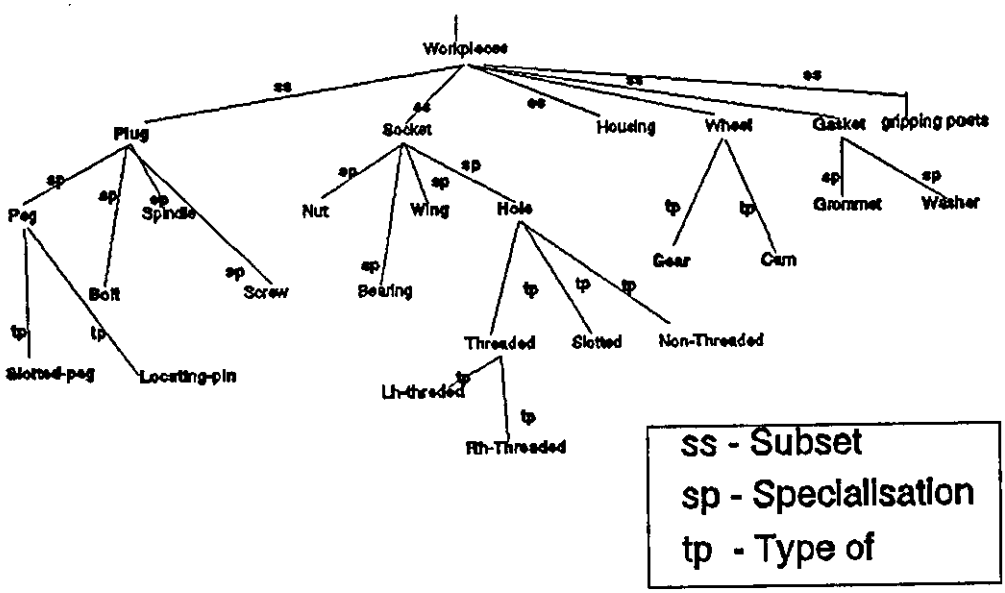


Figure 6.13 An outline of the workpiece network.

solution of the constraints for a particular *WORKPIECE* gives it's position within the construction.

Primitive Coordinate Frame Constraints

The *PRIMITIVES* are the same as those homogeneous transformation operators used in [20] for describing manipulation. They specify constraints in terms of coincident origins and aligned axes. Given that a homogeneous transform can be expressed as,

$$\begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where l, m, n and p are 3x1 vectors, then the primitive relationships can be written in terms of homogeneous transforms as :-

Star :
$$\begin{bmatrix} ? & ? & ? & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hat :
$$\begin{bmatrix} ? & ? & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equal :
$$\begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where ? represents a degree of freedom.

Figure 6.14 illustrates diagrammatically the relationships specified by the *PRIMITIVES*. A Star relationship indicates that the frames should share the same origin, leaving all rotational degrees of freedom unspecified. A Hat holds between two frames with a common origin and who also share common z axis directions. Finally, for an Equal relationship to exist the frames must coincide.

Groups of primitives which refer to the same workpiece can be solved using the solutions to the simultaneous equations found in [20]. (Note: there are a number of typographical errors in the original paper, and so for clarity an example proof is given in Appendix C). A solution exists for the position of a workpiece, if (a) there is an Equal relationship between a part and another whose position is known; (b) between a part and three others with which three Star relationships hold; (c) a part and two other parts where two Hat relationships hold; (d) a Star and a Hat

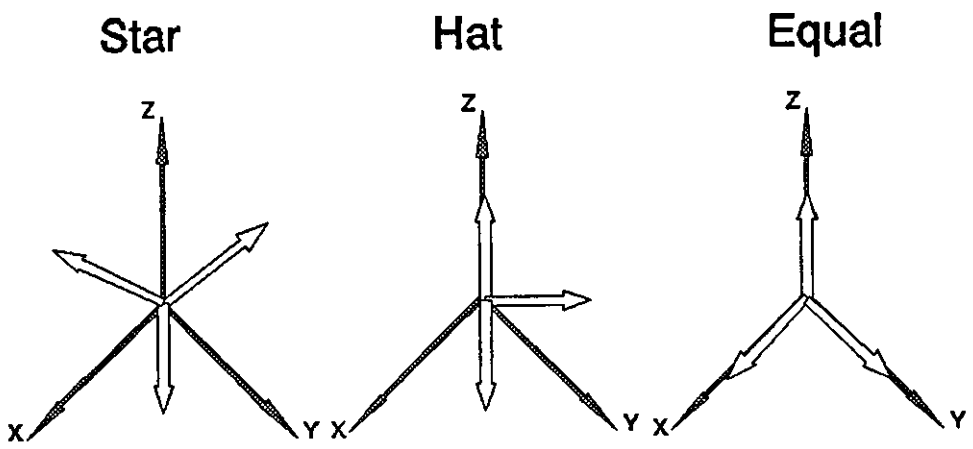


Figure 6.14 PRIMITIVES.

relationship exists between two parts. Figure 6.15 illustrates how a FIXTURE between a Screw and Hole can be described using the simultaneous relationships {Hat(hole[1],screw[1]) ; Hat(hole[2],screw[2])}.

Assembly Descriptions

This section describes how the representation can be used to determine the final position of a workpiece from a declarative description of the assembly. It discusses previous approaches, and then goes on to describe the approach taken in Plethora and the KS which implements it.

Previous Work

The work falls into two camps, one is exemplified by the robot programming language RAPT[21]. This is concerned with the specification of implicit motion and

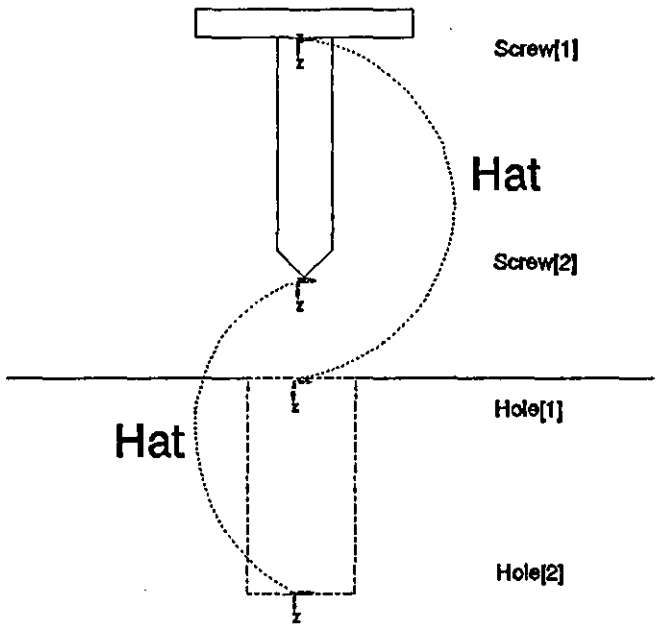


Figure 6.15 Screw Fixture.

leaves the assembly order to the programmer. The other [21,22] concentrates upon deriving assembly sequences.

RAPT is a task orientated language which accepts descriptions of spatial relationships such as "*against(bottom_ of_block, table_top)*". It interprets the descriptions in terms of the constraints which they imply, and then it solves these to produce a final position of the part. The relationships are expanded and analysed by a rule based program, which uses re-write rules to solve the constraints. Initially, the constraints were solved algebraically[10]. However, this was found to be slow and could not be guaranteed to converge. A restriction in the class of problem and a modification to the inference mechanism (so that it produced groups of simultaneous equations with known numerical solutions) increased the efficiency of RAPT, but not by enough for it to be completed within the typical critical times of the work cell.

The approach taken in [23] produces a complete representation of all possible assembly sequences for a given assembly in the form of an AND/OR graph (see [24] for a comparison with other forms of representation). It then [25] searches through

the graph using a AO* search to extract a tree which represents a single assembly sequence. The initial creation of the graph relies upon the availability of two predicates *Connected-stable*, which determines whether the resulting assembly step would be stable, and *Task-Feasibility*, which determines whether it is possible to make the assembly step based upon geometric constraints (e.g. is the path obstructed). Similarly, it is necessary to provide cost functions for the AO* searches and for the comparison of the possible assembly sequences. These latter cost functions it was said could be based upon part entropy, a measure of assembly complexity, parallelism in construction or multiple paths. However, due to the exhaustive nature of the graph generation and the complexity of the predicates, this is only ever likely to be an off-line process.

Assembly Descriptions in Plethora

The approach used within Plethora is not complete, but is extensible and efficient. The method expands a hierarchical description of the assembly (using the relational plane of the frame network) in to it's necessary *PRIMITIVE* constraints (i.e. those which must hold between coordinate frames attached to workpieces in the completed assembly). Choosing an initial workpiece about which the construction is to be built, the list of constraints is searched to find groups of constraints related to a single part, which when solved specify all its degrees of freedom (i.e. determine the position of that part in the assembly). Hence the techniques uses the necessary precondition, that a part must already exist within the assembly before another part, whose own stationary position relies upon it can be added, to order the series of solutions.

However, this initial ordering is not sufficient to be used as the sequence of assembly actions. It is still possible for the current order of solutions to allow the path of a workpiece to be obstructed by another, whose position had been previously solved (e.g. it is necessary to put all the contents of a box into the box before placing the lid on the box, otherwise the lid will obstruct the addition of the contents). Hence, further ordering constraints based upon the geometrical properties of the workpiece and the possibility of obstruction by previous solutions, is necessary.

This further ordering is achieved by modelling the approach path of the newly solved workpiece as it is added to the assembly (as described earlier). If a collision occurs with any workpiece already in the construction the workpiece causing the collision ought to be added after the workpiece test. Thus the initial order is re-ordered by the results of these tests, to produce the final precedence graph for the construction.

Unlike [23] this is not based upon the decomposition of the completed assembly but upon it's construction, and so has a different constraint unavailable to it when ordering the assembly steps. (i.e. Stationary relationships can only be solved if arguments of that relationship already have their position defined within the assembly). Hence, if initially only one *WORKPIECE* has a defined position the assembly will be built around it.

What follows is a description of the details of a KS which produces the precedence graph.

Solve-Construction

A KS, called *Solve-Construction,* exists which implements the method above and can be used to produce assembly sequences. It can also be used to solve the spatial relationships specified in terms of *FIXTUREs* on the blackboard. This allows the goals on the blackboard to be specified in terms of a relationship with respect to the object and not in absolute coordinates. Hence, the position of an object (with which there will be an associated uncertainty) can be determined at run time by sensory information and only then the equations solved to produce the absolute coordinates. This allows the system to reason with up to date information rather than using less reliable predicted values.

Solve-Construction operates in two stages, initially it stacks the goal *FIXTURES* and *SUB-ASSEMBLIES* and then recursively removes the expressions from the top of stack and replaces them with their component descriptions derived from the relational network. When *Solve-Construction* encounters a *PRIMITIVE* relationship on top of the stack it removes it and places it in the Primitive table. During each

expansion *Solve-Construction* substitutes the actual parameters from the original *ASSEMBLY* description, for the dummy *FIXTURE* parameters.

Once the stack has been emptied it then begins to search the primitive table for stationary relationships between the *BODYs*, and solves for the position of a particular workpiece. It then marks that workpiece's position as *KNOWN* and copies the equation (e.g. Equal, 3 Stars etc) and bodies concerned with that stationary relationship into a Stationary relationship table. This process continues until all workpieces, which may form stationary relationships within the current state of the assembly, are *KNOWN*.

The stationary relationships determined during a cycle are linked via *BEFORE/AFTER* relationships to those stationary relationships determined on the previous cycle, producing the initial partial ordering.

At this stage the partial ordering takes no account of possible collisions during assembly. *Solve-Construction* now tests for collisions between the members of the group of stationary relationships found during this cycle. It assumes that all members apart from that under test are already part of the assembly. It also assumes that the approach path of the part under test, is directly above the *KNOWN* position and begins at the maximum height of the work space. Using the results of the collision test, it then re-orders the relationships between the group and the part under test. It does this by adding additional *BEFORE/AFTER* links between the stationary relationships. (Note: if during a test it is decided that the original workpiece, about which the assembly is to be built, must come after another. This indicates that the *KNOWN* position cannot be reached from above and the process is abandoned).

Any relationships not used by this stage are used to check the consistency of the overall assembly description, by evaluating the relationships using the *KNOWN* position of the parts. (Note: most assembly descriptions formed this way are over specified).

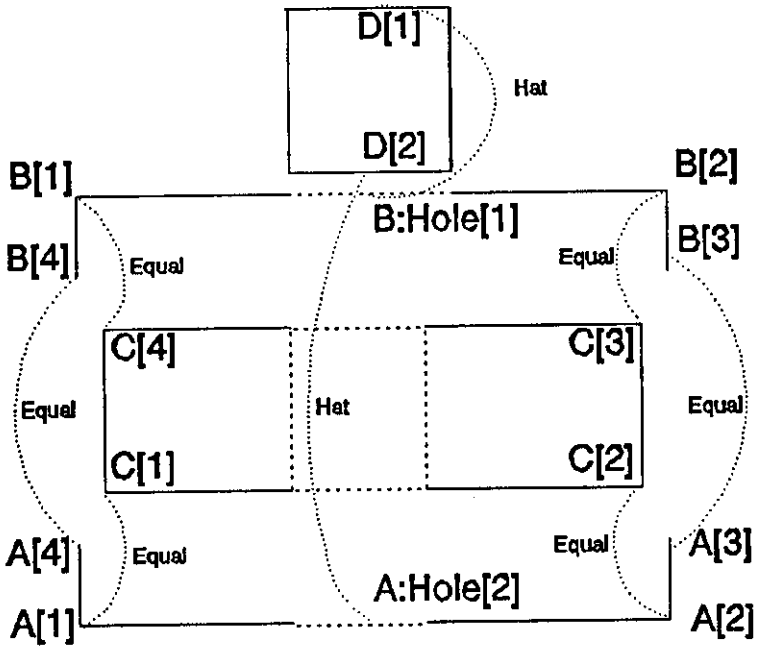


Figure 6.16 Example Assembly.

An example of the process is shown in figure 6.16 and 6.17. Figure 6.16 shows the cross section of the assembly where the two halves of a box each with a hole, are to enclose C. C also has a hole to allow pillar D to be inserted through B,C

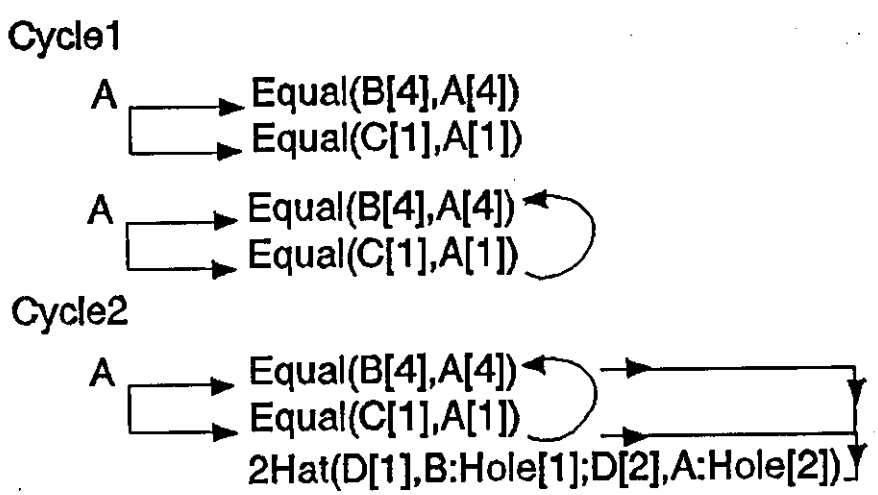


Figure 6.17 Resulting Precedence Graph.

and A. The dotted lines indicate the *PRIMITIVE* relationships between the frames attached to the workpieces in the final assembly. Figure 6.17 shows the initial *After* ordering produced by the solution order of the constraints (indicated by straight lines) and the addition constraints added by the collision test (indicated by arcs).

Summary

GeoMod provides a new means by which declarative descriptions of an assembly can be interpreted as a partially ordered set of homogeneous transformation operations on coordinate frames attached to workpieces. It assumes a workpiece about which the assembly is to be built, but in comparison to exhaustive schemes (such as [23]) is relatively efficient. In so doing, GeoMod partially fulfils requirement *ct1* and completely fulfils *kn2*.

GeoMod achieves its representation and inferential adequacy in two ways. Firstly, where ever possible it maintains explicit representations of the geometric properties and relationships, and so trades memory size for speed of access. Secondly, the hierarchical nature of its data structures is used to reduce the need for exhaustive search techniques.

In common with the previous work described in this chapter, GeoMod does not model the uncertainty in position associated with an object (as performed during the off-line operation of SPAR), but uses a single position which corresponds to the highest *Confidence* measurement. This is done because the propagation of the affects of uncertainty through GeoMod would dramatically increase the complexity of it's basic operations.

Whichever approach is taken to the modelling of uncertainty, there will eventually be a limit placed on the modelling which can be under taken within the critical times required of GeoMod. Hence, the approach taken within Plethora is to model only those elements which can be modelled efficiently, and reduce the

requirements made of the predictive element of its representation. To compensate for this, emphasis is placed upon Plethora's ability to reason during the execution of the plan using sensory data.

Although feature/value pairs could easily have been added to GeoMod, it is assumed that the KS which is responsible for matching (and/or generating) the features should maintain them itself. These could be linked to GeoMod by the name of the frame its uses to represent the features, so forming a distributed feature data base. This has two advantages, it maintains encapsulation and will allow matching to take place in parallel.

References

[1] Koshikawa,K. Shirai,Y.
"A 3-d Modeller for Vision Research",ICAR 85,p185-190.

[2] Boissant,J.D.
"An Automatic Solid Modeller for Robotic Applications", Advanced Software in Robotics, Elsevier Science Publishers, p65-72, 80.

[3] De Pennington,A. Bralila,M.
"Geometric Modelling: A Contribution Towards Intelligent Robots",13th ISIR/Robots 7 Conference, April 17-21, 1983.

[4] Wesley,M.A. Lozano-Perez,T. Lieberman,L.I. Lavin,M.A. Grossman,D.D.
"A Geometric Modelling System For Automated Mechanical Assembly", IBM J. Res. Devel., vol.24, no.1, p64-74 , 80.

[5] Pertin-Troccaz,J.
"S.M.G.R : A Geometric and Relational Modeller for Robotic Applications", ICAR 85, p23-31.

[6] Lumia,R.
"Representing solids for a Real Time Robot Sensory System", Software for Discrete Manufacturing, Elsevier Science Publishers, IFIP 86, p393-402.

[7] Kak,A.C Vayda,A.J. Cromwell,R.L Kim,W.Y. Chen,C.H.
"Knowledge Based Robotics", International Journal of Production Research, vol.26, no.5, p707-734, 88.

[8] Ambler,A.P
"Robotics and Solid Modelling: A Discussion of the Requirements", Advanced Software in Robotics, Elsevier Science Publishers, p361-367, 80.

[9] Rich,E.
Artificial Intelligence, Mc Graw-Hill, ISBN-0-07-052261, 83.

[10] Requicha,A.A.G.
"Representations for Rigid Solids: Theory,Method and Systems", Computing Surveys, vol.12, no.4, Dec.80.

[11] Tilove,R.B.
"Set Membership Classification: A Unified Approach To Geometric Intersection Problems", IEEE Trans. Comp., C-29, no.10, p874-833, 80.

[12] Brooks,A.R.
"Model-based Computer Vision", UMI Research Press, ISBN 0-8357-1526-4, 81.

[13] TIPS
A commercial package available from Institute for Precision Engineering, Hokkaido University, Sapporo, Japan.

[14] Hearn,D. Baker,M.P.
"Computer Graphics", Prentice-Hall, ISBN 0-13-165598-1, 86.

[15] Craig,J.
"Introduction to Robotics: Mechanics and Control, Addison-Wesley, ISBN-0-201-10326-5, 89.

[16] Boyse,J.W.
"Interference Detection Among Solids and Surfaces", Comm. ACM, vol.22, no.1, p3-9, 79.

[17] Pennington,A. Balia,M.
"Geometric Modelling: A Contribution Towards Intelligent Robots", 13th ISIR, p1-19, 83.

[18] Cameron,S.
"A Study of Clash Detection Problem in Robotics", IEEE Int. Conf. Robotics and Automation, p488-493, 85.

[19] Cameron,S.
"Modelling of Solids in Motion", PhD Thesis, Dept of Artificial Intelligence, University of Edinburgh.

[20] Takase,K. Paul,R.P. Berg,E.J.
"A Structured Approach to Robot Programming and Teaching", IEEE Trans. Sys. Man and Cyber., vol.smc-11, no.4, p275-289, 81.

[21] Popplestone,R.J Ambler,A.P. Bellos,I.M.
"An Interpreter for a Language for Describing Assemblies", Artificial Intelligence, no.14, p79-107, 80

[22] Corpner,D.F. Ambler,A.P. Popplestone,R.J.
"Reasoning about the Spatial relationships Derived from a Rapt Program for Describing Assembly by Robot", 8th IJCAI, p542-4, 83.

[23] Homem de Mell,L.S. Saunderson,A.C.
"AND/OR Graph Representation of Assembly Plans",IEEE Trans. Robotic and Automation, vol.6, no.2, p188-199, April 90.

[24] Homem de Mell,L.S. Saunderson,A.C.
"Representation of Mechanical Assembly Sequences", IEEE Trans. Robotic and Automation, vol.7, no.2, p211-288, April 91.

[25] Homem de Mell,L.S. Saunderson,A.C.
"A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences", IEEE Trans. Robotic and Automation, vol.7, no.2, p288-241, April 91.

Chapter 7
Object Recognition

This chapter describes the results of a new approach to real time object recognition. It was developed with the use of Plethora and is based upon 3D moment invariants (The initial results were reported in [1], found in Appendix D).

The method uses a tuplet generated from taught examples. The tuplet, consisting of a pseudo inertial tensor and volume, is used to identify and determine the orientation of single objects.

The chapter begins by describing the advantages of 3D object recognition. This is followed by a description of a parallel implementation [2] (see Appendix E) of the sequential algorithm [3], which is used to derive the volumetric representation of a scene. It then introduces 3D moment invariants and explains their use in classifying volumetric representations of objects. Finally, the experimental results are discussed in terms of, the assumptions made during the generation process and the problems of camera calibration.

Overview

Object recognition is of particular importance in assembly operations, as it provides a means of identifying workpieces and determining their position and orientation. If a suitable model-based representation is employed it can also serve to verify the assembly actions, by modelling these changes in the workpiece and matching the predicted model against the scene. Before truly automated assembly is possible, it is essential to have a robust representational model and technique for determining the orientation and position of 3D objects, for a large class of industrial parts [4].

Introduction

Previous work [5] can be categorised as 2D recognition, where a single image is used, 2.5D where some depth information is available, such as in stereo vision[6], and 3D, where a number of views are combined to produce view-independent volumetric representations of the scene. The latter is often the preferred form for object identification[5], since it results in a single object centred representation, in contrast to the 2D systems in which a number of views must be represented. The design of 2D systems is further complicated, since the means by which the views are chosen and how they are to be represented, is not straightforward. The practical limitations placed upon these choices constrain the number of view points from which the object is recognisable.

A 3D system not only reduces the difficulty associated with the interpretation of the one-to-many transformation involved in a 2D system, but it also overcomes the time constraints that are associated with the correlation of images in stereo vision systems. The choice of three orthogonal views provides a compromise between the simplicity of the hardware required and computational efficiency.

Previous Work

Much of the model based recognition work [7] uses a geometric model of the object to generate a 2D representation of the scene from a given view point. It then determines a set of features, and finally tries to match these against the feature vector of the image (e.g.[4])

A different approach [8] generates volumetric models from multiple views and uses a boundary representation (based upon bicubic parameter patches) to represent the object and scene. It then heuristically searches for overlapping segments to produce a description of the connected patches. This technique is applied iteratively until the whole scene has been covered. More recently, a relaxation technique based upon bipartite matching[9] has been described. The technique can be used to prune the search space when matching surfaces and has polynomial time complexity.

Surface orientated approaches, which by their nature utilise local feature recognition, tend to be computationally expensive. At present, these approaches have no obvious application in real time and/or do not yield features for suitably efficient object recognition. Although, they may be essential when analysing clustered or occluded scenes, where local features are the only means of recognition.

In order to produce an efficient 3D model based recognition system, we must choose a compatible form of representation and method of identification. The approach described here attempts to do this by combining an efficient means of generating the volumetric representation and utilising a feature representation, which can be simply manipulated to provide a useful set of compact global features. The feature representation can be derived from both predicted geometric and sensory data.

Methodology

The calculation of the tuplet takes place in two stages. Firstly, the generation of the volumetric representation of the object in terms of a number of non-overlapping rectangular parallelepipeds (RPs) and secondly, the derivation of the tuplet from that set of RP's. It has been shown that RP coding is more efficient in terms of storage requirements and execution time required for its generation than present similar volumetric representations [3]. It provides simple and efficient intersection and merging operators, and can be readily inverted to produce a representation of the unoccupied space about the object (the process is described in Chapter 8). These characteristics make RP coding an especially attractive candidate for spatial representation in robotic applications.

Generation of the Volumetric Model

The RP's which represent the object are coded as in figure 7.1. They are derived in real time using a modified version of the method found in [3]. This method is implemented in parallel on a multi-transputer system and sequentially using of the vision server in Plethora. The parallel version takes $\sim 1/2$ second to generate the volumetric representation.

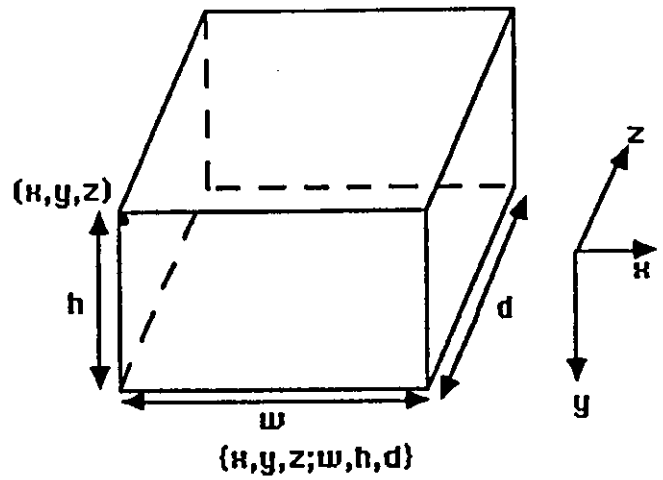


Figure 7.1 The coding scheme used to represent a RP.

The system uses three orthogonal views of the object, which are assumed to represent parallel projections in each of the viewing directions. These views are coded as rectangles, which are then swept in their respective viewing directions to form rectangular parallelepipeds (RPs). The intersection and merging of these RPs produces the final list of RPs from which the tuplet is derived. The essential elements of this process are shown in figure 7.2.

What follows is a more detailed description of the process and the parallel projection assumption on which it relies upon.

Outline of the Algorithm

The necessary stages of the algorithm found in [3] are outlined below in steps (a) to (e).

a) **Thresholding**

The three grey scale images were thresholded to produce binary images, using a variable thresholder based upon the simple image statistic technique [10].

b) **Run Length Coding**

For each horizontal line of each image, the start and run length of all the black segments were found.

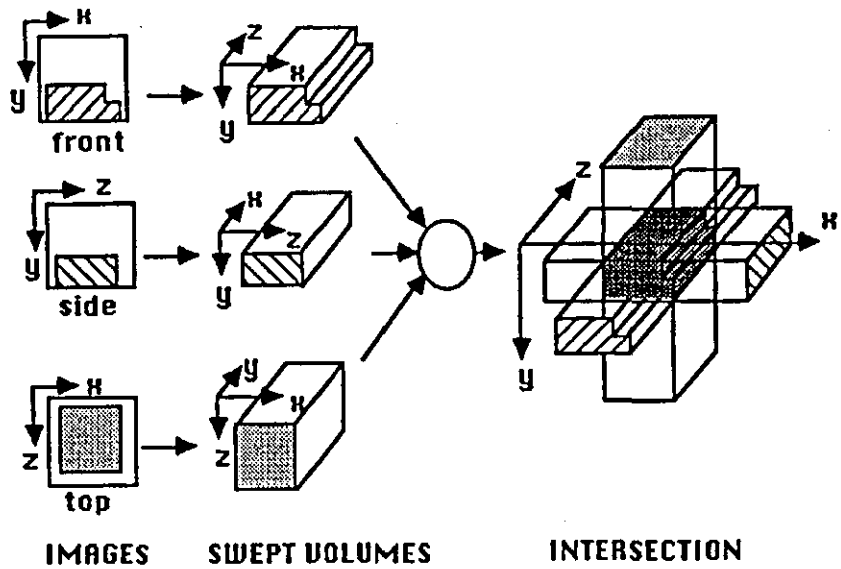


Figure 7.2 An overview of the method.

c) **Two Dimensional Rectangular Coding**

From the run length codes of adjacent lines, rectangles of maximum area were grown to produce a 2D rectangular coding of each image. Figure 7.3 shows the method by which rectangles are grown and figure 7.4 shows a typical result of this process.

d) **Sweeping**

3D RPs were generated from each rectangle by sweeping the rectangles along their respective viewing directions.

e) **Intersection and Merging**

The intersection of the 3 sets of RPs produced an intermediate set of RPs. This process was completed in two stages. Initially the top and front views were intersected and then each result of this intersection was then intersected with the side view (see figure 7.2). As this process can produce pairs of RPs with a common face, this intermediate set was then checked to see whether any of the parallelepipeds could be merged and so remove redundancy from the final representation.

The initial experiments used compressed 128x128x8 bit grey scale images of the object. However, even with the intrinsic simplicity of RP representation, a sequential implementation using the vision server did not run fast enough for use in an industrial

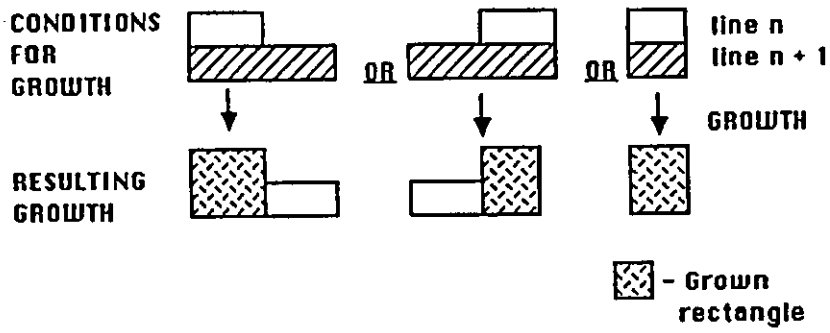


Figure 7.3 The condition required for the growth of a rectangle.

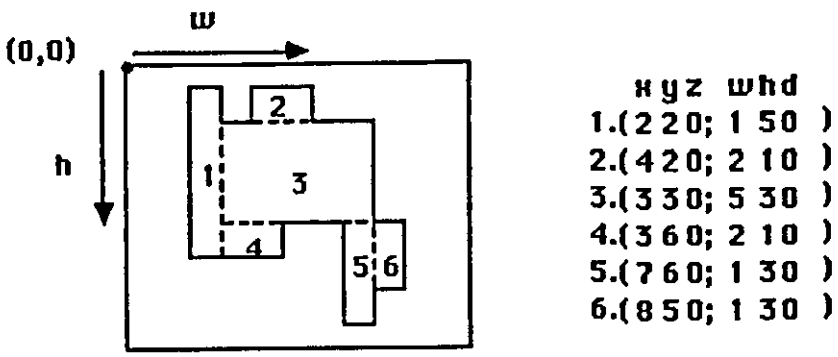


Figure 7.4 The results of two dimensional RP growth.

work cell (it took ~25 sec to complete the process which included the initial convolutions required for filtering the image). Hence, a modified parallel version was developed using *Streams* and a *Company* of *Players*, to make use of the technique's inherent geometric parallelism. A skeleton version of the code run on 3 PC's is shown below in figure 7.5.

This modified version took ~20 seconds to complete. However, most of the time was spent in communication and the potential speed-up was hidden by the slow response of the prototype network (This was determined as a result of timing

```
Rpx:
{
new grey-scale-image $x  Assert            ... Take a picture
                &x '' Smooth               ... 3x3 averaging filter
                &x '' rats                 ... Calculate and perform thresholding
                &x '' run-code $y Assert   ... Produce Run length code version of
                                               binary image
                &y  Generate-Stream
}
Vision-Server Begin-Stream        ... Get the vision server to produce the
                                      Run length codes
' Grow-2D-Rectangles Intersect-XY Map-Stream ... Grow Rectangle and pass them on
                                                 to Intersect-XY
Vision-Server End-Stream


Intersect-XY:
        Rpx Rpy Intersect Intersect-XYZ Merge-Stream

Intersect-XYZ:
        Rpz Intersect-XY Intersect Merge-RPs Merge-Stream
```

Figure 7.5 Skeleton code of the parallel implementation.

calculations made by the message passing system). Finally, the system was implemented on a dedicated network of 3 transputers with a similar form of partition and mapping (see figure 7.6 and Appendix E for a copy of the paper reporting the detailed design of the network). The generation of RPs, using the Transputer

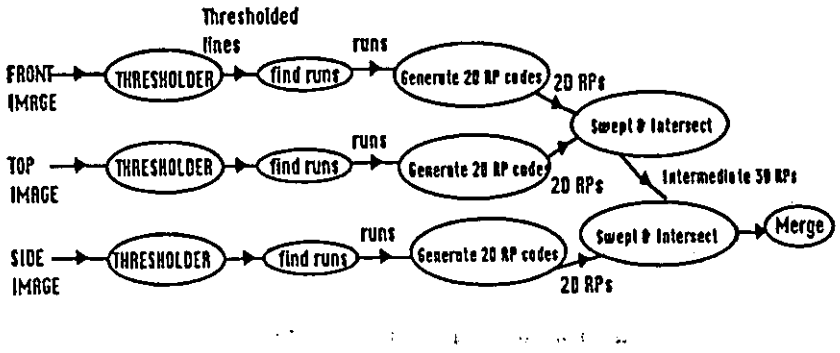


Figure 7.6 The overall partition of the system.

network, took between 0.25 and 0.5 seconds, well within the critical times of most processes within an assembly work cell. Figures 7.7 and 7.8 illustrate the result of the technique on two of the more complex shapes.

Figure 7.8 Illustrative examples of the coding scheme for cup and pencil sharpener.

Figure 7.7 Photograph of cup and pencil sharpener.

The Parallel Projection Approximation

The implicit approximation made by this technique can be best seen in figure 7.9. In figure 7.9 there is a single ideal camera with focal length f, a line at x2

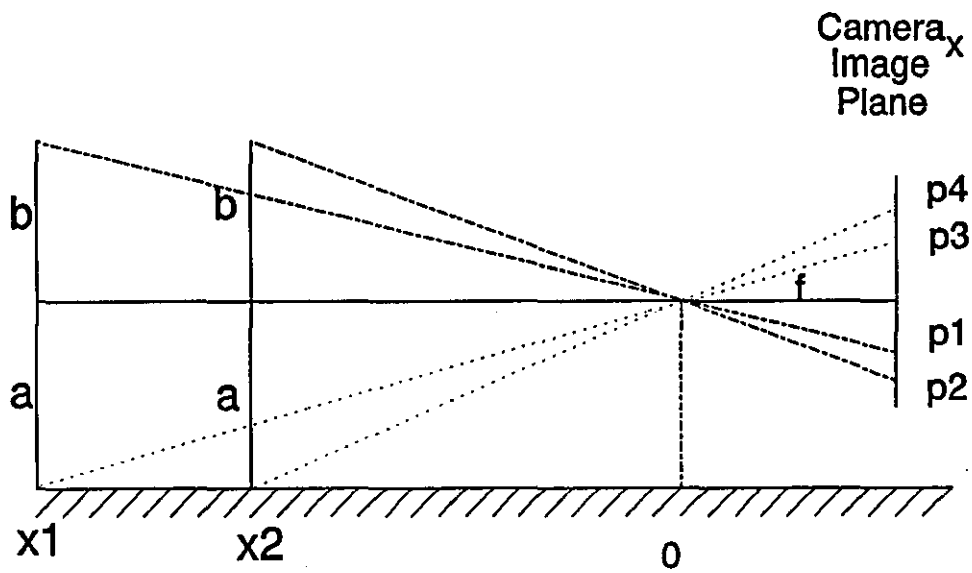


Figure 7.9 Parallel projection approximation.

representing the front of a box in view and x1 representing the plane used to calibrate the camera pixel separation. Because the front of the box is closer to the camera it will appear larger than it ought in the image plane. Hence the generation process will always overestimate the size of the object by the ratio x1/x2. After the intersection stages, the RPs will have the minimum of the two commonly perceived dimensions, resulting in an overall expression for the volume, as below.

Using similar triangles,

$$p1/f = b/x1 \; ; \; p2/f = b/x2 \; ; \; p3/f = a/x1 \; ; \; p4/f = a/x2$$

Defining the error in length relative to the correct length for camera$_x$ to be ${}^{y}\epsilon_z$,

$${}^{y}\epsilon_z = (p3+p1)/(p4+p2) = x1/x2$$

and similarly for ${}^{x}\epsilon_z$ *etc.*

However, the error in the dimension after intersection will be,

$$\epsilon_z = min(^x\epsilon_z, ^y\epsilon_z)$$

$$\epsilon_x = min(^y\epsilon_x, ^z\epsilon_x)$$

$$\epsilon_z = min(^x\epsilon_y, ^z\epsilon_y)$$

giving a final error in volume of,

$$\epsilon_v = \epsilon_x . \epsilon_y . \epsilon_z$$

The expression can be used to scale the estimate of the true volume. However, it will still be an upper bound, as the values on which it is based, are approximations to the actual dimensions of the box.

Generating the Centralised Moment Matrix

If we assume that RPs have a uniform density of 1, then the sum of the RP volumes will result in a pseudo mass for the object. If the individual pseudo masses are taken to act at the mid-point of the RPs, we can generate moments and centralised moments of order p+q+r as below.

$$M_{pqr} = \sum_{i=1}^{k} (x_i^p . y_i^q . z_i^r . vol_i)$$

and the centralised moments

$$\mu_{pqr} = \sum_{i=1}^{k} ((x_i - \bar{x})^p . (y_i - \bar{y})^q . (z_i - \bar{z})^r . vol_i))$$

where

[x,y,z] are the coordinates of the midpoint of the individual RP volumes.
K is the number of RP's.
vol_i is the volume of the ith RP.

This gives rise to the matrix $\underline{U}$, of second order centralised moments, required for the tuplet representation and which is used to derive the 3D moment invariants [7] and principal axes.

$$\underline{U} = \begin{bmatrix} \mu_{200} & \mu_{110} & \mu_{101} \\ \mu_{110} & \mu_{020} & \mu_{011} \\ \mu_{101} & \mu_{011} & \mu_{002} \end{bmatrix}$$

The volume is simply found by summing all the individual RP volumes used to represent the object.

Moment Invariants

Sadjadi and Hall [11] have generalised the results of 2D moment invariants [12] by linking 3D moments to ternary quantics and produced a set of 3D moment invariants. These are invariant under scale, orientation and translation and are shown below,

$$\frac{J_{1\mu}^2}{J_{2\mu}} \text{ and } \frac{\Delta_{2\mu}}{J_{1\mu}^3}$$

where

$$J_{1\mu} = \mu_{200} + \mu_{020} + \mu_{002}$$

$$J_{2\mu} = \mu_{020}\cdot\mu_{002} - \mu_{011}^2 + \mu_{200}\cdot\mu_{002} - \mu_{101}^2 + \mu_{200}\cdot\mu_{020} - \mu_{110}^2$$

$$\Delta_{2\mu} = \det [\underline{U}]$$

The invariants can be readily obtained from $\underline{U}$ and are used as features when identifying the object.

Principal Axes

Given that the object's reference frame origin is at the centre of mass, when this frame is aligned with the principal axes the matrix $\underline{U}$ becomes diagonal, ($\underline{U}^*$). The elements of $\underline{U}^*$ correspond to the principal moments of the object. The direction of the principal axes are given by the eigenvalues of the matrix $\underline{U}$. Hence, the rotational matrix ($\underline{R}$) which is required to align the object with its principal axes can be determined from,

$$\underline{R} = \underline{U}^* . \underline{U}^{-1}$$

Where $\underline{U}$ represents the objects present position and $\underline{R}$ represents the orientation of the object with respect to principal axes.

Results

The initial experimentation performs two functions. Firstly, to verify the operation of the system and secondly, to determine the effectiveness of the tuplet for use in object recognition.

Volume/(voxels)	$\Delta_{2\mu}/J_1^3{}_{\mu}$	$J_1^2{}_{\mu}/J_{2\mu}$
2369	3.23×10^{-3}	3.9
2492	2.75×10^{-3}	3.6
2390	2.5×10^{-3}	3.6
2394	3.89×10^{-3}	4.0
2296	2.08×10^{-3}	3.3
2420	2.94×10^{-3}	3.6

Table 1. Feature values for 6 positions of the ball.

Volume/ (Voxels)	$\Delta_{2\mu}/J_1^3{}_{\mu}$	$J_1^2{}_{\mu}/J_{2\mu}$	Measured Rotation	Calculated Rotation
5054	3.11×10^{-2}	3.5	0°	5°
5881	3.88×10^{-2}	3.78	45°	53°
5039	4.80×10^{-2}	3.48	90°	93°
5095	4.01×10^{-2}	3.01	142°	142°

Table 2. Feature values for the first 4 rotations of the box.

Verification

Two experiments were used to verify the operation of the system. Firstly, a white ball was placed at differing positions within the working volume of the system (this consisted of a 15 cm cube at the intersection of the camera's viewing space) and the tuplets calculated. Secondly, a white box, placed at the centre of the working volume, was rotated at multiples of 45°, and the tuplets calculated. Together, these were used to independently test the system's invariance under translation and rotation. The results are shown in tables 1 and 2.

Classification

The effectiveness of the tuplet as a means of object recognition was illustrated by comparing a set of chess pieces and set of randomly selected assembly parts. The tuplet was used to form a three element feature space, consisting of the volume and the two moment invariants. The chess pieces were of similar shape and had no holes, while the assembly pieces were irregular and had both holes and reflective surfaces. These sets were used to separately evaluate, the ability of the system to distinguish between similar objects, and to cope with the problems associated with typical workpieces. It was found that the chess and assembly pieces could be identified using a simple minimum distance classifier. However, it was also noticed that at certain orientations that the holes in the assembly parts were hidden from the view of all three cameras, and so were not constructed in the volumetric representation. This gave rise to a greater variation in the position of the workpieces within feature space than occurred with the chess pieces. Although, they still remained linearly separable. Similar effects due to variations in the illumination resulted from bright spots on reflective surfaces.

Discussion

In general, the variation of the feature vectors for a single object were due to three factors. Firstly, the loss of gross features which were hidden from the cameras. Secondly, the effects of illumination and finally, a combination of the effects of quantisation and camera axes misalignment.

Obscured features are a common problem in most vision systems. The problem is often overcome by associating a number of areas in feature space with a particular object (i.e. one for each of the systems blind spots). These could be represented by Plethora as *EQU's* of a *INDV* in GeoMod.

Illumination

The lack of control over the illumination of the object usually results in a loss of local detail. However, since the method of classification is based upon an analysis of global features, and distortions due to illumination must be correlated in all three images if they are to appear in the final volumetric representation. The technique tends to be tolerant of the effect of minor variations for the limited range of objects tested.

Calibration

During experimentation it was found that the system is particularly sensitive to misalignment of camera axes and this was aggravated by the low image resolution used in the initial experiments. A number of calibration techniques were investigated including [13]. However, it was observed that whichever technique was employed they resulted in large parameter ranges (the uncertainty in the position of the origin of the focal axis was ~ 8% of full frame resolution, using [13]). These fluctuations have been observed in [14] and attributed to timing mismatch between the camera and frame grabber hardware. In this case the cameras were 3 CCD Punix TM-760 with 756x581 square pixels, the frame grabber was a Data Translation DT2851 and was connected to the cameras via a standard 1v Composite video interface. The images were initially digitised as 512x512x8 bits.

Eventually, a very rough calibration procedure was used which provided similar accuracy to the more sophisticated technique in the presence of the jitter. A white 15 cm frame work cube was placed at the centre of the viewing volume and the cameras focused until the face of the cube filled the image. The relative displacement between the camera planes was measured using the orthogonal views of a white ball. The centroid of the ball was used as a reference between adjacent image planes and the displacements required to align the reference coordinates, were used as offsets between the individual image plane coordinates.

The error in calibration overshadowed the variation in the feature vectors due to the other effects. Further experiments with images of 256x256x8 bits, showed an amelioration in the problem and improved the correlation between the measured and calculated rotations. However, this was not sufficient to improve on the volume measurements.

Conclusions

The initial experiments illustrate the use of Plethora as a proto-typing tool and can be used to develop new techniques. The technique was used to successfully recognise single objects from a given set and less successfully determine their orientations. The time taken to perform this process took between 0.25 and 0.5 seconds (Note: This could be further reduced if T800's were used rather than the original T414's) within most work cell critical times.

Recently, new 3D invariants based on second and third order complex moments have been discovered [15] (12 in all) and they allow a unique determination of the principal axes. Extending the system to include these extra features would increase the robustness of the technique at little extra computational cost.

However, it is apparent that the jitter must be removed before any further development can take place. The most obvious method would be to remove the processes of generation and decoding the composite video signal, by providing a direct serial interface between the CCD chip and processor memory. This would also have a useful side effect, in that the full resolution of the camera could now be used at increased frame update rates.

References

[1] Sillitoe,I.P.W Edwards,J.
"A Multi-View Robotic Vision System for Efficient Object Recognition", Proc. Int. Conf Systems Science, Wroclaw, Poland 89.

[2] Sillitoe,I.P.W Edwards,J. Falkner,A.H.
"The Design of a Real Time Three Dimensional Vision System for Object Identification", Proc. 12th Occam User Group, p198-205, April, 90.

[3] Kim, Y.C. Aggarwal, J.K.
"Rectangular Parallelepiped coding: A volumetric representation of three dimensional objects", IEEE Journal of Robotics and Automation, vol.7, no. 6, Nov. 83.

[4] Bhanu,B. Henderson,T.
"CAGD Based 3-D Vision", Proc. IEEE Int. Conf. on Robotics and Control, p411-417 March, 1985.

[5] Marr, D.
"Representing Visual Information", Computer Vision Systems, A.R. Hanson and E.M. Rieseman, Academic Press, Orlando, F1. p61-88.

[6] Wildes,R.P.
"Direct Recovery of Three Dimensional Scene Geometry from Binocular Stereo Disparity", IEEE PAMI, vol.13, no.8, p761-774, August 91.

[7] Chin, R.T., Dyer, C.R.
"Model Based Recognition in Robot Vision", Computing Surveys, p67-108, March 86.

[8] Potmesil, M.
"Generating Models of Solid Objects by Matching 3D Surface Segments", 8th IJCAI, p1089-1093, August 83.

[9] Kim,W.Y. Kak,A.C.
"3D Object Recognition Using Bipartite Matching Embedded in Discrete Relaxation", IEEE PAMI vol.13, no.3, p244-251, March 91.

[10] Kittler, J. and Illingworth, J.
"Threshold Selection Based upon a Simple Image Statistic", Computer Vision, Graphics and Image Processing, vol. 30, p125-147, 85.

[11] Sadjadi, F.A. Hall
"Three Dimensional Moment Invariants", IEEE PAMI-2 no.2, p127-135, March 80.

[12] Hu,M.K.
"Visual Pattern Recognition by Moment Invariants", IEEE Trans. Information Theory, vol.8, p170-187, 62.

[13] Liu,Y. Huang,T.S. Faugeraus,O.D.
"Determination of Camera Location from 2D to 3D line and Point Correspondences", IEEE PAMI, vol.12, no.1, p28-37, March 90.

[14] Tsai,R.Y.
"A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology using Off-The-Shelf TV Cameras and Lenses", IEEE Journal of Robotics and Automation, vol Ra-3, no.4, p323-344, August 87.

[15] Lo,H.C Don,H,S
"3-D Moment Forms: Their Construction and Application to Object identification and Positioning", IEEE PAMI, vol.11, no.10, p1053-1064, October 89.

Chapter 8
Path Planning

An important component of automatic planning for robot assembly operations is the gross motion planner. This chapter describes an on-line gross motion planner for a restricted class of problems applicable to assembly operations. The approach has been implemented on two revolute manipulators, executing straight line paths. The relative speed of the algorithm makes it a desirable candidate for use in on-line error correction strategies. It also will be shown that the technique can either use sensory data from a vision system or predicted information from the geometric modeller, as the basis for its path generation.

Introduction

Motion with reorientation of the workpiece can be split into four phases, initial gross motion, reorientation of the workpiece and the final gross motion preceding the fine motion control[1]. Given sufficient open volume for manoeuvrability, reorientation may be accomplished with a special purpose planner[2]. However, due to the non-decomposability of the problem, algorithms that determine general collision-free paths for payload and manipulator in highly cluttered work spaces, are often impractical owing to their implementation complexity. (The general solution to the related Piano-Movers problem has been shown to be NP-complete [3]).

Previous work [1-2,4-9] has considered off-line solutions, using geometric modellers as their source of information about the work cell. Whilst at the same time noting that they provide "good" paths, most are not efficient enough for inclusion in on-line techniques, such as would be required for error correction strategies.

However, if we consider the assembly work cell sub-problem, practical assumptions can be made which allow the efficient implementation of a gross motion path planner, which still produces suitable paths. These assumptions include that the workpiece and gripper should be of comparable size and less so in shape, that there should be sufficient volume above the manipulator so as not to limit its manoeuvrability and that manipulations do not take place at the extremes of the manipulators work volume (i.e. All manipulations take place within it dextrous work space).

Given these assumptions a number of techniques [9,10,11] have been designed to provide "fast" collision free path planning. (Note: the term "fast" is difficult to define as little timing information is provided and there is no analysis of the complexity of the path to be planned. The term is justified by the order of complexity given for each of their component operations. However, without absolute timing information it is not possible to make comparisons).

The Hierarchical Orthogonal Space (HOS) approach [9] uses an Octree representation of the work space derived from boundary representations, in which each of the objects has been grown by the dimensions of the payload. This growing process allows the payload to be modelled as a point and simplifies the search through the Octree for a collision free path. The search is made in each of the 3 orthogonal 2D sub-spaces, this is shown to be more efficient than a single search through the representation of 3D space.

A similar approach is taken in [10], once again it uses an Octree but recursively decomposes the free space of the work space to find a collision free path. It models the payload as a "stick", constructed from a cylinder and two spheres, and checks for collisions when directing the decomposition process. It also mentions that a suitable Octree could be derived from a vision system using a technique described in [12].

The approach taken at NBS [11] models the swept path of the manipulator using primitive volumes and searches through an Octree representing the work space. The Octree is derived from a geometric modeller. The search technique uses a combination of methods (hypothesis and test, hill climbing and A*) at different points in the search to increase its efficiency. The time consuming elements of these methods is the initial transformation from boundary to Octree and the complexity in modelling rotations of the work piece with an Octree.

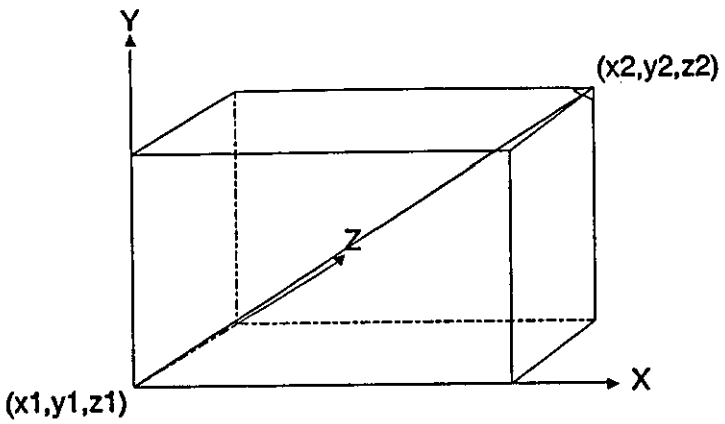
The approach taken here uses rectangular parallelepipeds as the basis for its representation of space. It plans the path through the free space around the objects, rather than using an interference operator and the object space itself.

Method

Free Space Representation

The work space and objects in a work cell can be modelled in terms of the space occupied by the objects or in terms of the free-space between them[1]. A free-space representation is used here, as it obviates the need for an interference operator (providing the advantages described in Chapter 6). Further, knowledge of the free-space generated by this method can be used by other modules concerned with aspects of on-line planning (i.e. determining where to place an object on the assembly table).

However, all free space schemes have the disadvantage, that after each movement the free space representation must be updated (This is also the case in [9,10,11]). So, ideally the free-space generation should be efficient and be readily updated from sensory information. A rectangular parallelepiped code(RP) fulfils both these criteria. RP representations can be determined from vision systems using multiple views (as described in Chapter 7), easily generated from geometric modellers and can be efficiently manipulated. The RP representation used in the path planner is shown in figure 8.1. It is chosen to maximise the efficiency of the operations within the path planner, however, it differs from that used in the vision system and so a simple conversion is necessary.



Rp = (x1,y1,z1;x2,y2,z2)

Figure 8.1 RP representation.

An operator, *###*, similar to the Sharp[3], (an operator used in the determination of prime implicant sets) is used to efficiently invert the object space to produce a free-space. The free space is represented as a set of maximal overlapping rectangular parallelepiped. A graphical representation of the operator is shown in figure 8.2.

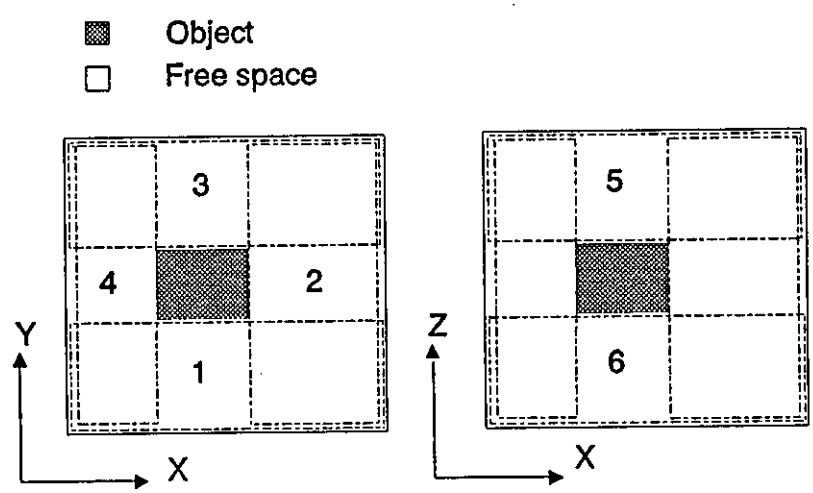


Figure 8.2 Graphical illustration of *###*.

$$FS = Un \; ### \; s$$

where *FS* is the set of overlapping RP's, representing free space. *Un* is the RP representing the work space. *s* is the RP representing the object

A ### B, performs the function (A ∩ (not B)) (i.e what belongs to A that does not exist in B). However, unlike Sharp, ### operates on continuous variables with a maximum of 3 degrees of freedom.

For a number of objects the FS is defined to be

$$FS = ((Un \; ### \; s1) \; ### \; s2) \; ....$$

Where $S = (s1, s2 ..... sn)$ is the set of all objects within the work space.

The path is determined in three stages. Firstly, the derivation of the FS using ### and a model of the objects in the world. Secondly, an initial path in two dimensions with nominal z coordinates is then found. Finally, this initial path has the nominal z coordinates of it's via points modified, in such a way as to avoid limb collisions (Note: Limb collisions are not considered in [9,10]).

Determination Of the Initial Path

Once the free-space has been determined, the maximal RP covering the start position, ST, and goal position, G, are found. A search is made through FS via overlapping RP's from ST to G. Only the RP's that would allow the passage of the payload are expanded. The payload is modelled as sphere of radius, Vr, this includes both the workpiece and gripper. This choice of representation makes the predicate for expansion efficient, but limits the forms of paths available to those which do not employ reorientation of the payload as a means enabling the payload to pass between objects.

The search is based upon a heuristically guided best first search. At present two equally weighted cost functions are used, the distance from the proposed RP to G, (using the mid-points of the RP's as point volumes) and the distance from the present position to the point volume of the proposed RP. The search is similar to an A^* search and the path is encoded within the parent links of the expanded nodes. However, the cost of testing for previously expanded nodes and modifying their cost fields (as is required in the A^* search), was found to outweigh the cost of maintaining

multiple paths to duplicate nodes and so no such tests are made during the search. The initial via points for the manipulator, are generated using the point volume coordinates of the intersecting RP's (figure 8.3). These then have their *z* coordinates modified to be

$$min(zmin+vr,\ zmin+zmax/2)$$

where *zmin* and *zmax* are the extent of the free space RP in the z direction (see figure 8.4). This reduces the need for unnecessary variations in the z direction, as a manipulator moves from RP to RP.

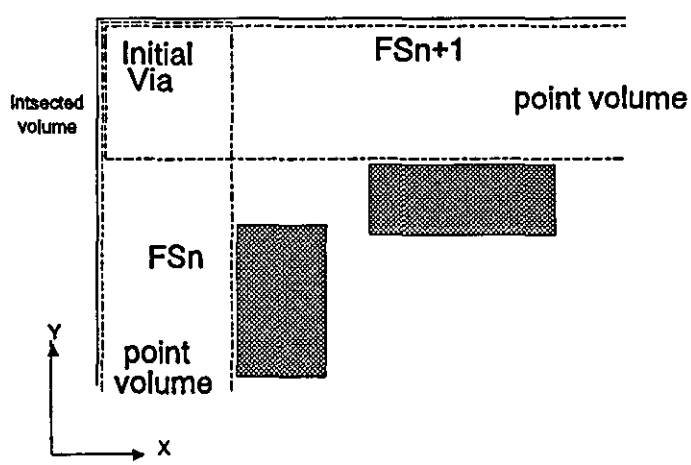


Figure 8.3 Via point determination.

Optimisation of the path length was considered and a simple technique implemented. The technique was a look-ahead strategy, which ensured the payload crossed into the next intersected volume at the optimal point to approach the next RP. However, when the cost of the extra manipulator movement was compared with the time required to perform the procedure, it was not considered necessary. Further, since the technique is to be supplied by data derived from sensory devices (with which there will be an associated uncertainty), the process of optimisation may not produce a better path in practice. It was also noted that a suitable choice of *Un*, by considering ST and G, produced a similar improvement in path length to that found when using the optimisation technique. A suitable technique was found to be, to bound the X-Y dimensions of *Un* by 3Vr about ST and G, this provided enough

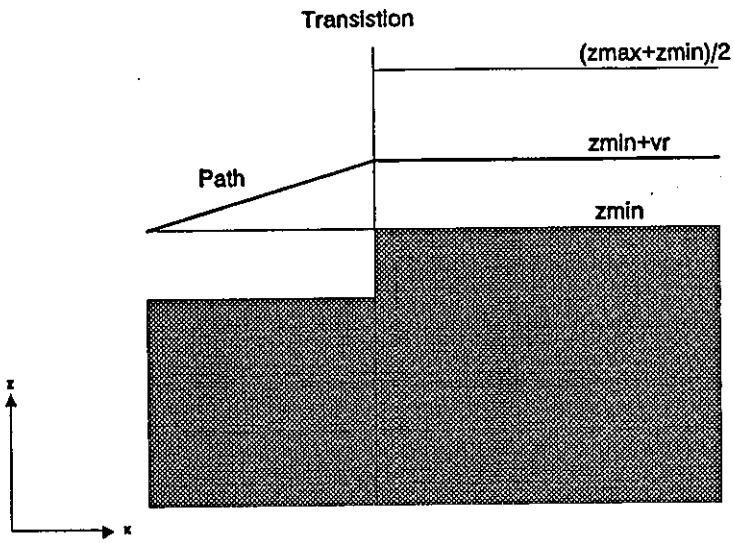


Figure 8.4 Modification of z component at Via point.

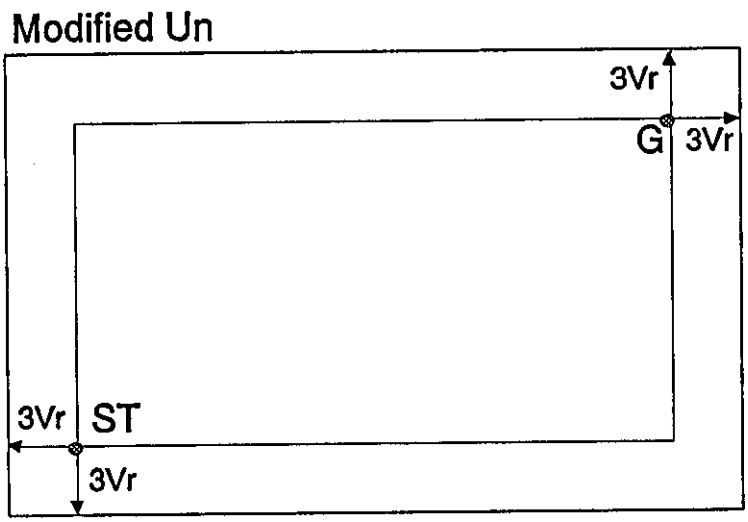


Figure 8.5 Modified Un.

manoeuvrability while restricting the operation of path planner to volume of interest (see figure 8.5).

Modification of the Z Coordinates

The procedure stated so far, would provide a suitable path for a cartesian manipulator without further adaption. However, when applied to a revolute manipulator, the form of manipulator introduces the possibility of collisions with forearm as it stretches over an obstacle (figure 8.6). Hence, the inverse kinematics

must be calculated in order to determine the manipulator limb positions and so modify the path.

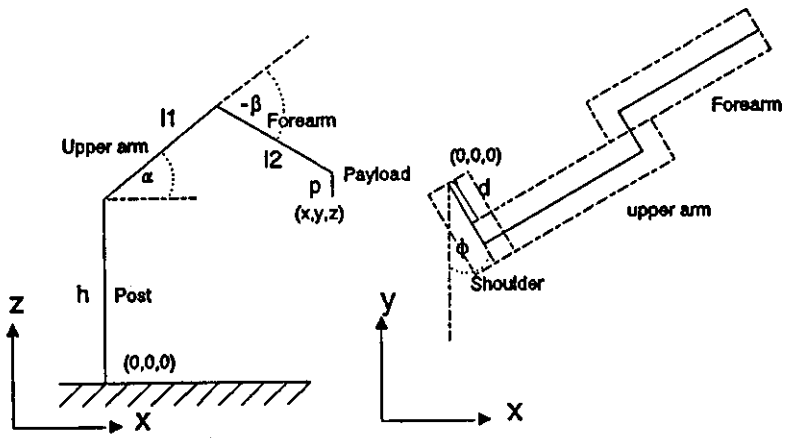


Figure 8.7 Schematic of manipulator.

The manipulator used in the initial experiments was a Puma 560, which has 6 degrees of freedom. In order to simplify and quicken the inverse kinematic calculations joint 4 (the upper wrist of the manipulator) was fixed. This reduced the manipulator to 4 degrees of freedom, which allows for a less complex geometric form of solution to the inverse kinematics[1], figure 8.7. This is quicker to calculate, while the 4 degrees of freedom still allows enough flexibility to complete assembly operations[1,13]. The geometric solution for the Puma is shown below,

$$w := z + p - h$$
$$e := x^2 + y^2 - d^2$$
$$r := e^{1/2}$$
$$c := (e + w^2 - l_1^2 - l_2^2)/ 2l_1l_2$$
$$s := -(1 - c^2)^{1/2}$$
$$a := l_1 + l_2c$$
$$\phi := atan(y,x) + atan(d,r)$$
$$\alpha := atan(aw - l_2sr, ar + l_2sz)$$
$$\beta := atan(s,c)$$

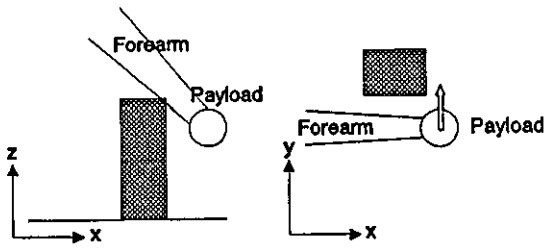


Figure 8.6 Side swiping.

To avoid "side swiping" and "stretching" collisions, a simple heuristic was applied to modify the z-coordinates of the via-points. This involved ensuring that along any path segment, the via points were chosen so that lowest point on the forearm of the Puma was always above the footprint of the manipulator. The footprint was defined as an RP, whose x and y dimensions were determined by the maximum and minimum X-Y coordinates (i.e. xmax,ymax,xmin,ymin) which were swept out along the path segment by the manipulator limb (see figure 8.8). The z coordinate was set equal to the maximum height of the obstacle contained within this x-y rectangle.

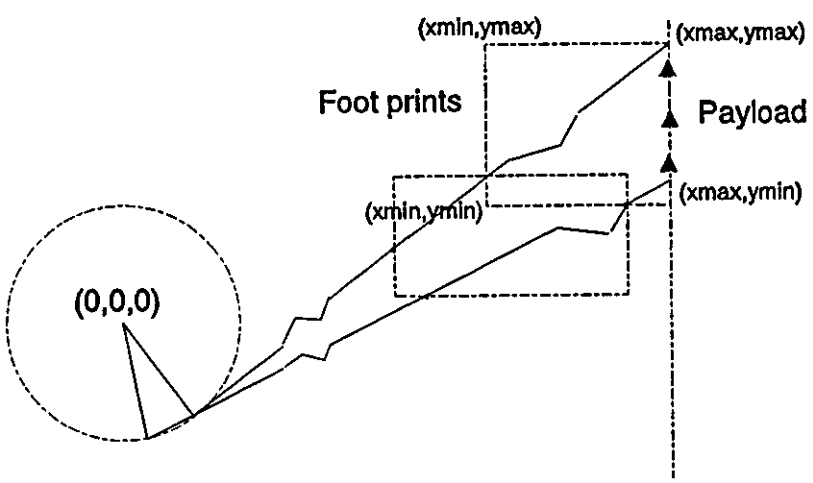


Figure 8.8 Foot prints.

The footprint is determined in two stages. Firstly, the extreme X-Y coordinates were calculated from the geometry of the manipulator, using the inverse kinematics, and the foot print in the X-Y plane determined. The z coordinates are set to the maximum and minimum extent of the work space in z. Secondly, these initial forms of footprint RP are tested for intersection with the object RP's. The intersections are used to determine the maximum height of objects within the path segment. The z coordinate of tallest object contained within the path, is then used to fix the final height of the footprint.

Experimental Results

A number of paths were planned (using a 12MHz AT computer running the planner implemented in Forth), based upon those actions determined in [13] and using information derived from GeoMod. The RP's, whose axes needed to be aligned with those of the *WORLD,* were created from the spheres which bounded the boundary representations of the *BODY's*. The Puma was controlled via its parallel interface and a resident VAL-1 program, which allowed the planner to move the end effector in steps of 5 or 10 mm. It was found that the generation of free space took between .2 and 1.5 seconds and the determination of the via points took between 2 and 5 seconds, depending on the complexity of the path and number of via points.

Later experiments, using the Mini-Mover manipulator and RP's derived from the 3D vision system, took 3-4 times as long for similar paths. This was due to the superfluous detail provided by a more exact model of the scene, and as a result the planner spent the majority of its time creating free space which was too small to be of use during the planning stage. The Mini-Mover was controlled by a server KS, which provided the usual features of a commercial controller (i.e. straight line motion etc), as well as interactive control of the path during motion (Note: this feature is not available under VAL-1).

The planning times, in the experiments which derived their information about the work space from the geometric modeller, were comparable with the time taken for the manipulator to execute the path. This makes it possible to pipeline the execution of a movement and the planning of the next movement. The code has not been optimised, and it is expected a factor of improvement could be made after optimisation.

Conclusion

A new technique for on-line collision free path generation for assembly operations has been developed, and demonstrated under Plethora. It is efficient and might be used as a part of an on-line adaptive path planner. Importantly, it makes no requirement on Geomod to model solids in motion. Further work is required in order that it might cope with paths which require reorientation of the work piece

References

[1] Brooks,R.A.
"Planning collision-free motions for pick and place operations", The International Journal of Robotics Research, vol.2, no.4, 83.

[2] Lozano-Perez,T.
"Automatic planning of manipulator transfer movements", IEEE Trans. on Sys.,Man,Cyber., vol.SMC-11, no.10, 81.

[3] Schwartz,J.T. Sharir,M.
"On the piano movers problem II:General properties for computing topological properties of real algebraic manifolds", Rept. 41, New York University Dept. of Computer Science.

[4] Davis,R.H. Camacho,M.
"The application of logic programming to the generation of paths for robots" Robotica, vol.2, 83.

[5] Grechanovsky,E. Pinsku,I.
"An algorithm for moving a computer-controlled manipulator while avoiding objects", 8th IJCAI, p806-813, 83.

[6] Kuntze,H.B. Schill,W.
"Methods for Collision Avoidance in Computer-Controlled Industrial Robots", 15th ISIR , p519-529, 85.

[7] Lozano-Perez,T.
"Spatial-planning: A Configuration Space Approach", IEEE Trans. on Computing, C-32, p108-120, 83.

[8] Udupa,S.M.
"Collision Detection and Avoidance in Computer Controlled Manipulators", 5th IJCAI, p737-748, 77.

[9] Wong,E.K. Fu,K.S.
"A Hierarchical Orthogonal Space Approach to Three-Dimensional Path Planning", IEEE Journal of Robotics and Automation, vol. RA-2, no.1, 86.

[10] Hayward,V.
"Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Work Space", IEEE Int. Conf Robotics and Automation, p1044-1049, 86.

[11] Herman,M.
"Fast, Three Dimensional Collision-Free Motion Planning", IEEE Int. Conf. Robotics and Automation, p1056-1063, 86.

[12] Connolly,C.I.
"Cumulative Generation of Octree Models from Range Data", IEEE Int. Conf. Robotics and Automation, p25-32, 84.

[13] Roth,J.P
"Algebraic topological methods for the synthesis of switching systems", INT.Trans. America Maths Soc., July, p301-6, 58.

[14] Nevins,J. Whitney,D.
"Computer-controlled assembly", Scientific American, no. 238, p63-74, 78.

Chapter 9
Experimentation

The function of this chapter is to illustrate some of Plethora's unique features and its operation as an entity. The chapter describes a contrived task, performed under the control of Plethora, in an experimental "toy" work cell. The examples also highlight a number of implementation issues which are then discussed in Chapter 10.

The intention of each experiment is to illustrate a particular feature of Plethora, and not to propose solutions to the Fitter problem.

Introduction

The difficulty in assessing a framework is to demonstrate the validity of its approach, without producing a number of large scale examples which exhaustively exercise it's facilities. This is particularly evident in the case of Plethora. It's function is to enable the development and comparison of new techniques, and so the effort required to produce comprehensive examples, would rival that originally expended in its development. (This was brought home forcibly, during the development of the path planner and object recognition KSs)

The examples in this chapter arrive at a compromise. Firstly, by performing an assembly task which illustrates that Plethora can incorporate as sub-techniques, a number of the techniques found in Chapter 2. Secondly, introducing an error into one part of this assembly sequence, to induce an error correcting behaviour which is not possible using the techniques in Chapter 2. In so doing, it is hoped to demonstrate the flexibility of Plethora without the need to generate a number of complete systems.

The Problem

Description of the Work cell

The work cell comprises a MiniMover-5 robot under the control of an IBM

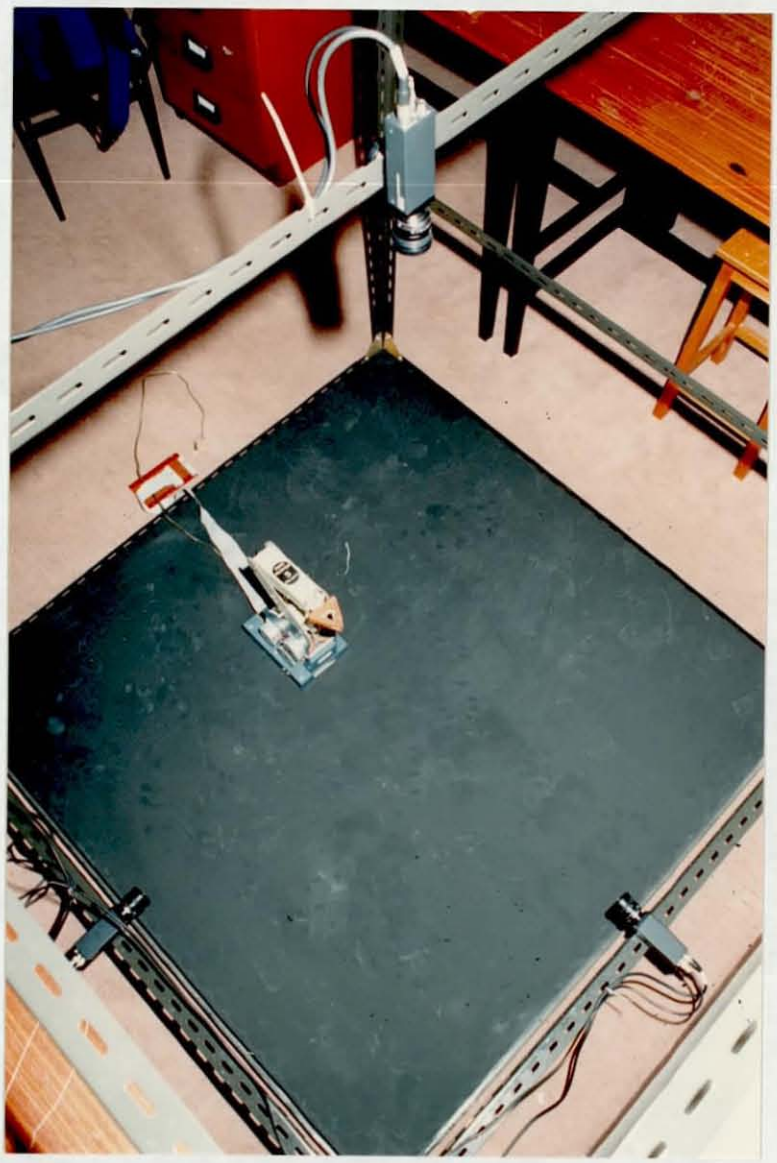


Figure 9.0 Photograph of Toy Work cell.

PC, and 3 cameras, focused on the work space along three orthogonal axes (see figure 9.0 for a photograph of a partial reconstruction of the work cell). The Mini-mover's gripper is equipped with infra red binary proximity sensors, controlled by a 65F12 processor. The frame grabber, special purpose image processor board and camera multiplexer (Data Translation boards DT2851, DT2859, DT2858) are hosted

within another IBM PC. They are connected to the rest of the network via a PC containing the message passing switch *Aether* (see figure 9.1).

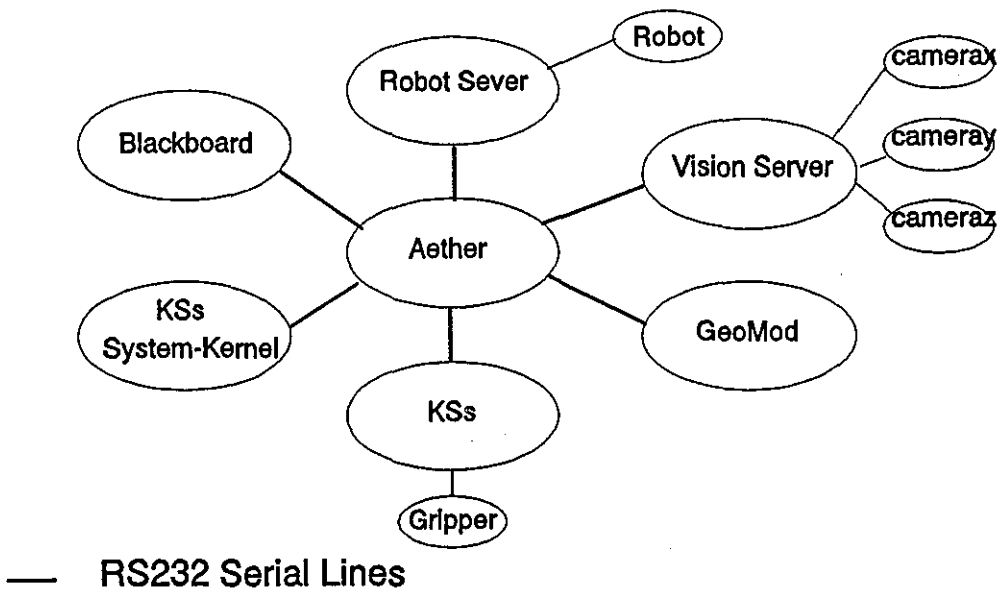


Figure 9.1 Network connection and resident KS's.

The Assembly

The assembly was constrained by three attributes. Firstly, the choice of the assembly was severely curtailed by the equipment available. Secondly, the task should contain at least one Pick-and-Place operation, since they are found in the overwhelming majority of assembly tasks[1]. Thirdly, it should minimise the planning detail (since the adequacy of planning with a similar architecture, has already been demonstrated in [2]), while still providing suitable features to explore the use of Plethora during execution.

The task set Plethora was to assemble the structure in figure 9.2. The work pieces were constructed from Lego (see figure 9.3), whose characteristics were entered into the geometric modeller. The accuracy of the Mini-Mover, within the work space of the assembly and under the control of the manipulator server KS, was bounded by +/- 4mm. Hence, to compensate for the misalignment, owing to the limited accuracy of the manipulator, the holes in the lid of the box were 25% larger

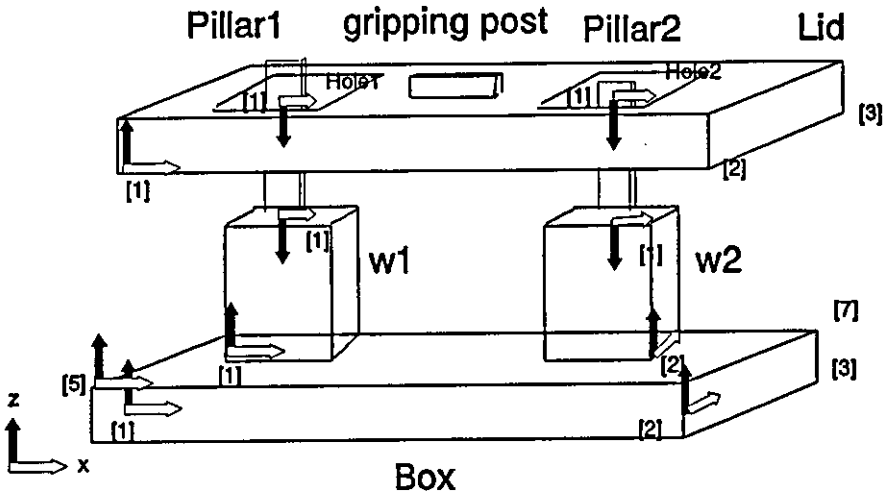


Figure 9.2 Schematic of the experimental assembly.

than necessary. The square cross section of the pillars and additional gripping post on the lid, were added to ease gasping by the Mini-Mover.

The description of the assembly, its expansion and final set of *PRIMITIVE* goals are shown below.

Original description

Lego-Construction(box,w1,w2,lid):

Affix-lid{(box[5],lid[1]) (box[6],lid[2] (box[7],lid[3]) (box[8],lid[4])}

Against{(box[1],w1[1]) (box[4],w1[4])}

Against{(box[2],w1[2]) (box[2],w1[2])}

Around{pillar1[1],hole1[1]}

Around{pillar2[1],hole2[1]}

Original description expanded into Primitive relationships

Equal(box[5],lid[1])
Equal(box[6],lid[2])
Equal(box[7],lid[3])
Equal(box[8],lid[4])

Equal(box[1],w1[1])
Equal(box[4],w1[4])
Equal(box[2],w2[2])
Equal(box[3],w2[3])

Star(pillar1[1],hole1[1])
Star(pillar2[1],hole2[1])

Selected necessary Primitive relationships.

Equal(box[1],w1[1])
Equal(box[2],w2[2])
Equal(box[5],lid[1])

Black backgrounds with white workpieces were used to maximise the image contrast and simplify the image processing. In an attempt to reduce the effect of leading and trailing shadows, each camera had two light sources one on either side (see [3]).

The following is a paraphrased description of the log file produced by the BB, during the planning and execution of the task. Absolute timings are not included, as the BB was in single step operation in order that screen dumps could be taken and errors introduced. Plethora ran using the Default Basic-KS.

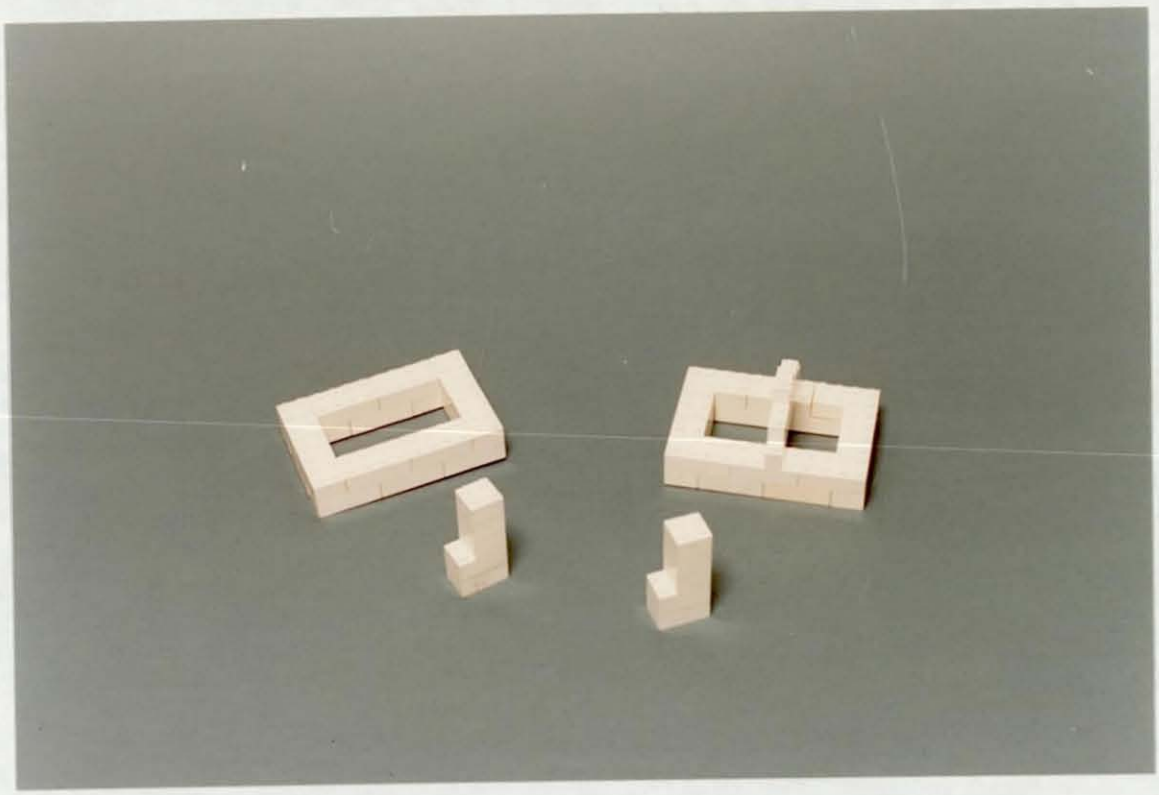


Figure 9.3 Photograph of the Lego pieces.

Planning the Task

Initially, the BB is devoid of decisions and it's *mode* is *Awaiting-Problem*. During *Awaiting-Problem*, the BB will only respond to the *System-Kernel*. It changes it's *mode* to *Planning* at the end of a *System-Kernel* transaction containing a *Problem* decision entry. Once the *mode* is *Planning*, the normal problem solving behaviour begins. During this initial period the *Outcome*, *Policy* and *Problem* shown below, are added to the BB.

> ***Policy:*** *Prefer-Control-Decisions*
>
> This ensures a general top down approach to the planning by rating KSARs, which produce decisions in the Control plane, higher than others by adding *Very-High* to their ratings.
>
> ***Problem:*** *Plan a single solution to the current Outcome*
>
> The *Problem* goal and its status are checked by the Basic-KS during each cycle of the planning process. It checks to see whether the *Outcome* is *Planned* (as defined in Chapter 3) and the *Problem Completed*.

Outcome: Assemble Lego construction

Before the experiment began the construction details were entered in to the relational data base and geometric details of the workpieces into geometric data base. Hence, the *Outcome*'s goal could be interpreted by the *Solve-Construction* KS.

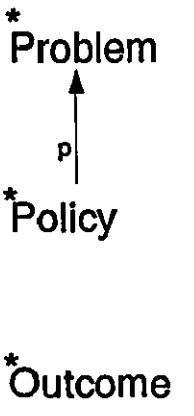
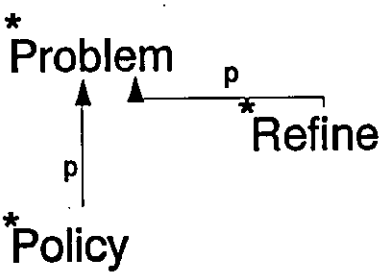


Figure 9.4 Initial contents of BB.

The entry of the *Problem*, the *Outcome* and the resulting change in BB *mode*, elicits broadcasts of the internal events (figure 9.4 shows the contents of the BB prior to *Planning*. P indicates a parental dependency relationship, b a *Before/After* relationship and a star indicates a *Viable* decision). As a result, the new *Problem, Policy* and *Outcome* goals are send to those KS's in the KSAR *Group* slot. The change in BB *mode* is sent to all KS's which have their action in the control plane of the BB.

In this limited population of KS's, this triggers the *Assemble-Approach* KS and *Refine* KS (described in Chapter 3). The *Assemble-Approach* KS is triggered by the entry of an *Outcome*, which it thinks it can help to design (i.e. it is an *Assemble* goal). The *Refine* is triggered in response to the single solution goal in the *Problem.* However, on the basis of its higher action level the *Strategy Refine,* is made the *Internal-Chosen-Action.* It replies with a *State-Change* message, entering the *Refine Strategy* onto the BB (see figure 9.5).



*Outcome

To-Do-List: Assemble,Implement-Refine

Figure 9.5 Refine Strategy decision.

This once gain triggers the *Refine-KS,* but this time it's aim is to implement the *Strategy* decision. (Note: In Plethora, unlike BB1, different KSs can compete to implement a *Strategy.* However, for the sake of simplicity *Refine* volunteers and implements its own *Strategy).* This becomes the preferred KSAR, and the *Foci* (shown in figure 9.6) are entered. Each Focus (*Pdg*, *Pprc*, *Pop* and *Pimp*) prefers KSARS with actions at different levels in *Outcome* plane, beginning with the *Design* level e.g.

Pdsg

Focus: Prefer KSARs with actions at the Design level.

This adds *High* to each KSAR rating which has its action at the *Design* level. It terminates when all *Outcomes* have decisions in the *Design* level.

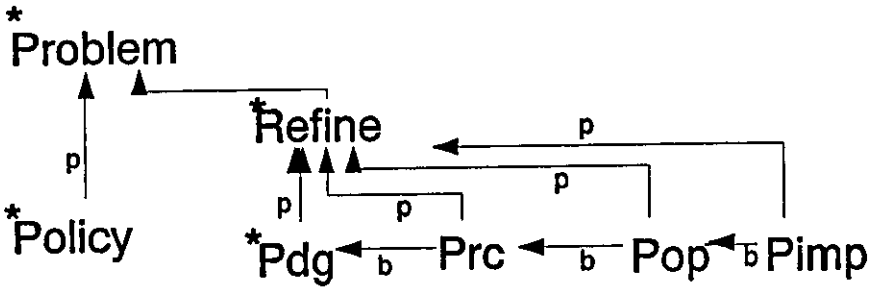


Figure 9.6 Foci of Refine strategy.

During the next cycle *Pdsg* has it's effect (since it is the only *Viable Focus)*, and a *Design* decision with a *Least-Commitment-Assembly-Sequence* goal (Note: this was originally triggered by the *Assemble* goal), is entered. This in turn triggers the *Solve-Construction* KS (described in Chapter 6).

However, the entry of the *Least-Commitment-Assembly Design* decision satisfies *Pdsg*'s termination condition and so it *Completes*. The next *Focus* in the *Before* order (i.e. *Pprc*, which prefers decisions at the *Procedure* level) now becomes *Viable*. Hence, the *Solve-Construction* KSAR is preferred in the next cycle and the KS expands the *Outcome* goal. In doing so, it forms a network of *Equal* goals within the *Procedure* level (see figure 9.7). This process continues until *Implementation* level decisions are made, which simultaneously *Completes* the *Refine Strategy* and the *Problem.*

The population of KSs was deliberately restricted so that each *Equal* goal would be "implemented" in the same manner. The intention was that this operation could be investigated separately from the rest of the plan, during the error correction

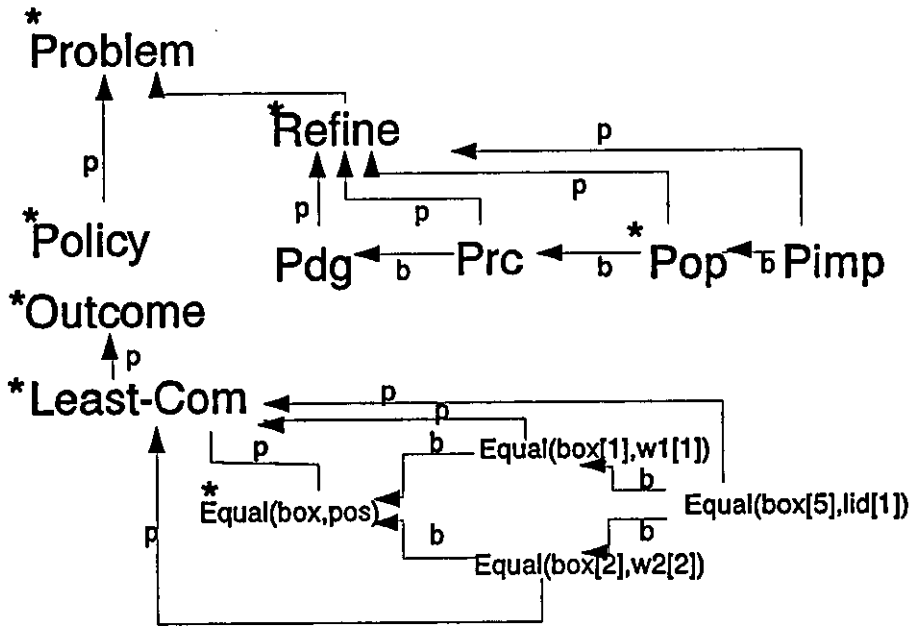


Figure 9.7 Operational Plan.

portion of the experiment. (see figure 9.8, The decisions in brackets are the *Implementation* decisions chosen for that operation. The dotted parent links point to the *Equal* goal).

In this example there were no competing KSARs left in the *To-Do-Set*, which had an *Implementation* KSAR *Failed* during execution, would have provided a source of alternative actions at run time.

Summary

This simple example illustrates how a technique, which has all the efficiency advantages of a PAOH-like approach (i.e. directly expands a partially ordered set of goals), can be implemented using the *Solve-Construction* and *Refine* KSs. However, unlike PAOH, *Solve-Construction* generates the partially ordered set directly without the intervention of ordering heuristics.

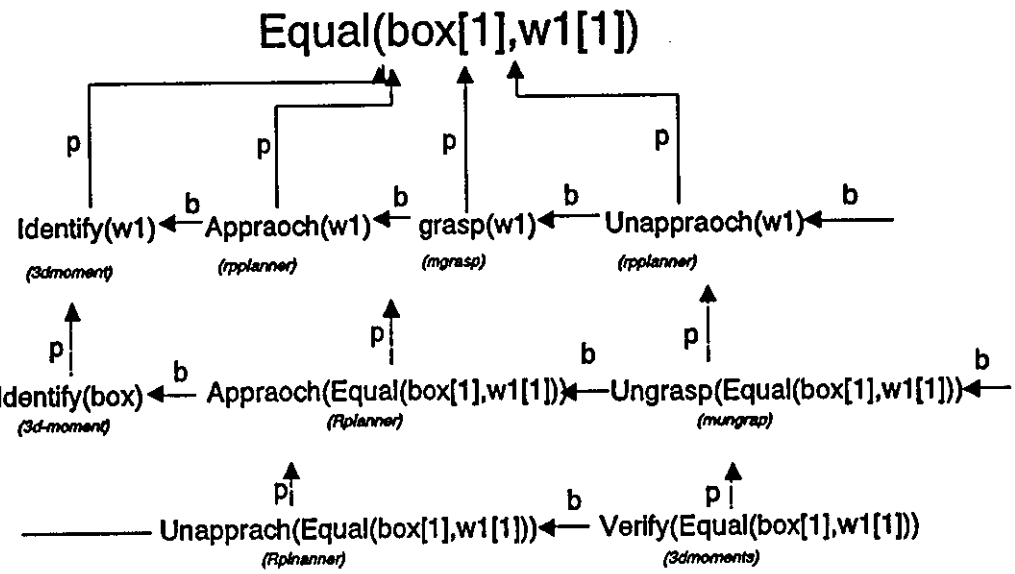


Figure 9.8 Single Equal Plan.

More sophisticated strategies might have provided a number of alternative *Implementation* decisions for each *Operation* goal (e.g. in the style of *Spar*). This could have been achieved with a simple modification to *Refine*, however, for the sake of simplicity this was not done.

Description of the Implementation KSs

3D-Moment

This uses the technique described in Chapter 7. The KS was shown separately each stage of construction, and used 10 views of each stage to build up a range of feature values. The nominal feature value was taken to be the mean of the measured values.

The use of teaching is not consistent with the aims of Plethora, since the teaching process increases the latency of system. Ideally, the features should be generated from predicted scenes using GeoMod, however, due to the jitter problem, such values are not easily derived. Chapter 11 proposes a technique by which this may be achieved, given suitable hardware to reduce the jitter problem.

RPplanner

This is a KS based upon the method given in Chapter 8. It initially solves the transformation described in its *Action goal,* by requesting the position of the robot from the robot server, and then plans the path, based upon the information found in GeoMod. Once planned, it send *State-Change* messages to the robot server to move the robot and updates GeoMod with the new robot position. The motion is halted by the BB sending an *Abort* or *Halt* message to RPplanner, which it then sends on to the server.

In order to cope with the inaccuracy of the robot, the sphere representing the gripper and payload, was set 20% larger than given in Chapter 8.

2D-Moment

This bases its recognition procedure on the 2D moment invariants found in [4]. These are calculated from the chain code of the blob with the longest chain code sequence, using the technique given in [5]. Its feature values were calculated in the same manner as 3D-Moment.

Approach and Unapproach

These are extremely simple. *Approach* evaluates the transformation in the *Action goal* and determines the height of the highest RP presently part of the construction. It then moves, under the control of the RPplanner, to the x-y position determined by the solution of the transformation. The z coordinate of this position is given by the z coordinate of the highest RP plus 2Vr (see figure 9.9). Similarly,

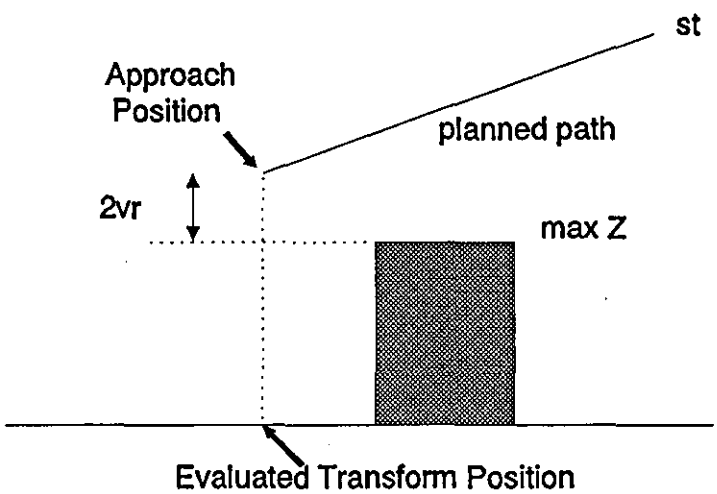


Figure 9.9 Calculation of approach positions.

Unapproach moves the manipulator from it current position directly "upwards" (i.e. in the positive z direction), using the same form of calculation. These make the assumption that all positions in the assembly can be reached from above (Note: this holds for both of the practical assemblies given in the Flymo and Fox examples).

Mgrasp and Mungrasp

Because of the 4 DOF restriction made by the RPplanner, the possible grasping positions are limited to grasps from above. *Mgrasp* finds its grasping point, by first interrogating the relational plane of the modeller, to determine whether the object contains any elements which belong to the *SET* of *Gripping-Posts* (i.e specifically designed additions to the object). If the object has a *Gripping-Post*, then this will be grasped otherwise, it uses the highest element of the object. In either case, once the element of the object to be grasped has been determined, the centre of

the central bounding sphere in the x-y plane, is chosen as the grasping point. This is a simplified form of the technique given in [6].

Execution

A second problem, *Complete-Outcome,* was now sent to the BB to execute the plan, this triggered a *Strategy* (*Least-Commitment-Execution*) and two *Policies.* Together they implement a form of opportunistic scheduling, similar to Fox's method.

Least-Commitment-Execution

> ***Terminate Strategy:***
> *Outcome Completed* with a *Confidence* >75%
>
> ***Focus0 Goal:***
> Prefer *Viable* decisions at the *Implementation* level which have no *Before* links.
> ***Terminate Focus:***
> A single *Completed* goal, as described in the focus, which has a *Confidence* > 75%
>
> ***Focus1 Goal:***
> Prefer *Viable* decisions at the *Implementation* level with the greatest number of *After* links.
> ***Terminate Focus:***
> Outcome *Completed* with *Confidence* >75%

This is not identical to the method used in [7], since it's equivalent of *Viable* actions were implicit in the constraint equations. It had a method which distinguished between competing actions based on the time taken to expand the equations and determine the next action. However in Plethora, the planned actions are explicit and so *Least-Commitment-Execution* finds a starting point in the plan (using *Focus0),* and then prefers *Implementation* decisions which enable the maximum possible number of actions (using *Focus1).* This as its name implies, effectively extends the least commitment strategy used in planning, to the execution stage. The *Policies* below are used to decide between equally rated KSARs.

Prefer-Most-Efficient-KSARs
This increases the rating of a KSAR in direct proportion to its *efficiency*.

Prefer-Most-Confident-Predecessors
This increases the rating of a KSAR by an amount proportional to the average *Confidence* of the *Completed* decisions in the *Before* links, of the decision the KSAR is to fulfil.

Initially, there is only one *Viable Implementation* decision and so it's solution KSAR is chosen as the *External-Chosen-Action*. This executes successfully, returning a *Confidence* value and the position of *box*, the KS transmits the position to the geometric modeller as a *State-Change*.

This makes the "Pick-and-Place" actions (*Equal(box[1],w1[1])* and *Equal(box[2],w2[2])*), and their initial *Operational* and *Implementation* decisions, *Viable*. However, as their *Implementation* decisions have no predecessors and so no *Confidence* values to differentiate between them, the *Implementation* decision with the most efficient KSAR is chosen. In this case, the goal *Equal(box[1],w1[1])* is preferred as it's KSAR has the higher *efficiency*. This sequence was left to continue unperturbed, eventually *Completing* the *Problem* goal.

In the next experiment, the illumination used by $camera_x$ was turned off prior to the identification of *w1*. The identification procedure continued, but only elicited a *Confidence* of 10% from *3D-Moment*, *Failing* the *Implementation* goal. This had the effect of retaining the identification goal as the only *Viable* goal in that sequence. The presence of the *Least-Commitment-Execution Strategy*, forced the BB to pick out another KSAR to solve the *Viable* goal. As a result of the broadcast of the failure of the identification goal, a KSAR volunteering *2D-Moment* using $camera_z$, was selected as an alternative solution to the *Viable* goal. This produced a *Confidence* value of 80% and allowed the plan to proceed as before. The experiment illustrates how a limited degree of fault tolerance can be achieved with the use of Plethora.

Summary

The previous example shows how a primitive form of adaptive behaviour might be produced by Plethora. However, this carefully contrived experiment was performed without the use of the proximity sensor *Prompter*, and so contained no *Events*. The following section illustrates how *Events* can be used to alter the predicted order of execution or reinforce the *Confidence* of *External-Chosen-Actions*, extending the range of adaptive behaviours. The example concentrates the pick and place operation described by the *Equal(box[1],w1[1])* goal.

Execution With Events

In the modified scenario, the first of two events proffered by the proximity sensor *Prompter* occurs during the execution of *grasp(w1)*. The first indicating the grippers transition from empty to full, and the second from full to empty. Both *Events* trigger the *Reconcile-Event Strategy*, focusing the attention of the BB on the reconciliation of the *Event* (see figure 9.10).

Reconcile-Event

Strategy Trigger:

A new *Event* during Execution.

Terminate Strategy

Event is reconciled.

Focus0

Prefer most *credible* decisions in *Execution* Plane concerned with the *Event.*

Terminate Focus0

Event has a *Solution* decision (i.e. is reconciled).

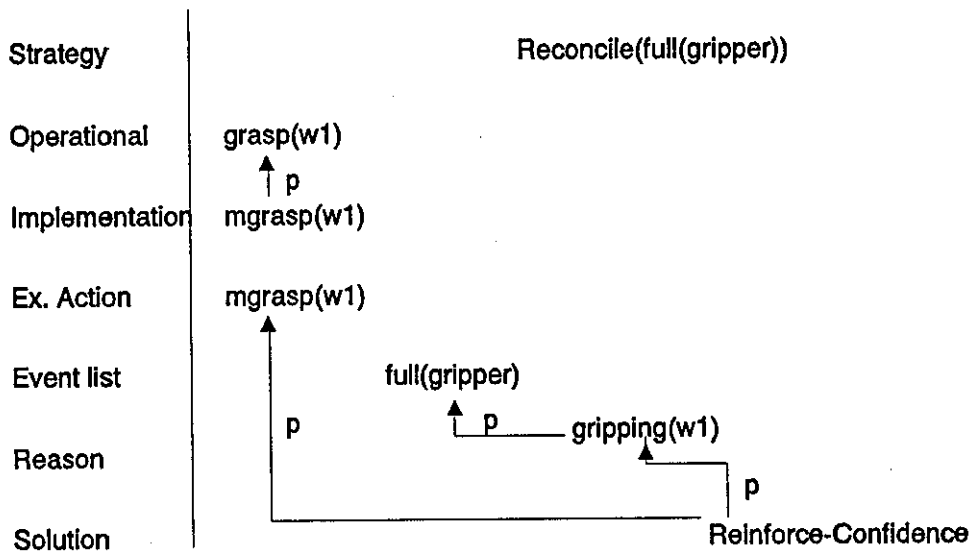


Figure 9.10 Reinforcement of confidence.

The first *Event* triggers a KSAR with action at the *Reason* level, which is preferred by the BB rather than further execution. The KS notes that the current *External-Chosen-Actions* function is to fulfil the *Implementation* goal to "grasp". As a result of this information and the contents of the *Event's* goal, it decides that the *Reason* for the *Event* is that *gripping(w1)* is taking place (see figure 9.10). A KS, *Reinforce-Confidence*, which acts as a execution critic (similar to the plan critics in [8]) and has its action at the *Solution* level, is then triggered and chosen as the internal action. It recognises that the *Reason* reenforces the *External-Chosen-Action,* and so then transfers the *Confidence* value from the *Event* to the *External-Chosen-Action,* changes the *Status* of the *Event* to *Reconciled*, thus *Completing* the *Focus* and *Strategy.*

The next action is selected and successfully *Completes* and so *w1* begins to move. However, during *Approach(Equal(box[1],w1[1]))*, *w1* is removed from the gripper introducing an error.

Execution With an Error

The removal of the block elicits a full to empty *Event* from the proximity sensor Prompter, which in turn triggers *Reconcile-Event*. However, this *Event* also results in a *not_gripping(w1) Reason* decision, in a similar manner in the previous experiment. The execution critic then *Requests* the BB for previous *Operational* goals which have been fulfilled, and recognises a contradiction between *not_gripping(w1)* and *grasp(w1)*. It enters an *Error* decision, *Dropped(w1)*, the *Confidence* of which is calculated from the original *Events Confidence*. The *Error* decision triggers a *Solution*, *"Replan_Parent_Goal"* (referring here to the *Equal* goal), which *Reconciles* the *Event*. The *External-Chosen-Action's Confidence* is made equal to that of the *Event empty(gripper)* and it's status set to *Error*. This forces the blackboard to

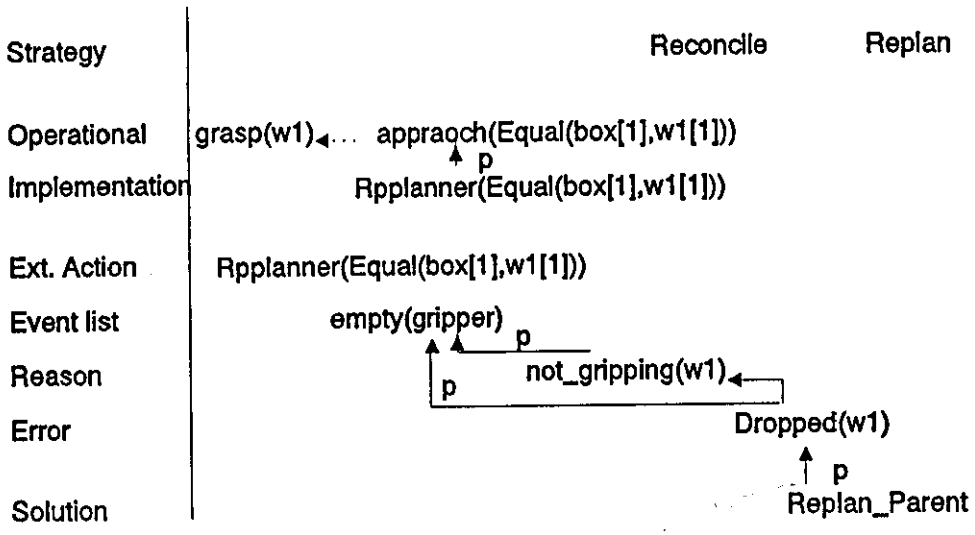


Figure 9.11 Reconciling an Error.

terminate the *RPplanner's* action, inhibit all the *Implementation* goals which belong to the parent goal, and change the *mode* to *Planning*.

The *"Replan_Parent"* goal triggers a *Strategy (Replan)* which prefers decisions concerned with the solution of *grasp(w1)* until it is *Planned*. Once it is replanned (in the same manner as before, in this case), execution recommences with the re-identification of *w1*.

Summary

This example illustrates the incremental nature of Plethora's error detection/correction behaviour. In comparison, if the error detection and correction were explicitly predicted (i.e. in the style of SPAR), an error not contained in the enumerated list or an error occurring after the time of the test, would go undetected. However, the incremental nature of the process in Plethora, its separation into detailed stages (indicated by the *Reason, Error* and *Solution* Levels), and the ability to reason during an *External-Chosen-Action,* allows it to react to unexpected *Events* at anytime during the execution of the plan.

The system would have reacted in the same way, if different sensors had been used but which sensed the same type of *Events*. Hence, the attributes above also help to fulfil the modular behaviour required in *ct2,ct4*.

During execution, the critic KS made use of previous goals which indicated the intent of the current action (i.e. Grasping), and used these to determine the course of actions to follow. Although, the example is a simple one, it illustrates a technique unavailable to the other systems. (Note: The approach could have been extended and used to investigate learning techniques, such as is discussed in [9]).

Discussion

The author is the first to agree that the experiments, with all their limitations and assumptions, do not demonstrate Plethoras use as a practical solution to the Fitter problem. However, they do illustrate how it can fulfil goals given in Chapter 1 (which were argued to be necessary for such a solution). It also demonstrates that Plethora can exhibit advantageous characteristics not found in the work in Chapter 2. Namely,

> The example moves between planning and execution using information from both phases, to guide its action.
>
> Sensors can be added without modification to the original planning.
> Fulfilling *ct4*
>
> Plethora has been shown to be able to opportunistically schedule execution and planning operations. In Chapter 2, this was highlighted as a means of achieving *ct1* and *ct3*. However, Plethora has the additional advantage that it is able to base its opportunism on events other than part availability.
>
> Plethora can adapt it's behaviour to unmodelled and unexpected events as they occur. It also can reason about the progress of any action it undertakes.
> Partially fulfilling *ct2* and *ct4*
>
> Plethora's error correction strategy is incremental and can take into account the intent of a particular action

However, the examples do not illustrate how a YAMS type of behaviour (where it is possible for the initial planning stages to be guided by sensory data) could be exhibited by Plethora. A method whereby this might be achieved is discussed Chapter 11.

References

[1] Nevins,J. Whitney,D.
"Computer Controlled Assembly", Scientific American, no. 238, p63-74, 78.

[2] Hayes-Roth,B.
"A Blackboard Architecture for Control", Artificial Intelligence, no.26, p251-321, 85.

[3] Schroeder,H.E.
"Practical Illumination Concept and Technique for Machine Vision Applications",
Robot Sensors, Edited by A. Pugh, IFS Publications Ltd, ISBN 0-948507-01-2, p289-244, 86.

[4] Hu,M.K.
"Visual Pattern Recognition by Moment Invariants", IEEE Trans. Information Theory,
vol.8, p179-187, 62.

[5] Pugh,A.
"Processing Binary Images", Robot Sensors, Edited by A. Pugh, IFS Publications Ltd,
ISBN 0-948507-01-2, p63-87, 86.

[6] Kak,A.C. Vayda,A.J. Cromwell,R.L. Kim,W.Y. Chen,C.H.
"Knowledge-Based Robotics", International Journal of Production Research, vol.26, no.5,
p707-734, 88.

[7] Fox,B.R. Kempf,K.G
"Opportunistic Scheduling For Robotic Assembly", IEEE Conf. Robotics and Automation,
p880-889, 85.

[8] Wilkins,D.E.
"Practical Planning Extending the Classical AI Planning Paradigm", Morgan Kaufmann
ISBN 0-934613-94.

[9] Hayes-Roth,B Hewitt,M.
"Learning Control Heuristic in BB1", Technical Report, STAN-CS-85-1036, Stanford
University.

Chapter 10
Implementation

It is intended that Plethora's response should satisfy the critical times of a real work cell. While, in the main, it is able to do this for the "toy" work cell, there are implementation bottlenecks which cause the present version to be too slow for use with a practical work cell. Similar delays would arise in the toy work cell, if Plethora were used to investigate techniques which required very many more KS's than in Chapter 9.

This chapter begins by describing Plethora's underlying programming structure and then proceeds to discuss separately, those sources of delay inherent in Plethora's architecture, and those which are a function of the implementation alone. It also describes modifications which can be employed to overcome these inefficiencies, and gives timing details for the effects of the modifications.

System Implementation

It has been pointed out in similar earlier studies, that significant difficulty is encountered when integrating programs from different sources or levels of abstraction (e.g. in [1] the linking of a planner to a CAD modeller took approximately one quarter of the time for the project). It was decided to use Forth as the system implementation language, in the hope that it would reduce the degree of the problem.

Forth [2] is an extensible and interactive language, it can accommodate a range of programming styles (e.g logic [3,4], functional [5], object orientated [7], procedural [2] programming) and implement these techniques on a range of processors (e.g. the processors used so far include the single chip Rockwell 65F12 and Intel 80286). Using Forth, it was hoped that programs could be developed using the appropriate paradigm, and yet be readily integrated - this was shown to be the case.

Forth confers advantages in terms of its flexibility and system integration. However, when run on conventional hardware with conventional interpreters, it can be too inefficient to run complex tasks in real-time. In order to find those elements of Plethora which could be most effectively modified to reduce the overall execution time, a profile for each routine in Plethora was calculated. The profile (using a technique similar to that found in [7]) produced a pair of numbers which represented the number of times the word ran, and the worst-case time taken for execution.

The following section details the elements of Plethora with significantly large profile products (where this is the product of the profile pair numbers), and then discusses the changes (or proposed changes).

Dynamic Storage Allocation

Dynamic storage allocation is used extensively within the message passing system, and was initially implemented using the MSDOS allocation routines. However, this was found to consist of approximately 10% of the total message passing product. An alternative arrangement, in which space for all messages was pre-allocated during initialisation and routines written to implement a simplified dynamic allocation strategy, reduced the allocation\de-allocation element of the product to approximately 2%.

Two forms of storage are used to support the basic features of the message passing system, *chunks* (128 bytes) which hold the message and *links* (8 bytes), which are used in the link lists of the message passing system. This space is hidden from the rest of the system and *Players* requiring dynamic storage declare their own separately (using storage allocation system routines provided by Plethora).

Rule Tables and Message Interpretation

During the development of Plethora, rule tables (using the procedural interpreter) were used to parse and evaluate messages. Figure 10.0 shows the extensive use made of rule tables in the implementation of the BB. In the BB, a single rule table extracts messages from the message queue and passes them on to the relevant rule table to implement the necessary operation.

Although, this was useful during experimentation, it became apparent that it was too slow for use in all but the simplest system. This is epitomised by the KSAR message interpreter, this has 50 rules and takes 0.25-0.5 seconds to complete. Given

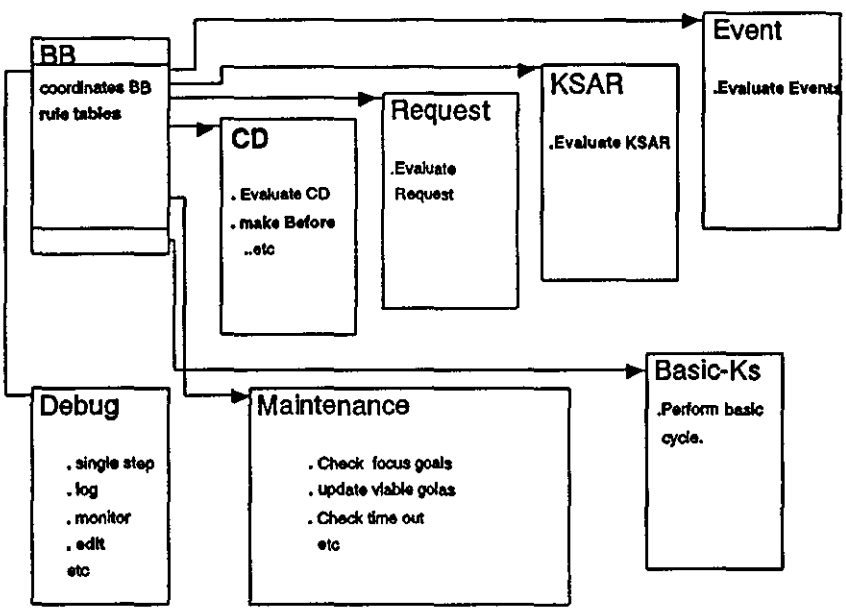


Figure 10.0 The structure of rule tables used to implement the BB.

that message interpretation takes place many times during one BB cycle, it can be seen that the time taken to interpret a message is a dominant factor in the response of Plethora.

The time consuming element of this method of message interpretation, is the association of the method token in the message with it's appropriate function. However, if the commonly used method tokens were not hashed, but were each given a value in the sequence 0 to n-1 (where n is the number of such methods). The association process could be implemented as a simple look-up operation in a table of addresses. In such a table, the K^{th} function address would implement the K^{th} method. The remaining methods could be hashed in the normal manner. If there were a large number of remaining methods, the association could be performed using a binary search technique [8] (reducing the complexity of the search to $\{\log^2(n) - 1\}$). The two types of method could be distinguished using the most significant bit of the method token.

Implementation of Objects

Intuitively, it is clear that the implementation of the retrieval and inheritance mechanisms will have a profound effect upon the overall efficiency of the servers (e.g. GeoMod etc). From the profile calculations, it can be seen that although the absolute times taken for these operations is relatively short, they are used so frequently that any increase in their efficiency would be welcome.

This could be achieved by re-designing the servers and/or by suitable hardware or compilation techniques. Since the latter approaches could be also applied to the other parts of the system, compilation and hardware changes are discussed separately. The following section describes a programming technique which is specifically aimed at increasing the efficiency of server objects.

At present objects are implemented in a naive fashion, where during the evaluation of a method a sequential search is made of the object's method slots using a *case* like structure. If the method is not found then the *Class* hierarchy is followed, where once again a sequential search is performed. This has the advantage of maintaining "late-binding", which is a feature of the flexibility of object orientated systems. However, when performing predefined operations (Note: many types of service request fall into this category), large portions of the bindings do not change and the enforcement of a late binding strategy becomes an unnecessary overhead.

These problems have been addressed in [9], where the use of late and early binding is controlled by the programmer and the association of the method with it's operation is performed using a jump table. Together these techniques have been shown to reduce some of the overheads associated with object orientated programming.

Server Architecture

When an event occurs and is broadcast, the KSs evaluate their triggers and during this time many will require access to external information held in servers. The servicing of these requests forms a significant delay in the operation of Plethora.

The present approach (described in Chapter 5) uses a single *Player,* which extracts messages and then evaluates the *Request.* The *scope* mechanism gives each KS the illusion that it has the complete attention of the server, but in essence it is a set of interleaved sequential operations. However, the distribution of the computational effort when evaluating a *Request*, can be used to direct a possible parallel solution.

During many requests, the dominant portion of the time spent is in the evaluation of the methods, rather than in the time taken to access data base. (This is particularly noticeable in GeoMod when it evaluates compound transforms). Hence, if a number of identical *Players* were created to evaluate the requests, and another coordinated the messages to and from a *Company* message queue to the slaves. It would be possible to evaluate the *Requests* in parallel. This architecture is sometimes known as a Farm (see figure 10.1) and protocols already exist to implement this form of parallelism in message passing systems (e.g. Administrator [10]).

However, the scenario assumes the slave players reside on different processors and that they all have access to the data base. How this access is to be best achieved, depends upon the ratio of time of message communication to message evaluation. If the ratio is high, then a common memory solution [11] would be suitable and would require no software mutual exclusion mechanisms (since the data base is only read by

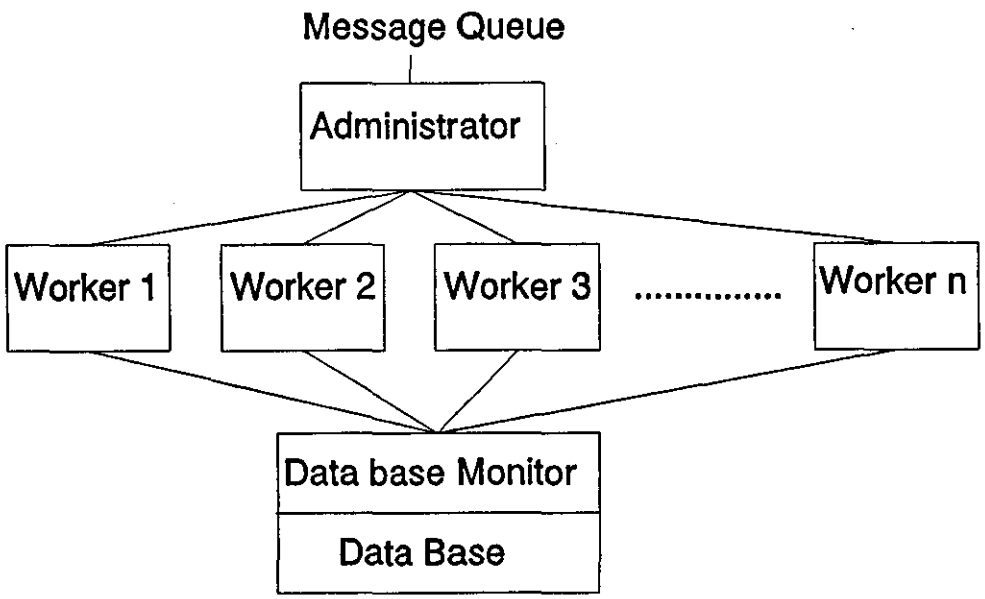


Figure 10.1 A farm implementation of a server.

the slave *Players)*. If low, a dedicated local network between master and slaves would suffice.

Network

Two points are not properly addressed in the experimental network described in Chapter 4. Firstly, a factory is an electrically "noisy" environment and so networks will be prone to communication errors. This is well understood (see [12]) and protocols and dedicated chips exist to implement real-time deterministic networks. Changing Plethora to operate with a more practical network would only require modifications to the network *Prompters*.

Secondly, the present network imposes delays on the transmission of high priority messages. They must wait for the completion of the message currently under transmission, before they can be sent. This problem has been addressed in INSTANET [13], and a similar approach could be used in a practical work cell.

INSTANET uses a commercially available network (IEEE 802.5 token ring) with additional hardware to implement a distributed interrupt. This mechanism allows messages to compete for access to the network and halt the transmission of the current message. If the current message has a lower priority than the waiting message, it halts the current message transmission and the higher priority message is allowed access to the network. Once completed, it resumes the transmission of the original message.

Compilation and Hardware

Plethora is written in a number of Forth's (FF,F83 and RSC-Forth), all of which are interpreted versions of the language. The majority of Plethora runs on Intel processors, these have restricted addressing modes (when compared with the Motorola 68000) and a segmented memory architecture. As a result, interpreted Forth implementations of large programs are slow when compared to other machines. Two options are available, either to compile the Forth or choose more powerful and appropriate hardware. Both approaches were investigated for critical pieces of code, using FEC [14] and a RTX2000 [15] development system.

FEC is a Forth compiler developed at Tartu University, and implemented in CFORTH. It was used to translate, test and compile selected pieces of code, to give an indication of the advantages of compiling Plethora. This was not a trivial task, as there are significant differences between CFORTH and FF. The code was run on an Intel 80286 at 12Mhz.

One of the most cost-effective and appropriate forms of hardware commercially available (at the time of writing), is the Harris RTX2000 family of processors. These execute Forth primitives directly, and because of the simplicity of the stack orientated nature of Forth, are able to run a number of primitives in parallel. The examples used a RTX2000 running at 20Mhz with a single wait state, and so not running at full speed.

The particular pieces of code chosen for the tests, implemented the homogeneous compound transformation, sphere intersection test and 3x3 image convolution. The choice of function was made based upon it's size of profile product. It can be seen that the functions are all similar, in that they are concerned with arithmetic manipulation. This similarity in operation led to a similar reduction in relative execution time for each of the functions.

Between the compiled CFORTH and interpreted CFORTH implementations of each function, there was an increase in speed of 4-5. However, this was only reflected as a factor of 2-3 over the original interpreted FF implementation. The apparent discrepancy between these two results lies in the fact that, CFORTH (a 32 bit FORTH running on a 16 bit machine) uses slower addressing modes than are used in FF, to implement many of its 32 bit operations. FF (a Forth with a 16 bit wide stack and running on a 16 bit machine) makes selective use of the segmented architecture of the Intel-80286, producing a more efficient implementation. The increase in speed of the unoptimised RTX2000 version over the FF version, varied between 12-15 times.

Summary

It is intended that Plethora should work with a practical work cell, and it is expected that the critical times will be shorter than those in the "toy" work cell. The exact length of these times is difficult to predict, as they will depend upon the cell's constituents and function.

However, a timing analysis of Plethora has isolated a number of programming techniques, which would increase it's efficiency. Preliminary timing results for compilation and appropriate hardware, indicate radical improvements could be possible with these approaches. In addition, there are also opportunities to increase the response of Plethora by capitalising on it's inherent parallelism.

References

[1] Fahlman,S.E.
"A Planning System for Robot Construction Tasks", Artificial Intelligence, no.5, p1-49, 74.

[2] Brodie,L.
"Thinking Forth: A langauge and Philosophy for Solving Problems", Prentice-Hall, ISBN 0-13-917568-7, 84.

[3] Odette,L.L. Dress,W.B.
"Engineering Intelligence into Real-Time Applications", IEEE Expert Systems,vol.4, no.4, p228-239, November 87.

[4] Odette,L.L.
"Compiling Prolog to Forth", Journal of Forth Applications and Research, Vol.4, no.4, p487-533, 87.

[5] Hoffmann,U.
"A Lisp-Kernel for the NC4000", Euro'FORML 87.

[6] Pountain,D.
"Object-Oriented Forth", Academic Press, ISBN 0-12-563570-2, 87.

[7] Pountain,D.
MPE-Forth, User Manual, MPE Southampton, UK.

[8] Pfaltz,J.L.
"Computer Data Structures", McGraw-Hill, ISBN 0-07-49743-5, 77.

[9] Rayburn,T.
"Methods> Object-Oriented Extensions Redux", Euro'FORML 87, p1-13.

[10] Gentleman,M.W.
"Message Passing Between Sequential Processes: The Reply Primitive and Administrator Concept", Software, Practice and Experience, vol.11, p435-466, 81.

[11] Hwang,K. Briggs,F.A.
"Computer Architecture and Parallel Processing", McGraw-Hill,ISBN 0-07-031556-6, 85.

[12] Tanenbaum,A.S.
"Computer Networks", Prentice-Hall, ISBN 0-13-164699-0, 81.

[13] Charkassky,V. Lari-Najafi,H. Lawrie,N.L. Masson,D Pritty,D.W
"The Performance of Real-Time LAN Architecture for Sensor Fusion Applications", IEEE Int. Conf. Robotic and Automation, p120-128, 88

[14] Saarsen,T.
"FEC-Forth Environment Compiler", Euro'FORML 90.

[15] Harris RTX2000, Data Sheet 5DS-0219, May 88.

Chapter 11
Conclusions

This chapter is divided into 2 sections. The first recaps the original requirements of Chapter 1 and discusses by what means and to what degree, Plethora fulfils them. The second section describes possible further work.

Control

ct1 ***The ability to automatically generate and schedule, actions and plans.***

This is achieved in three ways. Firstly, through the use of a declarative description for the assembly, which is specified in terms of the domain description found in the relational plane of the modeller. Secondly, through *Solve-Construction,* which expands the declarative description into procedural goals, and finally the uniform extension of BB1's planning structure to encompass execution.

ct2 ***To automatically interpret sensory information within the frame work of the plan and actions.***

This is achieved in two ways. Firstly, through the use of *Events* with their associated *Confidence* values and secondly, via the explicit representation of reasoning on the BB.

ct3 ***To implicitly detect and cope with error conditions.***

The method used by Plethora is based upon the explicit representation of reasoning, *Events* and the extension of BB control to include execution. The opportunistic nature of Plethora has been shown to provide a degree of fault tolerance (when a failure was induced by the change in illumination, described in Chapter 9) and an ability to resolve *Events* during execution (illustrated by the proximity sensor examples in Chapter 9).

ct4 ***To provide a modular implementation which allows the addition of knowledge and devices.***

This is achieved in two ways. Firstly, by encapsulating the procedural and geometric context of a technique/sensor/actuator within the structure of a KS, and so only storing centrally that information which is essential. Secondly, by employing a uniform message passing mechanism through which the incremental action of the conflict resolution scheme can operate.

Knowledge

kn1 ***The provision of physical information about the contents of the cell.***

This is partially achieved through GeoMod. However, it does not model uncertainty in object position or multiple moving objects. These limitations arise from the need to maintain the efficiency of GeoMod.

kn2 ***The provision of relational information to plan and interpret the task.***

This is provided by the contents of the relational plane of the modeller and the representation of the intent of an action on the BB.

kn3 ***The provision of information in a form suitable for its end use.***

It has been shown (via the development of the path planning and object recognition techniques) that the information provided by GeoMod, copes with the typical requirements of geometrically based techniques. However, it is not possible to predict all the forms of feature information which might be required in the future. Hence, the approach taken in Plethora, is that recognition KSs and GeoMod should form a distributed and extensible data base. In such a system, the features/value pairs and the method of feature generation should reside within the KS, but be referred to through the common language of the GeoMod frame names.

kn4 The provision of information rapidly enough to satisfy critical times.

This is closely related to the satisfaction of *kn1* and *kn3* and relies to some extent on the function of the KS requesting information. However, putting this to one side, attempts have been made to reduce possible delays. As a result, geometric information is explicitly represented (e.g. there is a unique boundary representation for each *VOLUME)* and it is of a restricted form. Both geometric and relational representations have been designed to increase the efficiency of a limited number of operations.

However, it is still expected that GeoMod will limit the overall response of Plethora. Techniques to reduce some of these delays are described in Chapter 10.

Communications

cm1 ***To be amenable to implementation in a distributed environment.***

It has been demonstrated, through the use of encapsulation and simple experiments in Chapter 9, that Plethora can operate within a distributed environment.

cm2 ***To have attributes which uniformly support both the real-time and knowledge based requirements of the system.***

The same language and message passing mechanism is used throughout Plethora, whatever the form of communication. The necessary response can be tailored through suitable choices of Prompters/Players and message priority. (Note: It has not been possible to demonstrate this point completely, owing to the simplicity and limitations of the available hardware)

cm3 ***To adapt to changes in the structure of work cell.***

This is achieved through the use of encapsulation, a uniform message passing system and the incremental nature of the BB operation. An illustrative example is the addition of the proximity sensors in Chapter 9.

Further Work and New Avenues

However, there are a number of questions which arise out of the development of Plethora. These, and the questions left unaddressed, form the basis for further investigation.

Partition and Mapping

What is the most efficient partition of the blackboard structure and mapping of Plethora? An inherent restriction on the performance of Plethora is the use of a central conflict resolution scheme, where all KSs return their KSARs to the BB to be evaluated. However, if each processor in the network maintained a list of the *Foci* and *Policies* in their own local levels, then locally generated KSARs could be evaluated locally. This would result in a partial *To-Do-list* within each processor. The function of the BB during Conflict resolution would now be, to interrogate each processor for the highest rated local KSAR, determine the KSAR with the highest overall rating and make it the chosen action.

Model Based Recognition

The initial hope for the 3D recognition technique was that it would lead to a model based recognition scheme, which could be used to verify the results of predicted assembly actions. The position of the object in the assembly could be determined from the solution of it's *PRIMITIVE* relationships. The moments could be derived from a volumetric representation of the assembly, using RPs fitted about the bounding spheres and modified according to the expression for parallel projection error. However, before this can be investigated the jitter problem must be removed.

Path Planner

Further analysis of the path planner needs to be undertaken before it could be employed within a commercial work cell. This would include,

1. Quantitative investigation of the performance and "quality" of paths produced by the planner. This might also be done using varying degrees of approximation to the scene. A further possibility, would be an investigation into the generation of the free space representation by the 3D vision system directly.

2. Modification to the path control used at via points and removing the need for the straight line trajectories. Straight line trajectories at the via points require rapid changes in velocity, and have the disadvantage of increasing wear within the manipulator. A suitable approach might be to use parabolic blend [1] or cubic functions [1] at the via points (given an appropriate increase in Vr).

3. Increasing the accuracy of the model of the payload, by regarding it as being represented as a number of bounding spheres, rather than a single sphere.

Parallel Collision Detection

It has already been determined that collision detection is computationally expensive. However, a simple approach, which would allow collision detection to take place in parallel, could be constructed from a *Farm* architecture. A supervisor process could be used to determine the areas of possible collision (based upon rough sphere intersection tests) and sub-divide these (or parts of these) between the workers of the processor farm (in a similar fashion to that described in Chapter 10 for the implementation of GeoMod server interface).

Parallel External Actions

The structure of the chosen action lists and the concept of *Viable* actions, allows the investigation of techniques which execute a number of actions in parallel (e.g. *Viable* complementary verification KSs or the *Viable* actions of KSs which are known not to interfere). Although, this would be expected to be a feature of such systems, this is not discussed in the literature of Chapter 2.

Confidence

At present, *Confidence* values are an equally weighted sum of 3 elements. Since, *Confidence* values affect the behaviour of the blackboard, and through this the behaviour of Plethora. It would be useful to investigate differences in the relative weights of the overlap values and the nominal separation value. It is the relative weight of the nominal element, which determines the behaviour of the function for values of low Confidence. Hence, successful goals but with low *Confidences*, may be represented as unsuccessful, given an inappropriate choice of weights.

Planning and Acting

A theory of domain independent problem solving known as "Planning and Acting" was developed by McDermott [2]. It regarded an action simply as a planning goal, which required to be evaluated in order that it might be solved. Although, the original work ran into difficulties (and to the authors knowledge never been expanded upon), the approach has significant advantages when applied to uncertain domains where sensed information is available.

The technique reduces the plan network to a pure AND tree, so that each problem has just one reduction, this is expanded via its various sub-problems until they can be executed. It executes each sub-problem's solution until it encounters an error.

Each sub-goal is expanded and executed before the next is attempted. Hence, the choice of which sub-goal is to be solved next, and how this is to be done, can be made a function of previous experience (as is possible with YAMS). It has the further advantage of reducing the system's reliance upon the modelling of the environment, which is known to be computationally difficult [3].

However, it was reported in the original work, that when using this technique the system encountered problems with error correction, since it did not maintain a record of its previous reasoning. Plethora, because of its seamless control, explicit reasoning and its ability to reason about the task during execution, has the necessary prerequisites for the further investigation of this technique. It is not expected that the

technique could be used without modification and yet again, not as a general approach. However, it could provide a useful alternative strategy, when an error has been detected and the modelled information is in doubt.

The technique would also have advantages in the determination of a plan network, prior to it's adoption for continuous operation by the cell. The plan network developed by *Solve-Construction* (which is constrained by structural requirements alone), would be executed under the control of this technique, but would recover from errors in the fashion described in Chapter 9. The resulting network, which finally led to a successful completion, could be used as the final plan network. In this way, the effect of actual parameters of the cell would be learnt rather than modelled. This should simplify the modelling problem and hopefully increase the portability of the assembly descriptions between differently structured work cells.

References

[1] Craig,J.
"An Introduction to Robotics: Mechanics and Control", Addison-Wesley,
ISBN 0-210-10326-5, 89.

[2] McDermott,M.
"Planning and Acting", Cognitive Science, vol.2, p71-109, 78.

[3] Brooks,R.A.
"Intelligence Without Representation", Artificial Intelligence, vol.47, p139-159, 91.

Appendix A

COMPUTING AND CONTROL DIVISION

COLLOQUIUM ON

"KNOWLEDGE BASED ENVIRONMENTS FOR INDUSTRIAL APPLICATIONS INCLUDING CO-OPERATING EXPERT SYSTEMS IN CONTROL"

ORGANISED BY
PROFESSIONAL GROUP C13
(AUTOMATION AND CONTROL SYSTEMS)

ON FRIDAY, 9 JUNE 1989

DIGEST NO: 1989/96

TOWARDS KNOWLEDGE BASED CONTROL OF A FLEXIBILE ASSEMBLY ROBOTIC WORKCELL

I. P. W. Sillitoe

This paper describes the work in progress in the development of an environment for a distributed opportunistic knowledge based control of an assembly workcell. It describes a centralised blackboard based upon a modified form of the architecture in [1], a relational and geometric modeller, and a message based communication mechanism.

Blackboard

There are few general and efficient solutions to problems in the robotic domain, in practice solutions tends to very specific and heuristic in nature (e.g. collision free path planning schemes). A blackboard (BB) architecture allows such partial solutions to be incorporated in a single framework under a knowledge based control, which itself can be made to adapt to the state of the plan or execution [1]. It also provides a simple large grained means of partitioning a system on to a distributed environment. More significantly, it allows at each stage of the planning/scheduling/error correction cycle the use of both local and global information generated during any of the other earlier stages. This was seen as a weakness in the approach taken in [2].

There are a number of modifications made to the original architecture in [1]. These include the addition of blackboard modes which state explicitly the BB's progress through the planning cycle and influence the action of basic Knowledge Specialists (KS). The basic KS's control the form of the inference mechanism which is to be undertaken by the BB and hence, with the use of the BB modes this too can be chosen to be of a form which most appropriate to the current state of the planning process. The evaluation of the KS's precondition now takes place within the KS itself, removing the need for the invocable list in [1]. There has also been a change made to the structure of Focus level decisions. They now each include a termination condition which allows a more efficient method of implementing distributed Strategies and attention focusing [3]. Each of these modifications is directed towards increasing the efficiency of the blackboard's inference mechanisms in a "real time" distributed environment.

An extra level, below the Operation level, is included in the domain plane of the blackboard and is known as the Implementation level. This is used to hold those decisions which control the individual devices in the workcell, rather than the Operation level which is used to hold the generic manipulation goals.

The plan solution is held in the form of an AND/OR graph of decisions which have their terminal nodes in the Implementation level. Control of external events is maintained with the use of a new structure within the BB, the Event-list. This is used by the Scheduling basic KS during the scheduling stage to hold messages which represent the status of external events.

Knowledge Representation

Much of the procedural and specialised knowledge is conained within the KS's

I.P.W. Sillitoe is a lecture in the Department of Electronic and Electrical Engineering, Loughborough University.

of such a system, the shared declarative knowledge is held with the geometric modeller and relational network. Each forms a plane of a hierarchical frame network.

The frame types in the geometric modeller consist of BODY, CHAIN and VOLUME. A CHAIN describes a doubly linked structure of BODY frames, which represents an open linear kinematic chain. The BODY frames form a part/sub-part hierarchy, which is used to represent various parts of a rigid structure and terminate in single VOLUME frames. VOLUME frames describe those volumetric and surface characteristics of primitive volumes attributed to the BODY. They also are organised in a hierarchical fashion, initially by a bounding sphere then by a bounding rectangular parallelpiped and finally a set of polygon surface patches, their edges and vertices. The hierarchy of part/sub-part and multiple volume representations increase the efficiency of many of the basic geometric operations (.e.g. intersection operations). The relational plane of the network provides access to the global information in the system and is formed from INDIVidual, SET and SET Descriptor frames in an inheritance hierarchy. In particular, it is used to describe the relationships between parts in a given construction, their functions and the primitive relationships used within the assemblies. The network allows the user to specify a declarative description of the assembly which the system can decompose into a table of primitive constraints. These are solved numerically by a KS to produce the final positions of parts for certain classes of practical constructions.

Communication

A message passing system is used to communicate between the knowledge based parts of the system and control workcell devices. It utilises a non-blocking SEND, blocking RECEIVE, polling RECEIVE which together with a queued letter box structure produce an actor [4] based environment. The SEND and blocking RECEIVE allow the creation of co-begin structures and so capitalise on the large grained parallelism within the system while the polling RECEIVE is used to implement a method of timeout when communicating with untrustworthy correspondents. The message headers include transaction identifiers and a number of transaction protocols have been evaluated.

Summary

The complete system has been implemented in Forth on a ring a PC's. A vision preprocessor, a collision free path planner and machine sequencing KS's have been developed and tested within the system. Although, the system does allow the evaluation of many new techniques and experimentation with their use within an integrated system, the hardware on which it runs is not powerful enough to produce a real time response. Hence, present work is directed towards building and porting it, to more sympathetic hardware.

References

1. HAYES-ROTH, B. "A Blackboard Architecture For Control", Artificial Intelligence, no.26, p251-321, 1985

2. VAN DYKE PARUNAK, H. "Manufacturing Experience with the Contract Net", Distributed Artificial Intelligence, Research Notes in Artificial Intelligence, Pitman, ISBN 0-273-087789

3. HAYES-ROTH, F. LESSER, V.R. "Focus of Attention in Hearsay-II Speech Understanding System", Proc IJCAI 5, 1977.

4. HEWITT, C. "Viewing Control Structures as Patterns of Passing Messages" Artificial Intelligence, vol.8, no.3, p323-364, 1977.

Appendix B

An Approach to the Programming of Distributed Shared Resources within an Experimental Robotic Workcell.

I.P.W Sillitoe,
Dept. Electronic and Electrical Engineering,
Lougborough University,Loughborough,Leicestershire. LE11 3TU.

Abstract

This paper describes the implementation of programming techniques applied to one form of inter-node communication within the distributed system[1]. The aim of the techniques is to maintain as many of the properties of integration and flexibility found within a single Forth environment, across a set of distributed asynchronous communicating FORTH processes. The approach can be thought of as a variant of Remote Procedure Calls[2] and object orientated programming and is derived from the pragmatic considerations encountered when providing flexible knowledge based services within a distributed system.

Introduction

The programming of robotic systems to act in a flexible manner is complex. This complexity can be regarded as having two distinct roots. The first owing to its physical properties. Such systems are usually distributed, heterogeneous and have timing constraints which must be upheld. The second is concerned with the representation, integration and flexible control of the facilities within the cell. It is with the question of flexible control and integration of a knowledge based service that the paper is concerned.

The system in [1] represents the facilities of the robotic work

cell as a set of asynchronous knowledge sources(KS) whose overall control is through a blackboard structure[3]. The knowledge sources are implemented as actor[4] like forms and communicate via a buffered message passing system. The type of knowledge encapsulated within a source defines the form of interaction which the source can undertake within the community of sources. The knowledge sources which encapsulate the physical facilities (i.e Control the motion of the manipulator) of the cell can be regarded as providing a shared resource to those knowledge sources which contain the techniques represented within the system (i.e How to identify an object or how to carry out a pick and place operation). What follows is an explanation and illustration of a number of techniques, using the implementation details of the camera control knowledge source[1], which highlight many of the problems encountered when managing a shared resource within such systems.

Vision Server

The camera provides the necessary sensory data for a large number of technique KSs which can be used to identify objects and verify actions. During the problem solving cycle, a number of these technique KSs make concurrent access to the camera KS in order to evaluate heuristics which will be used to select the most suitable technique for the current state of the plan. Therefore the design of the vision server must not only be able to cope with the problems of data sets (Image sizes range from 64-256 kbytes) and those problems associated with domain, but it must also maintain coherent responses under the influence of competing requests (see [5] for a full discussion of such problems) and minimise the programming complexity

required of a KS to obtain a service.

Representation

The elements of a transaction are represented as polymorphic objects within the camera KS, that is the Class of a particular object may change during its lifetime. The polymorphic nature of the objects is necessary because of the size of the data structure involved and the limited memory available. All objects are held and managed within the server (see figure 1 for an illustration of the fields found within an object) and reference to any particular object is made via its object token supplied by the server. Objects are owned by the KS whose transaction created them and they protected from destructive

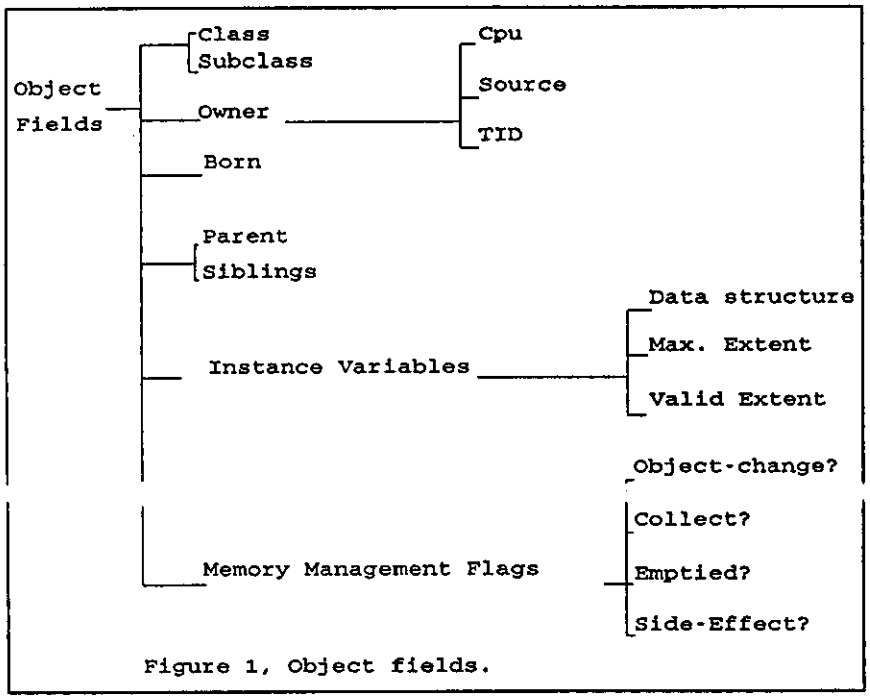


Figure 1, Object fields.

operations by other KS's. They form an inheritance hierarchy in the usual manner (see figure 2) via their class fields while the subclass field acts to modify the methods action and it is the change in this field which gives the object its polymorphic characteristics.

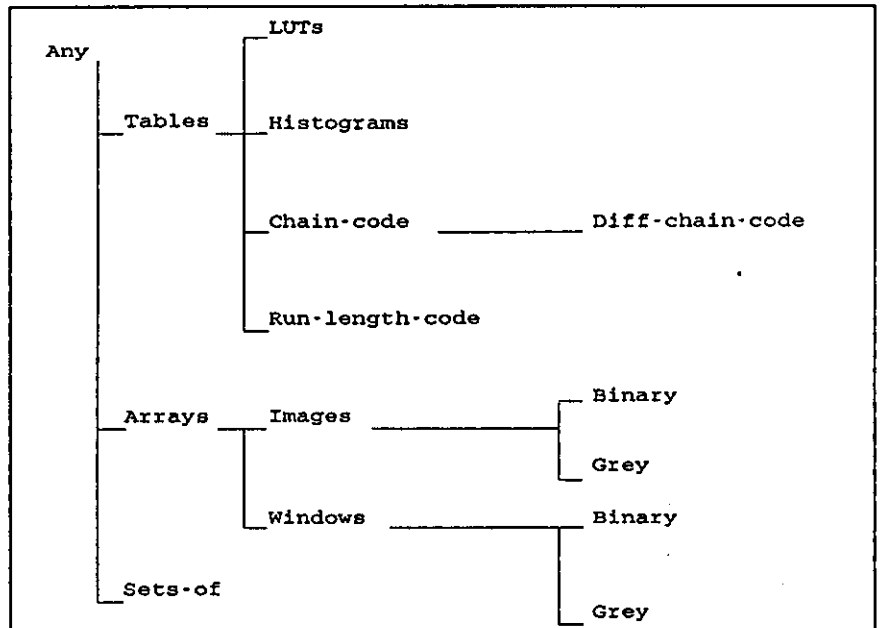


Figure 2.,Method Inheritance between Object Classes.

All objects are also part of a dependency hierarchy which is used by the implicit memory management and garbage collection(GC) of the server. The implicit nature of GC reduces the complexity required by a KS when requesting a service, while also endeavouring to make maximum use of the memory available to the server. At the top of this hierarchy is All-Objs (see figure 3) which is a permanent object of class scope of scopes. A scope holds the objects created within a particular transaction. A number of scopes can exist at the same time allowing separate KSs to carry out what seem to the KSs to be,

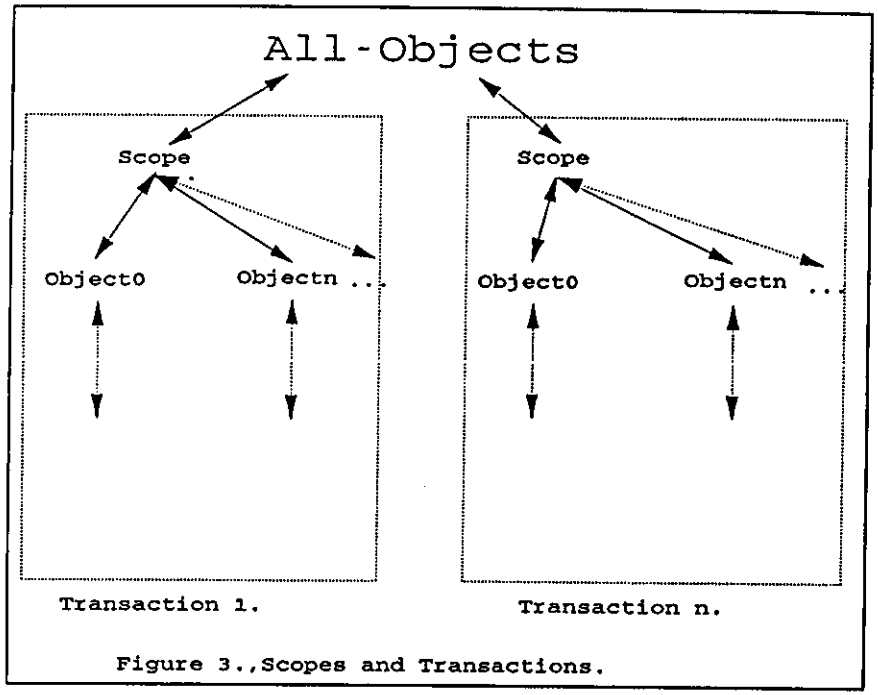


Figure 3.,Scopes and Transactions.

concurrent operations. A new scope is created or an existing scope is made the current scope, at the commencement of the processing a new message. If an object makes reference to data outside its scope (such as where a number of objects share the same source image), and if the method to be used is destructive, a copy of the original object is made in the local scope. The dependency hierarchy is used to indicate which particular object was generated as a result of which parental data and is used to limit the possible interactions between competing KSs. The function of the GC is to maintain the truth of such relationships during the progress of a transaction, that is the same method should be able to be applied to any parental data and still produce the same siblings.

The local instance variables (see figure 1) are used to define

the maximum dimensions of the object data structure and the valid region of that structure on which the methods can operate. For example after some methods, such as convolution with a 3x3 mask, the extreme pixels are nolonger defined in the processed image and so as a result the valid region is smaller than the maximum extent of the data structures. This reduces the amount of house keeping required to be undertaken by the requesting KS.

Garbage Collection

The server provides automatic garbage collection to simplify programming and ensure maximum usage of the memory available. Its action is triggered by a change in state in the memory management flags within an object. These flags are set by the methods and indicate the change in the objects data structure or subclass which may allow garbage collection to take place. Garbage collection takes place when,

1. A KS requests the contents of a particular object to be sent to it,indicating the end of the objects useful life. When this occurs the siblings are also nolonger valid and so the object, all its siblings and their siblings are collected. This situation is indicated by the method setting the Collect? flag.

2. A transaction is completed the contents of the corresponding scope are removed.

3. An objects subclass changes (say grey scale to binary image) the contents of the object remains valid, however, the validity of

siblings is dependent upon the previous subclass of the object and so they are removed.

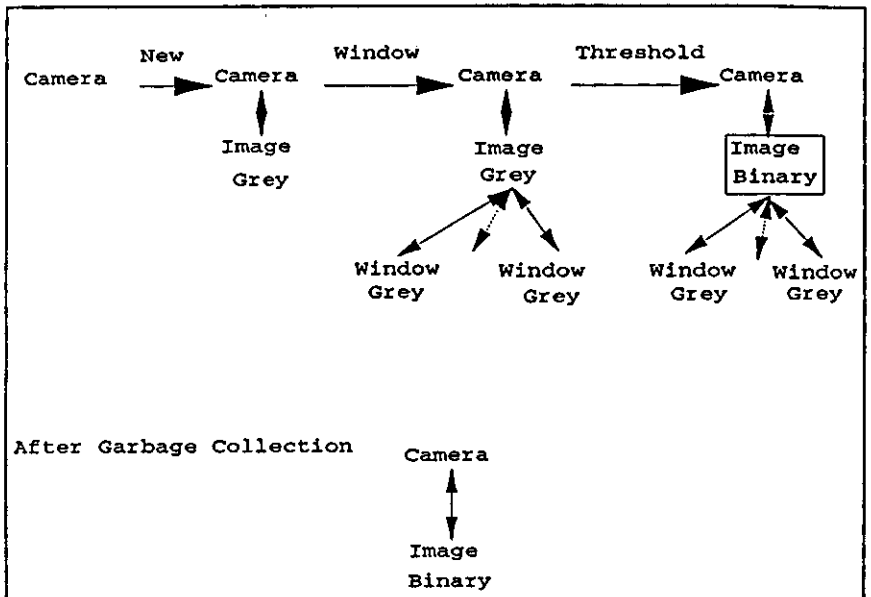


Figure 4.,Changes in Object Subclass.

4. Under certain circumstances the data structure can be deleted by the action of the method (i.e Certain forms of chain encoding algorithm), when this occurs it is indicated to GC by setting the Emptied? flag within the object. When the garbage collector is called it removes this object and moves its siblings to the parent of the emptied object (see figure 5).

However, there are still opportunities during a transaction when garbage collection could be performed but which are not detected by the rules above. These situations arise when objects are formed as

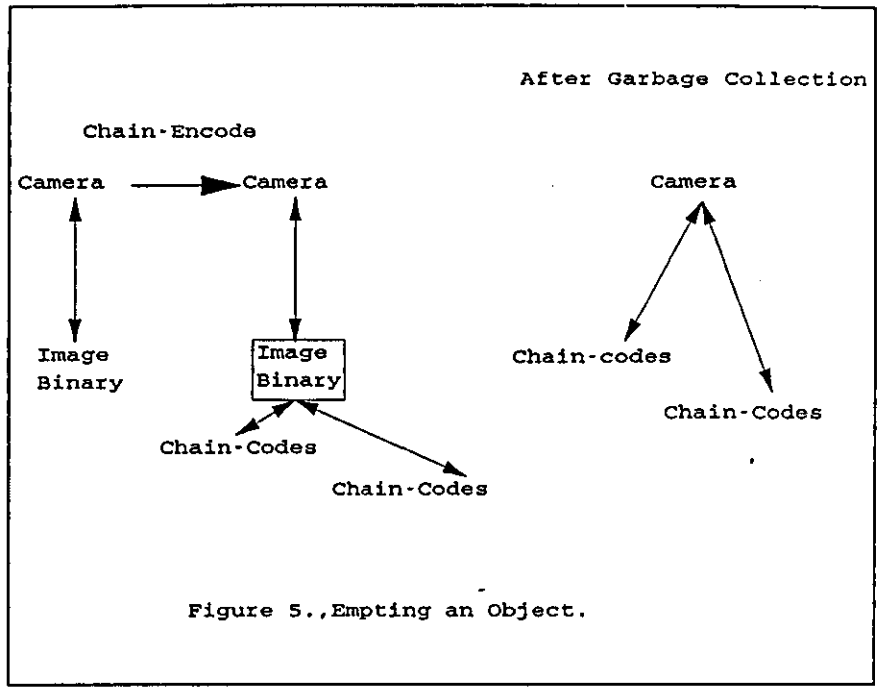


Figure 5.,Empting an Object.

side effects of the eventual goal object. Such is the case when enhancing the contrast of an image by the use of histogram equalisation, where during this process a LUT and Histogram are generated and have no further relevance once the contrast of the image has been modified. Since the detection of these situations would require an understanding of the intention of the KSs action they cannot be implicitly processed by the GC. In order to cope with these eventualities there is a construction recognised by the message interpreter within the server known as FORM. FORM takes as its arguments an object and an expression, during the evaluation of the expression it marks all objects generated as side effects ,except for the goal object specified in its arguments. When GC is called it removes all the objects and siblings marked as side effects leaving

only the goal object within the scope. In this way the calling KS can ensure that only necessary objects are maintained throughout a transaction and the syntax of FORM underlines the intention of the transactions code.

Message Interpreter

The function of the message interpreter is to remove the messages from the servers message queue, interpret the user portion of the message, determine the validity of the request, apply the relevant sequence of methods and finally return the appropriate response. The message interpreter maintains a coherent server response by forcing the execution of an individual message is be serial and within the scope of the transaction[5]. However, it also allows message requests from competing transactions to be processed in any order and so still provides a degree of concurrency.

The message interpreter is implemented as a set of forward chaining rules similar in construction to FORPS[6] but where the flow of execution is specified within the body of the rule. The main bulk of the rules implement an efficient form of recursive descent interpretation which defines the syntax and performs general error management. Rules for individual methods are simply added to the existing table of rules for them to be included in the server's repertoire of actions.

The inter node language used to communicate between the KS's has a LISP like syntax and specifies the required action in terms of assertions and goals in a similar way to PROLOG. This was chosen in preference to less restricted FORTH form in order that more precise syntax checks could be made on the message requests and so reduce

ambiguity. These were found to be significant problems in earlier versions which simply copied the contents of the message to TIB and evoked the FORTH interpreter.

Implementation

The server is written in FF on a IBM AT with image frame grabber and uses the MSDOS BIOS routines to allocate/deallocate the data structures. The object tokens are dynamically allocated within FF from a stack of pointers, which point to preallocated dictionary areas. The message passing services are provided by MP (a program written by the author) and communications are via point to point serial contacts through a message switch.

Summary

The problem domain outlined above highlights a number of problems which do not normally arise in a single FORTH environment (memory management, garbage collection and multiagent access to shared resources) but which are endemic in large scale and multiprocessor applications. At present there is no support within the core definitions of FORTH which could be used to address such problems (Note: even basic multitasking which appears in most implementations, is undefined within a standard). This paper describes novel techniques, written in FORTH, which have been used to solve these problems in a distributed environment and so might provide a discussion point for further extensions to FORTH for use multiprocessor environments.

References

[1] Sillitoe,I.P.W.
Towards Knowledge Based Control of a Flexible Assembly Robotic Workcell.,IEE Colloquium Knowledge Based Environments For Industrial Applications,June 1989.

[2] Birrell,A.D. Nelson,B.J.
Implementing Remote Procedure Calls.,ACM Trans. on Computer Systems,Vol.2,No.1,February 1984.

[3] Englemore,R. Morgan,T.
Blackboard System,Addison-Wesley,ISBN 0-021-17431-6.

[4] Hewitt,C.
Viewing Control Structures as Patterns of Passing Messages, Artificial Intelligence,No. 8,p323-364.

[5] Bernstien,P.A. Goodman,N.
Concurrency Control in Distributed Database Systems, Computing Surveys,Vol 13.,No.2,p185-221.

[6] Matheus,C.
The Internals of FORPS: An Forth-Based Production System, The Journal of Forth Applications and Research,Vol 4,No.1, p7-27.

Appendix C

The solutions of all the stationary position relationships take the same form, and so for brevity only the derivation of the simultaneous solution of three *Star* equations is given.

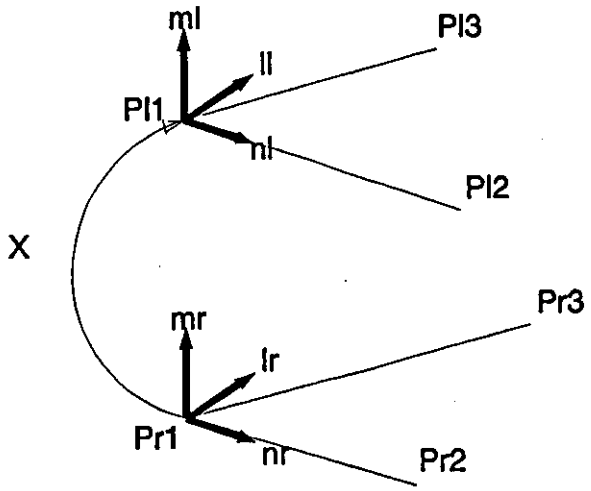


Figure c.0 The Transformation X for 3 Star relationships.

Simultaneous Solution of Three Star Relationships

Given the abbreviated form of the homogeneous transform in Chapter 6, we can write T_{ri} the initial position, T_{li} the final position and X the required transformation as

$$X = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad Tli = \begin{bmatrix} ? & ? & ? & P_{li} \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad Tri = \begin{bmatrix} ? & ? & ? & P_{ri} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the three star relationships as

$$\begin{bmatrix} ? & ? & ? & P_{l1} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} ? & ? & ? & P_{r1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 1$$

$$\begin{bmatrix} ? & ? & ? & P_{l2} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} ? & ? & ? & P_{r2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 2$$

$$\begin{bmatrix} ? & ? & ? & P_{l2} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} ? & ? & ? & P_{r3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 3$$

where P_{l1}, P_{l2}, P_{l3} and P_{r1}, P_{r2}, P_{r3} are the triangle of points in the 3 Star relationships.

Subtracting equation 1 from 2

$$\begin{bmatrix} ? & ? & ? & P_{l2}-P_{l1} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} ? & ? & ? & P_{r2}-P_{r1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore

$$[\, P_{l2} - P_{l1} \,] = [\, l\ m\ n \,].[\, P_{r2} - P_{r1}]$$ 4

This can be normalised as

$$n_l = [\, l\ m\ n \,].n_r$$ 5

where n_l = unit($P_{l2} - P_{l1}$), n_r = unit($P_{r2} - P_{r1}$).

By subtracting 1 from 3

$$[\, P_{l3} - P_{l1} \,] = [\, l\ m\ n \,].[\, P_{r3} - P_{r1}]$$ 6

By forming the cross product of 5 and 6

$$n_l \times [\, P_{l3} - P_{l1} \,] = [\, l\ m\ n \,].(n_r \times [P_{r3} - P_{r1}])$$

which can be normalised as

$$m_l = [\, l\ m\ n \,].m_r$$ 7

where m_l = unit (n_l x ($P_{l3} - P_{l1}$)); m_r = unit (n_r x ($P_{r3}-P_{r1}$))

by combining 6 and 7

$$[\, l_l,\ m_l,\ n_l \,] = [\, l\ m\ n \,] . [\, l_r\ m_r\ n_r \,]$$ 8

where $l_l = m_l \times n_l$; $l_r = m_r \times n_r$

from 1 and 8

$$\begin{bmatrix} l_l & m_l & n_l & P_{l1} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} l & m & n & p \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} l_r & m_r & n_r & P_{r1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence the solution for X is

$$X = \begin{bmatrix} l_l & m_l & n_l & P_{l1} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} l_r & m_r & n_r & P_{r1} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Appendix D

A Multiview Robotic Vision System For Efficient Object Recognition.

I.P.W. Sillitoe
Loughborough University, Dept. of Electronic & Electrical Engineering, U.K.
J. Edwards
Loughborough University, Dept. of Computer Studies, U.K.
A.H. Falkner
Coventry Polytechnic, Dept. of Electrical Electronic & Systems Engineering, U.K.

0.1 Abstract

This paper describes an experimental model based vision object recognition system designed for use in real time robotic assembly. The system utilises three orthogonal views to generate a volumetric representation of the object, from which it determines the equivalent of the object's inertial tensor and volume. The matrix and volume form a tuplet which is used to identify and determine the orientation of objects. The tuplet provides a compact and flexible representation for the object and is efficiently derived from the rectangular parallelepiped (RP) representation generated by the vision system.

0.2 Overview

Object recognition is of particular importance in assembly operations as it provides a means of identifying workpieces and determining a part's position and orientation. If a suitable model based representation is employed, it can also serve to verify the assembly actions by modelling these changes in the workpiece and matching the predicted model against the scene. Before true automated assembly is possible it is essential to have a robust representational model and general purpose technique for determining the orientation and position of 3D objects for a large class of industrial parts [1].

0.3 Introduction

Previous work [2] can be categorised as 2D recognition, where a single image is used, 2.5D where some depth information is available such as in stereo vision and 3D where a number of views are combined to produce view-independent volumetric representations of the scene. The latter is the preferred form for object identification [3], since it results in a single object centre representation, in contrast to the 2D systems in which a number of views must be represented. The design of 2D systems is further complicated since the means by which the views for a 2D system are chosen and how they are to be represented, are not straightforward and the practical limitations made upon these choices constrain the number of view points from which the object is recognisable.

A multiple view system not only reduces the difficulty associated with the interpretation of the one to many transformation involved in a 2D system, but it also overcomes

the time constraints that are often associated with the convolution of images in stereo vision systems. The choice of three orthogonal views provides a compromise between simplicity in terms of hardware requirements and the computational efficiency of the generation of the volumetric representation.

0.4 Previous Work

Much of the model based recognition work uses a geometric model of the object to generate a 2D representation of the scene from a given view point, then determines a set of features and finally tries to match these against the feature vector of the image (e.g. [1])

A different approach described by Potmesl [4] generates volumetric models from multiple views and uses a boundary representation, based upon bicubic parameter patches, to represent the object and form a 3D representation of the scene. It then heuristically searches for overlapping segments in the representation to produce a description of the connected patches. This technique is applied iteratively until the whole scene has been covered.

These approaches tend to be computationally expensive and so have limited application in real time and/or do not yield features suitable for an efficient means of object recognition. In order to produce an efficient 3D model based recognition system we must choose a form of representation and compatible method of identification. The method described here attempts to do this by combining an efficient means of generating a volumetric representation and utilising a representation which can be simply manipulated to provide a useful set of compact global features.

0.5 Methodology

The calculation of the tuplet takes place in two stages. Firstly, the generation of the volumetric representation of the object in terms of a number of non-overlapping rectangular parallelepipeds and secondly, the derivation of the tuplet from the set of RP's.

0.5.1. Generation of the Volumetric Model

The RP's which represent the object are coded as in figure 1. They are derived in real time using a modified version of the method found in [5], which is implemented in parallel on a multi-transputer system. The system uses three orthogonal views of the object (see figure 2), which are assumed to represent parallel projections in each of the viewing directions. These views are then coded as rectangles and then swept in the respective viewing direction to form RP's. The intersection and merging of these codes produces the final list of RP codes from which the tuplet will be derived.

The initial experiments used 128x128 8 bit grey scale

images of the object. These were thresholded using a variable thresholding technique based upon [6]. The generation of RP's took between 0.25 and 0.5 seconds, well within the critical times of most processes within an assembly workcell. (see figure 3. for an example).

0.5.2 Generating the Centralised Moment Matrix

If we assume that RP's have a uniform density of 1, then the sum of the RP volumes will result in a psuedo mass for the object. If the individual psuedo masses are taken to act at the mid-point of the RPs we can generate the moments and centralised moments of order p+q+r as below.

$$M_{pqr} = \sum_{i=1}^{K} \left[x_i^p . y_i^q . z_i^r . vol_i \right]$$

centralised moments

$$\mu_{pqr} = \sum_{i=1}^{K} \left[(x_i - \bar{x})^p . (y_i - \bar{y})^q . (z_i - \bar{z})^r . vol_i \right]$$

where

[x,y,z] corresponds to the coordinates of the midpoint of the individual RP volumes.

K is the number of RP's representing the volume.

vol_i is the volume corresponding to the ith RP.

This gives rise to the matrix $\underline{U}$, the second order centralised moments required for the tuplet representation, which is used to derive 3D moment invariants [7] and it's principal axes.

$$\underline{U} = \begin{bmatrix} \mu200 & \mu110 & \mu101 \\ \mu110 & \mu020 & \mu011 \\ \mu101 & \mu011 & \mu002 \end{bmatrix}$$

The volume is simply found by summing all the individual RP volumes used to represent the object.

0.5.3 Moment Invariants

Sadjal and Hall [7] have generalised the results of 2D moment invariants [8] by linking the 3D moments to ternary quantics and so produce a set of 3D moment invariants. These are invariant under size, orientation and position and are shown below,

$$\frac{J_{1\mu}^2}{J_{2\mu}} \quad \text{and} \quad \frac{\Delta_{2\mu}}{J_{1\mu}^3}$$

where

$$J_{1\mu} = \mu_{200} + \mu_{020} + \mu_{002}$$

$$J_{2\mu} = \mu_{020} \cdot \mu_{002} - \mu_{011}^2 + \mu_{200} \cdot \mu_{002} - \mu_{101}^2 + \mu_{200} \cdot \mu_{020} - \mu_{110}^2$$

$$\Delta_{2\mu} = \det [\underline{U}]$$

The invariants can be readily obtained from $\underline{U}$ and are used as features when identifying the object.

0.5.4 Principal Axes

Given that object's reference frame origin is at the

centre of mass, when this frame is aligned with the principal axes the tensor a becomes diagonal matrix, ($\underline{U}^*$), whose elements correspond to the principal moments of the object. The value of the principal moments are given by the eigenvalues of the original tensor. Hence, the rotational matrix ($\underline{R}$) which is required to align the object with its principal axes can be determined from,

$$\underline{R} = \underline{U}^* . \underline{U}^{-1}$$

where $\underline{U}$ represents the objects present position and $\underline{R}$ represents the orientation of the object with respect to principal axes.

0.6 Results

The initial experimentation performs two functions, firstly to verify the operation of the system and secondly, to determine the effectiveness of the tuplet for use in the recognition of objects.

0.6.1 Verification

Two experiments were used to verify the operation of the system. Firstly, a white ball was placed at differing positions within the working volume of the system (this consisted of a 15 cm cube at the intersection of the camera's viewing space), and it's tuplets and moment invariants calculated. The second determined the tuplets and invariants of a white box rotated, at multiples of 45°, about a fixed point. Together, these were used to test independently the system under translation and rotation. The results are shown in table 1 and 2.

0.6.2 Classification

The effectiveness of the tuplet in object recognition was illustrated by comparing a set of chess pieces and set of randomly selected assembly parts. The tuplet for each of the elements was used to form a three element feature space consisting of the volume and the two moment invariants. The chess pieces were of similar shape and had no holes while the assembly pieces were irregular, had holes and reflective surfaces. These sets were used to evaluate the ability for the system to distinguish between similar objects and separately to cope with problems associated with typcial workpieces. It was found that the chess and assembly pieces could be identified using a simple minimum distance classifier. However, it was also noticed at certain orientations that the holes in the assembly parts were hidden from the view of all three cameras and so not constructed in the volumetric representation. This gave rise to a greater variation in the position of the workpieces within feature space than occurred with the chess pieces. Although, they still remained linearly separable. Similar effects due to variations in the illumination resulted from bright spots on reflective surfaces and shadows.

0.7 Discussion

In general, the variation of the feature vectors for a

single object were due to three factors firstly, the loss of gross features which were hidden from the cameras, secondly, the effects of illumination and finally a combination of the effects of quantisation and camera axes misalignment.

Obscured features are a common problem in most vision systems and is often overcome by associating a number of areas in feature space with a particular object, one for each blind spot.

The lack of control over the illumination of the object usually results in a loss of local detail. This is illustrated by the variation in the magnitude of the features in the ball and box experiments shown in table 1 and 2, and the form of the typical thresholded images shown in figure 4. However, since the method of classification is based upon an analysis of global features, and distortions due to illumination must be correlated in all three image if they are to appear in the final volumetric representation, the technique tends to be tolerant of the effect of minor variations.

During experimentation it was found, not unsurprisingly, that the system is particularly sensitive to misalignment of camera axes and this is aggravated by the low image resolution used in the initial experiments. The relative displacement between the camera planes was measured using the orthogonal views of a white ball. The centre of the ball was used as a reference between adjacent image planes and the displacements, required to align the reference coordinates, offset the image plane coordinates and so compensated for the misalignment. An error in this calibration overshadowed the variation in the feature vectors due to the other effects. Further experiments with images of 256x256 showed an amelioration of this problem and improved the correlation between the measured and calculated rotations.

0.8 Conclusions

The initial experiments illustrate that the system can successfully recognise single objects from a given set and at present less successfully determine their orientations. The time taken to perform this process takes between 0.25 and 0.5 second and is within most workcell critical times and the technique tolerates minor variation in lighting conditions. However, it is apparent from limited experiments that the performance of the system can be improved by increasing the resolution of the original images. For further work into the identification of multiple object scenes this will be essential and will incur a corresponding increase in the processing time.

Table of Contents

0.1 Abstract

0.2 Overview

0.3 Introduction

0.4 previous Work

0.5 Methodology

0.5.1 Generation of the Volumetric Model

0.5.2 Generating the Centralised Moment Matrix

0.5.3 Principal Axes

0.6 Results

0.6.1 Verification

0.6.2 Classification

0.7 Discussion

0.8 Conclusions

0.9 References

Figure 1. The Coding Scheme used to represent a RP.

Figure 2. The generation process of the volumetric representation

Figure 3. Illustrative examples of volumetric representation's generated by the system

(a) pencil sharpner (b) Mug

Figure 4. Typical thresholded images from the box experiment.
(a) top view (b) side view (c) front view

Table 1. Feature values for 6 positions of the Ball

Table 2 Feature values for the first 4 rotations of the Box.

0.9 References

[1] Bir Bhanu and Thomas Henderson, "CAGD Based 3-D Vision", Proc. IEEE Int. Conf. on Robotics and Control, March, 1985, p411-417.

[2] Marr, D. "Representing Visual Information", Computer Vision Systems, A.R. Hanson and E.M. Rieseman, Academic Press, Orlando, Fl. p61-88.

[3] Chin, R.T., Dyer, C.R., "Model Based Recognition in Robot Vision", Computing Surveys, March 1986, pp67-108.

[4] Potmesil, M. "Generating Models of Solid Objects by Matching 3D Surface Segments", Proc. 8th IJCAI, Aug. 1983, p1089-1093.

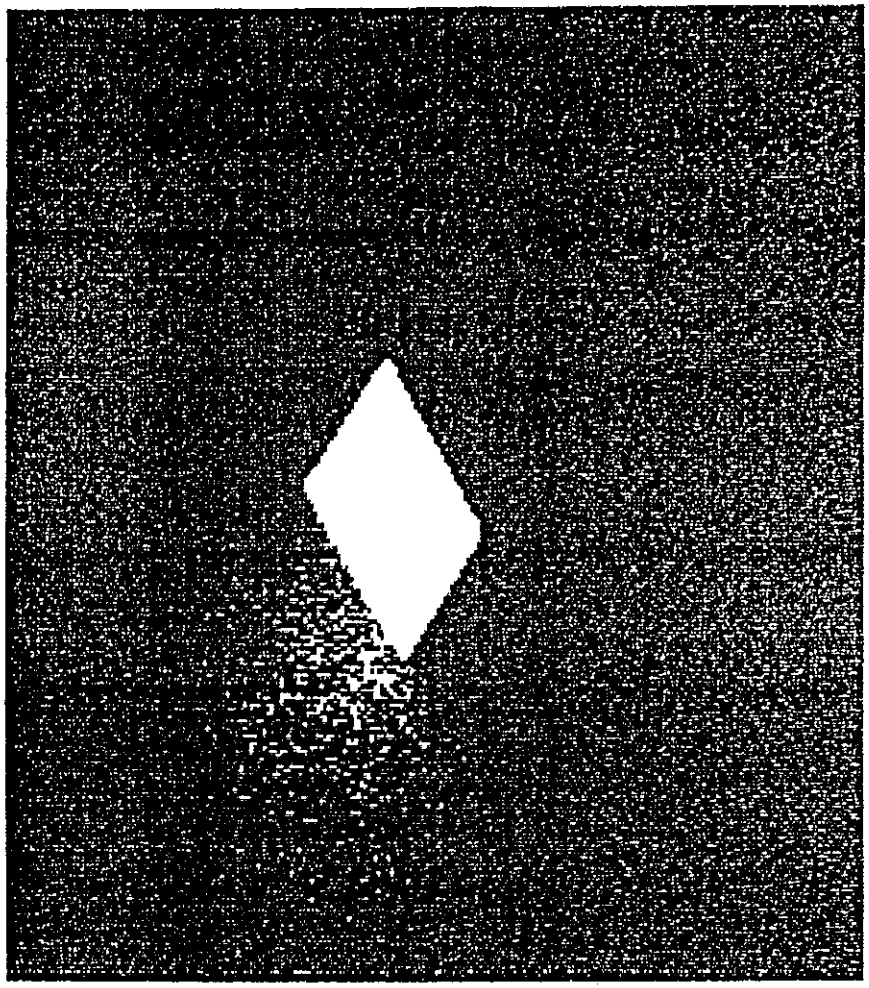
[5] Kim, Y.C. Aggarwal, J.K. "Rectangular parallelepiped coding: A volumetric representation of three dimensional objects". IEEE Journal of Robotics and Automation, vol. PAMI-7, no. 6, Nov. 83.

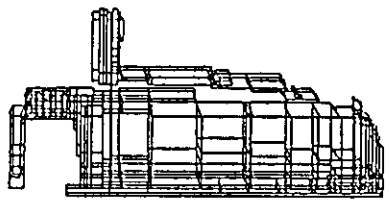
[6] Kittler, J. and Illingworth, J. "Threshold Selection Based upon a Simple Image Statistic", Computer Vision, Graphics and Image Processing, vol.30, 1985, p125-147.

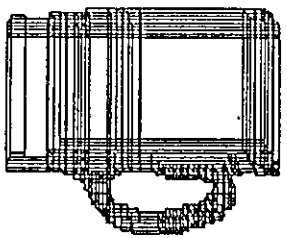
[7] Sadjadi, F.A. "Three Dimensional Moment Invariations", IEEE PAMI-2 No.2, March 1980, p127-135.

[8] Kim, Y.C. and Aggarwal, J.C. "Rectangular Coding for Binary Images", Proc. IEEE Conf. Computer Vision and Pattern recognition, 1983, p108-113.

Volume/(voxels)	$\Delta_{2\mu}/J_{1\mu}^3$	$J_{1\mu}^2/J_{2\mu}$	Measured Rotation	Calculated Rotation
5054	3.11×10^{-2}	3.5	0°	5°
5881	3.88×10^{-2}	3.78	45°	53°
5039	4.80×10^{-2}	3.48	90°	93°
5095	4.01×10^{-2}	3.01	135°	142°







Volume/(voxels)	$\Delta_{2\mu}/J_{1\mu}^3$	$J_{1\mu}^2/J_{2\mu}$
2369	3.23×10^{-3}	3.9
2492	2.75×10^{-3}	3.6
2390	2.5×10^{-3}	3.6
2394	3.89×10^{-3}	4.0
2296	2.08×10^{-3}	3.3
2420	2.94×10^{-3}	3.6

Appendix E

THE DESIGN OF A REAL TIME THREE DIMENSIONAL VISION SYSTEM FOR OBJECT IDENTIFICATION

Ms Janet Edwards and Mr Ian Sillitoe
Dept of Computer Studies Dept of Electronic and Electrical Engineering

University of Technology
Loughborough
LE11 3TU

ABSTRACT

The paper describes the design and analysis of a transputer based application, written in Occam-2 and implemented on a network of transputers. The system generates volumetric representations of industrial objects in real time from a set of multiple views and forms a testbed for the investigation of various identification techniques to be used within an experimental industrial robotic workcell.

The paper outlines the algorithm, its implementation and the efficacy of various design techniques used to increase the response of the system. It also makes recommendations for further work and useful tools to assist in writing efficient Occam code more effectively.

1. INTRODUCTION TO THE DOMAIN

Three dimensional representations of work pieces are an essential part of many robotic applications and boundary representations are often used with robotic geometric modellers. However, when using three dimensional representations for model based identification of objects from camera images, it is more appropriate to use volume representations wherever possible. Such representations vastly reduce the complexity of determining the inverse of the one to many transform associated with 2D representation of a 3D scene since they are view-independent [1].

Rectangular parallelepiped (RP) coding is a form of volumetric representation. It has a single primitive, the rectangular parallelepiped (x,y,z;w,h,d), which can be represented as in figure 1.

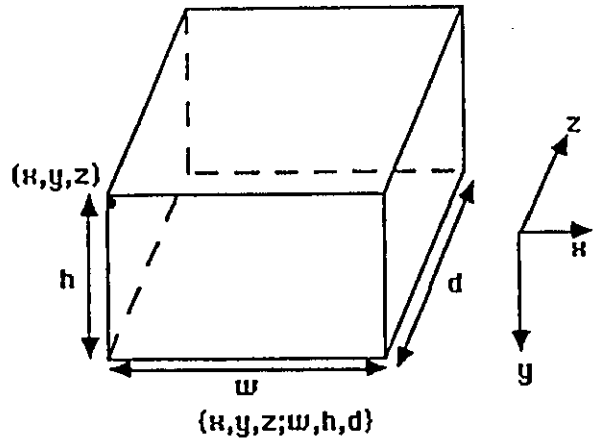


Figure 1 The coded rectangular parallelepiped

It has been shown that RP coding is more efficient in terms of storage requirements and execution time required for its generation than present similar volumetric representations [1]. It provides simple and efficient intersection and merging operators and can readily be inverted to produce a representation of the unoccupied space about the object [2]. These characteristics make RP coding an especially attractive candidate for spatial representation in robotic applications and in object identification [3].

However, even with the intrinsic simplicity of RP representation, sequential implementations do not run fast enough for use in an industrial workcell. Hence, a modified version of the original sequential algorithm [4] was implemented on a dedicated network of transputers in order to achieve the necessary response.

2. OUTLINE OF THE ALGORITHM

The modified version of the original algorithm contains extra preprocessing stages (linear and non-linear filtering) and detailed changes to the individual stages, however, the necessary stages are outlined below, in steps (a) to (e) and summarised diagrammatically in figure 2.

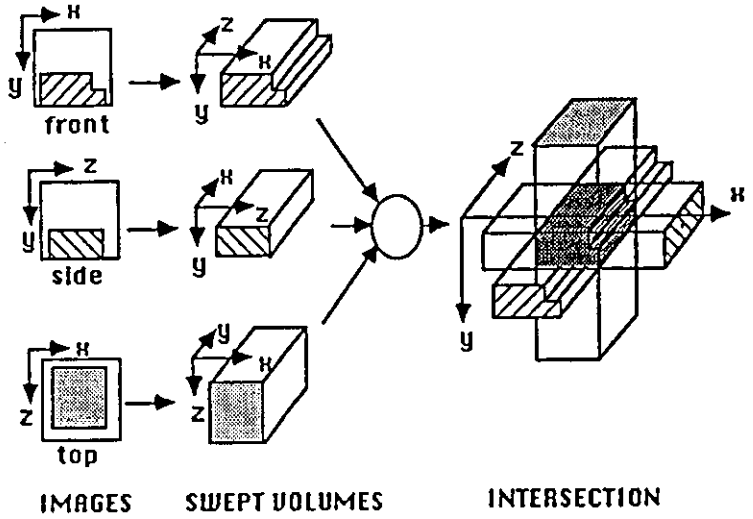


Figure 2 An overview of the method

a) Thresholding

The three grey scale images were thresholded to produce black and white images, using a variable thresholder based upon the simple image statistic technique [5].

b) Run Length Coding

For each horizontal line of each image the start and run length of all the black segments were found.

c) Two Dimensional Rectangular Coding

From the run length codes of adjacent lines rectangles of maximum area were grown to produce a 2D rectangular coding of each image. Figure 3 shows the method by which rectangles are grown and figure 4 shows a typical result of this process.

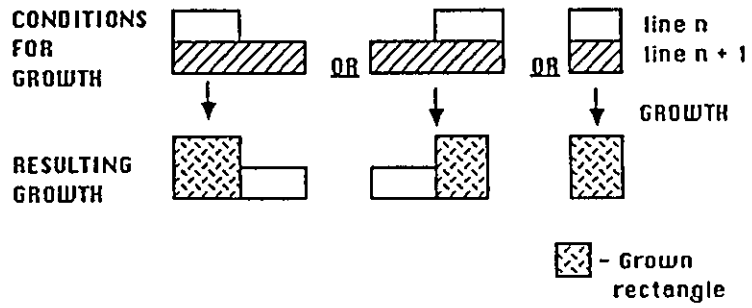


Figure 3 The conditions for the growth of rectangles

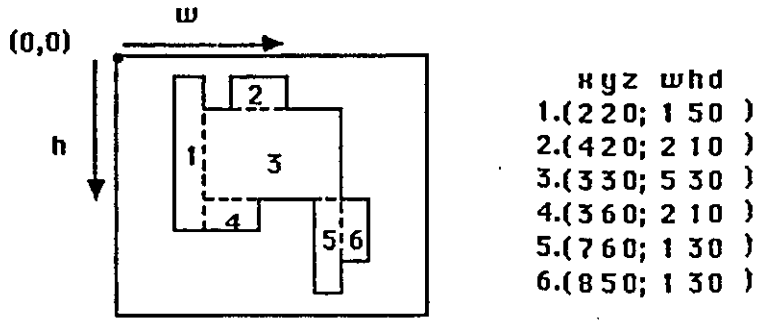


Figure 4 The result of two dimensional RP growth

d) Sweeping

3D RPs were generated from each rectangle by sweeping the 2D RPs along their respective viewing directions.

e) Intersection and Merging

The intersection of the 3 sets of 3D RPs produced an intermediate set of 3D RPs. This process was completed in two stages; initially the top and front views were intersected and then each result of this intersection was then intersected with the side view (see figure 5). As this process can produce pairs of RPs with a common face, this intermediate set was then checked to see whether any of the parallelepipeds could be merged and so remove redundancy from the final representation.

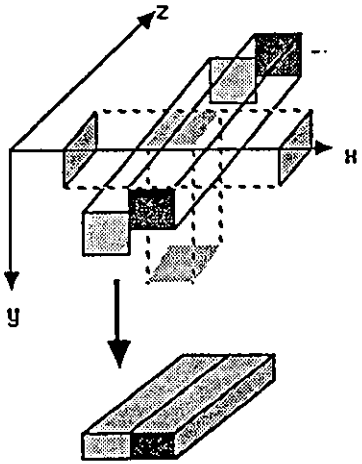


Figure 5 Intersection of swept volumes resulting in RPs with a common face

3.IMPLEMENTATION

The final subdivision of the algorithm was made in such a way that it exhibited both geometric and algorithmic parallelism. The same initial operations needed to be made on the three independent data structures (i.e. the images) and so they could be implemented in parallel. These stages constituted the processes up to and including the generation of the 2D RPs. In so doing this exploited the application's inherent geometric parallelism.The latter stages of the method were more suitably partitioned using algorithmic parallelism. Figure 6 shows how the whole system was partitioned.

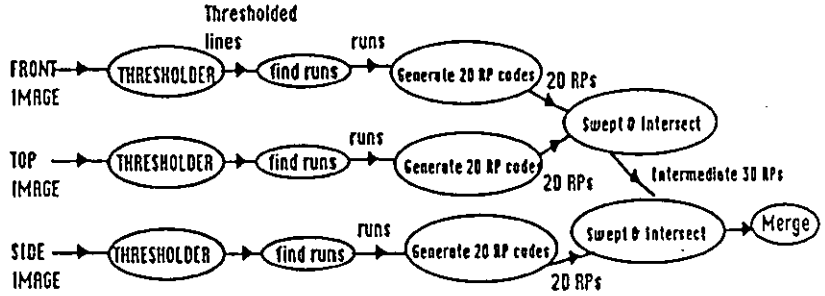


Figure 6 The overall partitioning of the system

The detailed partition and mapping of the processes was undertaken as a result of the analysis of the initial process timings, see figure 7a. An idle time monitor, (obtained from the transputer centre at Southampton) produced much of the information which identified the bottlenecks within the system and as a result a number of techniques were investigated to reduce their effects.

These techniques included:-

a) Buffering between the algorithmically decomposed stages.
b) Prioritisation of all processes which use links so that processors are not forced to wait for data.
c) The transfer of large data packets over the links.

4. RESULTS

The results and timings of the system were based on the use of 128* 128 images of a desk pencil sharpener, a cup and a hole punch of similar complexity to those found in the original paper [4], which describes the sequential method. Figure 8 shows the reconstructed representations of the pencil sharpener and the cup from the generated RPs.

The execution profiles are shown in figure 7 and illustrate the effect of the changes which were made to increase the performance.These are augmented by a comparison with a single transputer implementation and the final transputer network, the results of which are summarised in Table 1.

Note

As the idle time monitor had to be the only process to be run at a high priority it was not possible to use it when communication between the transputers was also at a high priority. Hence the diagram shown in figure 7b contains no figures for the percentage utilisation.

Similar comparisons are shown in Table 2 but in this case they illustrate the effects of the use of on-chip memory. Table 3 shows the differences in execution times when using and not using the PRI PAR construct to overlap communication and computation. All timings are in seconds and where efficiency is said to be,

$$\text{Efficiency} = \frac{\text{(Time taken on one transputer)}}{\text{(Time taken on n transputers)*n}} * 100$$

It was found that there was only a slight difference when using link transfers of a row or a complete image at a time and these were 34% and 33% of the total time taken to send a complete image a pixel at a time.

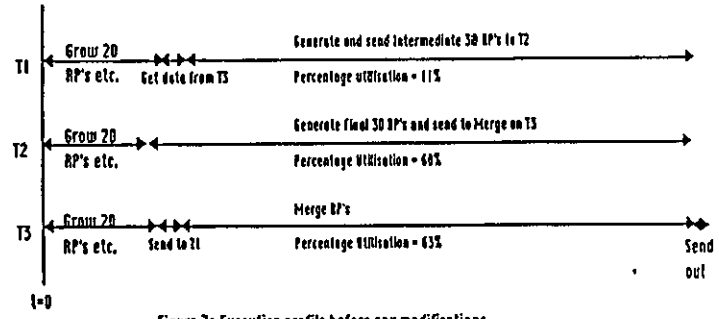


Figure 7a Execution profile before any modifications

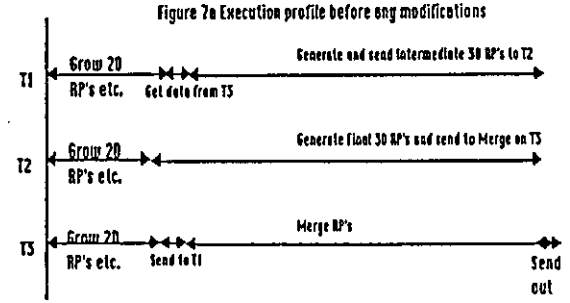


Figure 7b Execution profile after modifications

Note: The diagrams are drawn to the same time scale

Figure 7 Execution Profiles

	Pencil Sharpener	Cup	Hole Punch
One Transputer	0.87 sec	0.40 sec	0.61 sec
3 Transputers	0.47 sec	0.25 sec	0.33 sec
Efficiency	63%	54%	61%

Table 1 Where on-chip memory was used.

	Pencil Sharpener	Cup	Hole Punch
One Transputer	1.21 sec	0.56 sec	0.86 sec
3 Transputers	0.75 sec	0.36 sec	0.52 sec
Efficiency	54%	52%	55%

Table 2 Where no on-chip memory was used.

	Pencil Sharpener	Cup	Hole Punch
3 Transputers PRI PAR	0.47 sec	0.25 sec	0.33 sec
3 Transputers	1.92 sec	1.36 sec	1.26 sec

Table 3 A comparison of the use of the PRI PAR construct.

5.DISCUSSION

5. 1. Buffering

The results shown in figure 7a illustrate the response of the system with no buffering. It was found from these results that the algorithmic decomposition stages of the application were not balanced. In particular, processor T1, which determined the intersection of the top and the front images, was only used for 11% of the time available. In order to avoid the links waiting on the processor or vice versa, link communication needs to be decoupled from computation. In the case of the T1 processor it was necessary to provide a buffer which would contain enough RPs to ensure that this processor never had to wait, while communication of an intermediate RP took place.

In the case of the other processors each process was embedded between a pair of buffers. Experiments were performed to vary the size of these buffers and it was found that the optimum performance was obtained when each buffer would hold one RP. Figure 7b shows the response of the system after addition of the buffers.

5.2. Overlapping communication and computation

Once buffering has been introduced between processes running on different processors it was necessary to use a PRI PAR construct which gave priority to the communication. This ensures that whenever communication is needed it takes place directly and so the data can be sent immediately. If a PRI PAR construct is not used, the application time is increased by a factor of four, so in distributed systems it is essential that the correct prioritisation is used for the processes which communicate via links.

5.3. Link transfers

To minimise the time spent in transferring the initial data down a link, experiments were undertaken using various sized packets. A complete image, a row and a pixel were the sizes chosen. The figures show that it is slightly quicker to send the complete image down a link in one transfer than a row at a time.

5.4. Compiler options

Due to the limited amount of on-chip memory (in this case the transputer used was a T414 which has 2K bytes) the performance of the system is dramatically affected by which sections of data or code reside in this faster memory. In the version of the compiler used an option could be selected to use vector space. The compiler places what is to be on-chip in the order - code, scalars and vectors. So the programmer can use the option and then by trial and error put certain variables off-chip. This, however, is time consuming may not provide enough control to design a system which gives optimum performance. A better solution is initially to use only off-chip memory and then the difference in speed between two implementations is due to the algorithm and not the result of the code or data moving on- or off-chip as the program is altered. Then the programmer can selectively place data or code on-chip and see how performance is affected. By using this method the effects on performance of the inter-relationship between the algorithmic changes and code or data movement can be avoided.

6. CONCLUSION

A cost effective multi-transputer system has been implemented which converts three orthogonal images of an object into a rectangular parallelepiped volumetric representation of the object. The system has execution times which are very much shorter than those associated with the simplest manipulator movements used in typical assembly operations and so provides a practical testbed for the evaluation of identification techniques which would be suitable for industrial applications.

The application highlights a number of difficulties associated with parallel implementations which are required to process data in real time. A characteristic of the application is that each element of the data through the pipeline of the system requires different and unpredictable amounts of computational power and so make the exact balancing of the pipeline difficult. Since this was the most gross defect the use of buffers and the PRI PAR construct was the most effective means to reduce the delay associated with the synchronisation between processes running on different transputers.

Future work will initially involve the use of 256 by 256 pixel images, which are a more realistic size

for a practical application, and will then concentrate on connecting the frame grabbers directly to the transputer system. This will also allow the movement of the convolution processes, required in the preprocessing of the image, to special purpose hardware and so reduce the overall processing time further.

As the idle time monitor used does not show exactly where the time is spent in each process and cannot be used at all when there is a high priority process running, more work needs to be done to design a tool to assist the programmer to write efficient code. Similarly the experiments performed on buffer size were undertaken by trial and error and were time consuming. The facilities which would be usefully included in a future tool would allow the programmer:-

a) To identify well used pieces of code (ie how the time is actually spent on each processor) and how much time is spent waiting for data. The latter would help with the optimisation of buffer requirements.
b) To alter the monitored process at run time.
c) To identify the data and code which are on-chip.
d) To identify the number of times a particular variable is accessed. This would make it easier to decide whether it should be placed on- or off-chip.

REFERENCES

[1] CHIN, R T and DYER, C R 'Model-based Recognition in Robotic Vision' Computing Surveys Vol 18 No 1 (March 1986)

[2] SILLITOE, I 'A Gross Motion Planner for Robot Path Generation' VIth International Conference on Systems Engineering (1988) pp 499-504

[3] SILLITOE, I and EDWARDS, J 'A Multiview Robotic System for Efficient Object Recognition' International Conference on System Science, Wroclaw Poland (September 1989)

[4] KIM, Y C and AGGARWAL, J C 'Rectangular Parallelepiped Coding: A Volumetric Representation of Three-Dimensional Objects' IEEE Journal of Robotics and Automation Vol RA-2 No 3 (September 1986)

[5] KITTLER, J and ILLINGWORTH, J 'Threshold selection Based on a Simple Image Statistic' Computer Vision, Graphics and Image Processing Vol 30 (1985) pp 125-147

Appendix F

A GROSS MOTION PLANNER FOR ROBOT PATH GENERATION

Ian P. W. Sillitoe

Department of Electronic and Electrical Engineering,
Loughborough University of Technology

ABSTRACT

An important component of automatic planning for robot assembly operations is the gross motion planner. In this paper, an on-line gross motion planner is proposed for a restricted class of problems, which are applicable to assembly operations. The approach is implemented using a revolute manipulator, executing straight line paths. The relative speed of the algorithm makes it a desirable candidate for use in on-line error correction strategies.

INTRODUCTION

Motion with reorientation of the workpiece can be split into four phases, initial gross motion, reorientation of the workpiece and the final gross motion preceeding the fine motion control [1]. Given sufficient open volume for manouverability, reorientation may be accomplished with a special purpose planner [2]. However, due to the non-decomposability of the problem, algorithms that determine general collision-free paths for payload and manipulator in highly cluttered workspaces, are often impractical. This is owing to their implementation complexity and best known time bounds [3]. Previous work [1-2, 4-9] has considered off-line solutions to the problem, whilst at the same time it is noted that although these provide "good" paths, they have not been efficient enough for inclusion in on-line techniques, such as would be required for error correction strategies.

However, if we consider the assembly workcell sub-problem, practical assumptions can be made that allow efficient implementation of a gross motion path planner which produces suitable paths. These include that the workpiece and gripper be of comparable size and less so shape, that there be sufficient volume above the manipulator so as not to limit its manouverability and that manipulations do not take place at the extremes of the manipulators work volume.

FREE SPACE REPRESENTATION

The workspace and objects in a workcell can be modelled in terms of the space occupied by the objects or in terms of the free-space between them. A free-space representation is used here, as it allows direct reasoning about the configuration space of the robot. Further, knowledge of the free-space can be used by modules concerned with other aspects of on-line planning. i.e. determining where to place an object on the assembly table.

However, all free space schemes have the disadvantage, that after each movement the free space representation must be updated. So, ideally the free-space generation should be efficient and be readily updated from sensory information. A rectangular parallelpiped code (RP) fulfills both these criteria. RP representations can be determined from vision using multiple views [10,11], easily generated from geometric modellers and are efficiently manipulated. The RP code chosen is shown in figure 1.

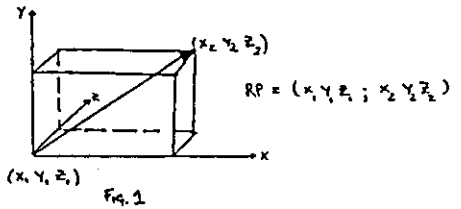


Fig. 1

An operator, ###, similar to the Sharp[12], operator used in the determination of prime implicant sets, is used to efficiently invert the object space to produce a free-space. The free space is represented as a set of maximal overlapping rectangular parallelepipeds. A graphical representation of the operator is shown in figure 2.

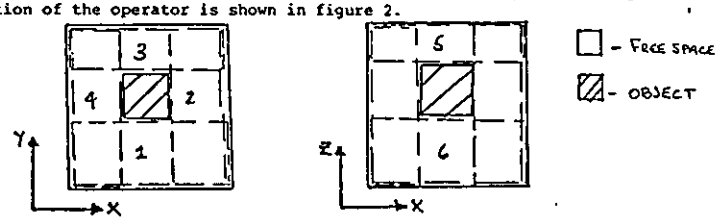


fig. 2

FS = Un ### s where FS is the set of overlapping RP's, representing free space.
Un is the RP representing the workspace
s is the RP representing the object

A ### B, performs the function $A \cap \overline{B}$ i.e. what belongs to A that does not exist in B. However, unlike Sharp, ### operates on continuous variables with a maximum of 3 degrees of freedom.

For a number of objects the FS is defined to be

FS = ((Un ### s1) ### s2) ....

Where S = (s1, s2.....sn) is the set of all objects within the workspace.

The path is determined in two stages, an initial path in two dimensions with nominal z coordinates is found, which is then followed by a process that modifies the nominal z coordinates of the via points, in such a way as to avoid limb collisions.

DETERMINATION OF THE INITIAL PATH

Once the free-space has been determined, the maximal RP covering the start position, ST, and goal position, G, are found. A search is then made through FS via overlapping RP's from ST to G. Only the RP's that would allow the passage of the payload are expanded. The payload is modelled as sphere of radius, Vr, this includes both the workpiece and gripper. This choice of representation makes the predicate for expansion efficient, but limits the forms of paths available to those without reori-

entation of the payload.

The search is based upon a heuristically guided best first search. At present two equally weighted cost functions are used, the distance from proposed RP to G, using the mid-points as point volumes and the distance from the present position to the point volume of the proposed RP. The search is similar to an A* search and the path is encoded within the parent links of the expanded RP's. However, the cost of testing for previously expanded nodes and modifying their cost fields, as required in the A* search, was found to outweight the cost of maintaining multiple paths to duplicate nodes and so no such tests are made in the search procedure. The initial via points for the manipulator, are generated using the point volume coordinates of the intersecting RP's (figure 3). These then have their z coordinates modified to be min (zmin+vr, zmin+zmax/2). This reduces the need for unnecessary variations in the z direction as a manipulator moves from RP to RP.

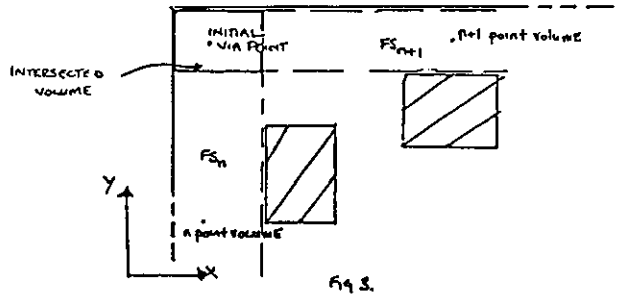


Fig 3.

Optimisation of the path length was considered and a simple technique implemented. The technique was a look-ahead strategy, that ensured the payload crossed into the next intersected volume at the optimal point to approach the next RP. However, when the cost of the extra manipulator movement was compared with the time required to perform the procedure, it was not considered necessary. Further, since the technique is to be supplied by data derived from sensory devices, with which there will be an associated uncertainty, the process of optimisation may not produce a better path in practice. It was also noted that a suitable choice of Un, when considering ST and G, produced a similar improvement in path length, to that of the optimisation technique.

MODIFICATION OF THE Z COORDINATES

The procedure stated so far, would provide a suitable path for a cartesian manipulator without further adaption. However, when applied to a revolute manipulator the form of manipulator introduces the possibility of collisions with forearm, as it stretches over an obstacle (figure 4). Hence, the inverse kinematics must be calculated in order to determine the manipulator limb positions and so modify the path.

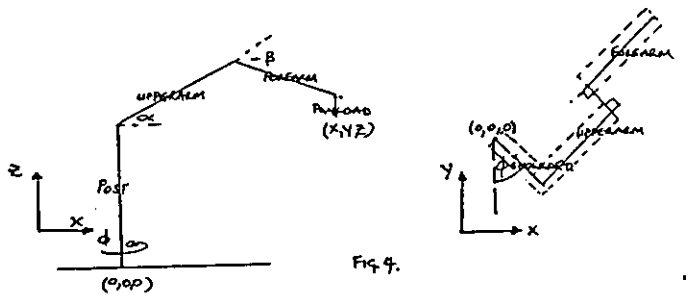


Fig 4.

The manipulator used for the experiements was a Puma 560, which has 6 degrees of freedom. In order to simplify the inverse kinematic calculations joint 4, the upper wrist of the manipulator, was fixed. This reduced the manipulator to 4 degrees of freedom, which allows for a less complex geometric form of solution to the inverse kinematics[1], figure 5. This is simpler to calculate, while the 4 degrees of freedom still allows enough flexibility to complete assembly operations[11].

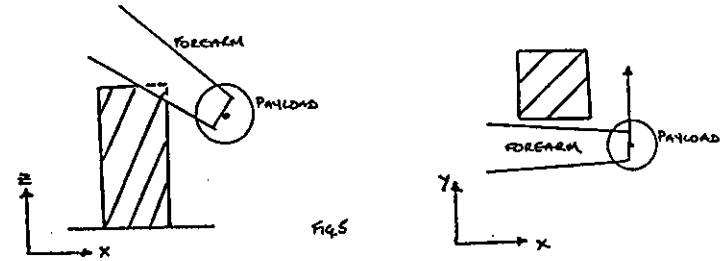


Fig 5

To avoid "side swiping" and "stretching" collisions, a simple heuristic was applied to modify the z-coordinates of the via-points. This involved ensuring that along any path segment, the via points were chosen so that lowest point on the forearm of the Puma was always above the footprint of the manipulator. The footprint was as those RP's, whose x and y dimensions were determined by the maximal X and Y coordinates which were swept out along the path segment, figure 6, and whose z coordinate was equal to the maximum height of the obstacle contained within this x-y rectangle.

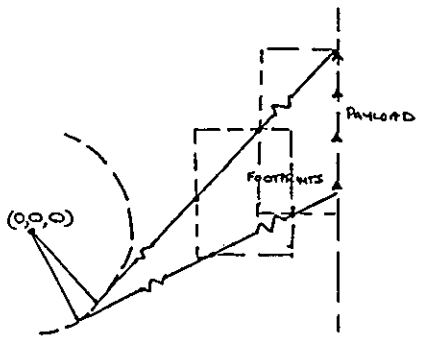


Fig. 6

The footprint is determined in two stages, firstly the extreme X,Y coordinates, using the inverse kinematics, were calculated from the geometry of the manipulator. The z coordinates are set to the maximum value of Z. Secondly, these initial forms of footprint's RP's are tested for intersection with the object RP's. The intersections are used to determine the maximum height of objects within the path segment and hence the height of the individual footprints.

EXPERIMENTAL RESULTS

A number of paths were planned, based upon those actions determined in [11]. It was found that the generation of free space took between .2 and 1.5 seconds and the determination of the via points took between 2 and 5 seconds, depending on the complexity of the path and number of via points. The planner was written in Forth running on a IBM AT clone. The code has not been optimised and it is expected a factor of improvement, of at least two could be made on the times, after optimisation. Hence, the planning times were comparable with the time taken for the manipulator to execute the path. This would make it possible to pipeline the execution of a movement and planning of the next movement and so subsume the cost of the planning.

CONCLUSION

A new technique for on-line collision free path generation for assembly operations has demonstrated. It is efficient and it's properties make it a candidate as a component part of an on-line adaptive path generator. Further work is required in order that such a technique can cope with paths requiring reorientation.

REFERENCES

[1] BROOKS, R. A.
"Planning collision-free motions for pick and place operations"
The International Journal of Robotics Research, vol.2, no.4, 83.

[2] LOZANO-PEREZ, T.
"Automatic planning of manipulator transfer movements"
IEEE Trans. on Sys., Man, Cyber., vol.SMC-11, no. 10, 81.

[3] SCHWARTZ, J. T., SHARIR, M.
"On the piano movers problem II : General properties for computing topological properties of real algebraic manifolds"
Rept. 41, New York University Dept. of Computer Science.

[4] DAVIS, R. H., CAMACHO, M.

"The application of logic programming to the generation of paths for robots" Robotica, vol. 2, 83.

[5] GRECHANOVSKY, E., PINSKU, I.
"An algorithm for moving a computer-controlled manipulator while avoiding objects", IJCAI 83, p806-813.

[6] KUNTZE, H.B., SCHILL, W.
"Methods for collision avoidance in computer-controlled industrial robots", ISIR 85, p519-529.

[7] LOZANO-PEREZ, T.
"Spatial-planning:A configuration space approach"
IEEE Trans. on Computing, C-32, p108-120, 83.

[8] UDUPA, S. M.
"Collision detection and avoidance in computer-controlled manipulators", Proc. 5th IJCAI, p737-748.

[9] WONG, E.K., FU, K. S.
"A hierachical orthognal space approach to three-dimensional path planning"
IEEE Journal of Robotics and Automation, vol. RA-2, no.1, 86.

[10] KIM, Y. C., ACCARWAL, J. K.
"Rectangular parallelepiped coding : A volumetric representation of three dimensional objects", IEEE Journal of Robotics and Automation, vol RA-2, no.3, Sept 86.

[11] HONG, T., SCHIENER, M. O.
"Describing a robot's workspace using a sequence of views from a moving camera", IEEE PAMI, Vol. PAMI-7, no.6, Nov 83.

[12] ROTH, J. P.
"Algebraic toplogical methods for the syntheseis of switching systems", INT. Trans. America Maths Soc., July p301-6, 58.

[13] NEVINS, J., WHITNEY, D.
"Computer-controlled assembly", Scientific American, p63-74, 238, 78.

Selected Glossary of Terms

Backtracking
This occurs in most classical search techniques. It requires the algorithm to restore the previous context of the search procedure, when encountering a failed or unwanted node in the search path.

Conflict Resolution
This is an important issue in those systems constructed from Condition-Action rules. During each cycle more that predicate may be true and so there is a need to select between the possible actions, this process is known as Conflict Resolution.

Extensions
Predicates are defined over a set of values known as the universe of discourse **W**. If we only concerned with unary predicates (i.e. predicates with a single argument), then we can represent each predicate by the set of values of its arguments for which it is true. This is called the extension of the predicate and the set will be a subset of **W**.

Meta-Planning
A technique for reasoning not just about the problem being solved but also about the planning process itself. A detailed description of the technique is found in "Planning and Meta-Planning (MOLOGEN part2) , Artificial Intelligence, vol.16, no.2, p141-169, 81.

Non-Linear Plan
This is the plan of a problem which cannot be decomposed into a linear sequence of independent sub-plans.

NP-Complete
Used to describe an algorithm which cannot be completed in polynomial time. In other words the expression describing its time complexity is exponential in form.

Pattern Directed Condition-Action Rules
These take the form of

IF <Circumstances> Then <do action or conclude something>

where the predicate is a pattern matching function. The exact form varies from system to system. A survey of the different forms and methods of operation is found in Pattern Directed Inference Systems, Academic Press, ISBN 0-12-7377550, 71.