THE UNIVERSITY OF HULL

Intelligent Assembly in Flexible Automation

being a Thesis submitted for the Degree of

Doctor of Philosophy

by

Klaus Kurt Willi Selke
BSc (Hull)

December 1988

To the best wife in the world

and the three loveliest boys

ABSTRACT

This work investigates the automation of assembly cells and the need to incorporate sensor-guided decision techniques. The experience of industry in this area is examined by observing a real cell on the shop floor. From the collected data, conclusions point to an alternative error interpretation which describes the successful completion rather than an enumeration of errors. A methodology for the description of the process in robotic assembly is developed. The constituent phases in handling components are identified as Feeding, Transport and Mating. Each phase has well defined characteristic properties which can be determined using appropriate sensing mechanisms. The Mating phase is given special attention by proposing the method of Information Spaces as a suitable frame work for sensor fusion and context directed interpretation. Thus the successful progress is described, regarding any deviations as errors. They, in turn, can be interpreted in the context in which they were encountered, and recovery is accomplished in the demonstration cell by operator taught routines. Where error repetition occurs, a simple look-up technique suffices to remove the need for another operator intervention. The required data structures and the implementation of the experimental cell are discussed. It is concluded from the results that the principle of knowledge-based assembly control exhibits an intelligent behaviour which contributes to an increase in the cell productivity. This method addresses only a part of the overall problem of assembly automation, but it has a central place in the system and could be extended to the complete system.

Preface

In part, this work was carried out in the context of a research grant which was supported by the Science and Engineering Research Council of Great Britain, under the ACME directorate, grant number GR/C/88772. (ACME $\hat{=}$ Application of Computers to Manufacturing and Engineering). As such, a portion of the work in implementing the demonstration cell is due to the research assistants Graham E. Deacon and S. Nicholas Davey, who carried out the work under the supervision and direction of the author.

The final shape of the implementation and detailed outcome has thus been developed within a research team. In addition, strong support was received from the remainder of the Robotics Research Unit within the Department of Electronic Engineering. At the time of this research, the group was very active in the general field of assembly, using not only engineering components but cloth and garments as well.

Consequently, it is difficult to attribute clear origins to some ideas which in the event turned out to be important to the development of the proposed method. For example, although a crude and immature form of the Information Space already existed, the first coherent and useable presentation of this description must be credited to G.E. Deacon. Similarly, much of the work on the slaves and their connection to the system has been carried out by S.N. Davey. However, the underlying principles of the work presented in this thesis are due to the author and have been conceived before the start of the project.

Acknowledgements

A thesis for the submission of obtaining a title of Doctor of Philosophy is a major piece of work for any individual and is usually not carried out in isolation. This is especially true for engineering subjects such as robotic assembly. Many people have (sometimes unwittingly) contributed to the basic process of forming ideas and putting them into practice. However, some can be singled out for their help and I would like to appologise in advance, if anybody has been inadvertantly forgotten.

Drs Paul and Gaynor Taylor must recieve special thanks for my initiation into the excitement which can be found in the subject of manufacturing automation, all too often regarded as dirty and menial. My supervisor, Dr. Ken Swift, has been a tower of strength when the task of bringing the work to a conclusion seemed to be too much at times. He encouraged the development of the ideas and in that context a special thank you must be given, too, to Dr. Helen Shen of the University in Waterloo, Canada. Her criticism and persistence did not let any untidy thoughts slip through the net and the constructive and lively discussions were at time more akin to a combat situation, but one which always remained friendly.

I would also like to acknowledge the support I was given by my Head of Department, Professor Alan Pugh, and all other colleagues. They have given me enough room and opportunities to carry out the research so that this thesis could be written. Jim Timulty and Kevin Stobbs were my contacts at the industrial partner, the FLYMO division of Electrolux in Darlington. I am grateful that they allowed me to use in my thesis the industrial knowledge and insights obtained from the factory floor. I was able to gain much practical experience otherwise not available to me. As far as support for practical implementation is concerned, hardly anybody has contributed more than John Hodgson. He never lost his temper with my sometimes impossible demands and (not always - but nearly always) managed *to mill everything out of a solid block to a 'mil'* precision. The work would not have progressed as it did without his understanding and patience.

The latter two qualitites have been outstanding in all the members of my family. For a very long time, they put up with me and my short temper and let me retreat to concentrate on working. I admire my wife even more for being so supportive and I am sure that my three boys do not yet know what *PhD* actually means, although they were regularly imitating me in play. I promise I will make up for the lost playtime. Even my parents-in-law, Mr. and Mrs. Lister, did not escape, thank you for all your interest and encouragement. Yes, Judith, I will now also have the time to fix all the things in our neglected house. And, thank you, Masha, for being such a good friend.

Finally, I would like to thank Dr. Susan Pulko for taking some time off her busy schedule to proofread the thesis and make suggestions to improve my imperfekt English.

Contents

List of Figures

Page

Commonly used Abbreviations
(in alphabetical order)

AI	Artificial Intelligence
ASCII	American Standard Code for Information Interchange
BASIC	Beginner's All purpose Symbolic Instruction Code
CAD	Computer Aided Design
CAM	Computer Aided Manufacture
CCD	Charge Coupled Device
CIM	Computer Integrated Manufacture
cm	centimetre (10^{-2} m)
DC	Direct Current
DEC	Digital Equipment Corporation
FTM	Feed - Transport - Mate
g	gram
Hz	Hertz
kg	kilogram (10^{3} g)
kHz	kilo Hertz (10^{3} Hz)
LED	Light Emitting Diode
LSB	Least Significant Bit
m	metre
mm	millimetre (10^{-3} m)
MHz	Mega Hertz (10^{6} Hz)
MSB	Most Significant Bit
N	Newton
Nm	Newton-metre
PCB	Printed Circuit Board
PID	Proportional/Integral/Differential (control)
PLC	Programmed Logic Control
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
SCARA	Selectively Compliant Arm for Robotic Assembly
VAL	Variable Assembly Language
VMS	Virtual Memory System (DEC Operating System)

Chapter 1

Introduction

The work presented in this thesis concerns itself with the topic of assembly in the context of the application of automation in industry. As this is a very wide ranging field, only one specific aspect can be considered here. It is attempted to provide a methodology which brings the benefits of automation into those areas of assembly in which conventional and dedicated techniques have so far proven to be inadequate. An increase of productivity in manufacturing as a whole is seen as one of the keys to commercial competitiveness and economic wealth. It is well known that automation is one of the factors contributing significantly to the desired increase in productivity. Present levels of success have come from machine utilisation over a very large number of products and have thus been associated with very large companies such as the car industry. These high levels of productivity are sought in all areas, but the inflexibility of dedicated machines has prevented comparable improvements in productivity for medium and small batches.

1.1 Assembly as a potential growth area

One area which still has considerable potential for growth in automation is that of assembly. It has been argued[40,41] that there are only a few types of products which can justify the large volume automation whereas the majority of products is made in relatively small batches. Less volume implies a more frequent setting up of production lines for different products and requires a comparatively large effort for a production run. This makes the application of the same automation techniques for small and large batches[42,43] alike too costly, despite the wealth of advanced and well accepted hard-automation methods.

The major difficulty with *large volume* techniques is the control of conventional machinery by mechanical means. Either end-stops or cam-following mechanisms or a combination of both are commonly used. Any changes in functions which are expected to be carried out by the machinery involve the intervention of operators. They have to halt the production line for as long as it takes to change the mechanism, install new equipment, and adjust the sequence and timing so that the assembly line actually performs in the way it was intended. Unless there is very little hardware to change or the functions themselves are simple, these times can be long and, naturally, the greater the differences in the products, the longer it takes for the changes to be completed. Equally important to the overall level of productivity is the fact that change-over times become less predictable, making it harder to provide precise facts for economic justification associated with batch production.

This aspect of inflexibility of a production assembly system is addressed in this thesis. The overall aim is the provision of a framework in which limitations are imposed only by unavoidable physical constraints. Examples are the weight handling capability of the machinery and/or its working volume, which restrict the actual number of individual parts by limiting the number of feeders.

Otherwise, the assembly process is defined in general terms, removing *guess work* (that is the lack of a high level of experience) and reducing the dependence on *'good ideas'* to a large degree.

It has to be stressed that the subject of assembly is very large and complex, and thus not every aspect can be addressed in this thesis. Hence it is not the intention here to condemn good ideas, rather it is felt that some of the blame for the failure of more widely spread applications of robotics can be found in the lack of a systematic approach to the subject. Of course, in the area of robotics, assembly and the integration of sensors, a significant wealth of publications exists where systems yielding high productivity are claimed. However, many of the examples presented in the literature do depend on good ideas which have worked well in the circumstances described[74,75,76] but are difficult to transfer to a different shop-floor with different products.

Alternative proposals for 'complete' systems exist as well. They are potentially able to manufacture a product automatically without human intervention. Futuristic factories are often portrayed as containing complete automation, from the design stages at sophisticated CAD terminals right through to the point at which products leave the factory ready for distribution. All too often, these propositions leave out the implementation stages, and although these are generally regarded as mere mechanistic exercises - particularly in academic circles - they are not trivial and need a great deal of attention because of the complexities involved in the production and assembly phases.

Effectively, the lack of understanding of the stages in a general assembly process has led to failures in the application of more flexible machinery. Unforeseen circumstances are the major contributors to unacceptable levels of errors; the elimination of which by conventional means would be too expensive, especially for small batch production. Thus new approaches are needed and the 'revolutionary' drop in cost for microelectronics and microcomputers has given rise to the hope that mechanical changes can be replaced 'one day' by software changes. The rationale is, of course, that changes in software are easier to perform and take a lot less time to implement on existing electronic hardware. In principle, this assumption is justified.

1.2 Robotics Within Automation

One very exciting development in this area was the advent of the computer controlled robot. Such technology promised to remove the long change-over times, firstly by removing the product dependence of dedicated mechanical manipulation and secondly by allowing the task changes to take place in software, with only a minimum of hardware changes. In fact, in the early stages of robot deployment in industry, the abilities of the manipulators themselves were surrounded as much by mystery as by enthusiasm for their potential, so that it was very difficult to separate fact from fiction. The universal robot appeared to be very much on the horizon and such hopes were exploited - either consciously or sub-consciously - by robot manufacturers.

The first real success of robots was in the car industry[38], where the mechanical dexterity of a six-degree-of-freedom robot gave significant advantages in production techniques. It provided considerably easier access from varying angles even within areas with restricted access. With little room to manoeuver, the weld gun or paint sprayer[39] could follow a complex path with high repeatability. Successes in these applications provided another impetus to the community involved in robotics.

It was quickly realised, however, that robots, whether controlled by computers or not, were not much different from other pieces of machinery, in that they followed a particular pattern which they had been taught, and making changes in the sequence of events also turned out to be very time consuming. Despite their mechanical flexibility, robots still retained one fundamental problem in common with conventional machines. They are not able to adjust themselves even to minor changes in their working environment and have to rely heavily on products and tools being presented within very small tolerances.

Such maintenance of the environment assumes a higher degree of importance in less dedicated systems. Flexibility also brings with it an increased amount of disorder[28] and hence a potentially larger source of errors. The immaturity of robotic techniques does not provide enough experience to make the robot operation more flexible, and thus its advantages are not fully exploited by matching the mechanical dexterity with appropriate skills. Solutions are taken from known production techniques in conventional automation. Consequently, much of the peripheral equipment and programming systems are still specifically designed for the goods currently in production. This also means that the potential advantages of robots are largely masked by the significant amount of time spent changing the robot's environment and re-programming to suit the new product. Many of the robot applications were again seen to be successful only in the restricted area of comparatively large volume productions, such as the FLYMO assembly.

The previous arguments are especially true for those operations where the product has not been designed for assembly, but where the design relies on traditional, manual skills of human operators. The gap in the ability of machines trying to imitate human skill is most apparent, where it is attempted to replace the human directly by a robot system. It is thus not surprising that the community involved in robotics eventually directed its efforts towards the solution of these shortcomings and identified the inability of the robot to react to changes in its environment as the main cause for the large failure rate. Despite the continuing anthropomorphic appearance of manipulators, the reality demanded to view human operators and robots not as equal but to identify their respective advantages and make use of their best individual abilities.

1.3 Necessity of Sensors

While the robot's performance is all too easily compared with that of a human operator, the machine inevitably looses out because of the human's superior and very versatile sensing mechanisms. In addition, he has an almost infinite capacity for interpretation of the signals and learning of new situations. For robots, there are mainly two different approaches to the improvement of productivity, but the one taken from conventional automation is very inflexible and costly, as discussed before, and is thus better suited to large volume production.

In the alternative method, the robot - or more correctly the assembly cell - can be equipped with some means of *environment identification*, which is commonly referred to by the human equivalent of sensing. By its very nature, this offers a significantly increased flexibility because the process of adjustment can be made to be essentially product independent. Precision requirements can be relaxed[1], making auxiliary equipment less costly to produce and reducing set-up times by an equivalent relaxation in positioning and teaching requirements. That is the theory, anyway.

Although the continuing comparison with the capabilities of human beings is really fundamentally wrong, solutions appear to be particularly elegant. At first glance, it seems to be quite simple to *look* at a scene, make a decision upon the scene state and take actions accordingly. In that context, vision has become the most important sensing mechanism, because it has fired the imagination of academic researchers and production engineers alike.

This enthusiasm for sensing has been vindicated only in some areas, though these are involved in key areas of production techniques like welding of thin sheet metal[77]. The degree of success is linked to the degree of understanding of the process under consideration and to the degree in which manipulators and

---

[1] Correction: People also work more efficiently in a pre-ordered environment

sensors could then be matched to those process parameters. Subsequently, individual solutions have emerged for particular problems, but many use the same primary sensing mechanism (like vision).

Nevertheless, the software remains tailored to the process for which it is written, and solutions lack general applicability - often even within the same problem area. Despite the apparent drawback of showing high levels of achievements only in comparatively narrow areas, the usefulness and great potential of sensing itself has been vindicated by such successes. This is partially due to the large range of applications which have used the same primary sensing mechanism. Especially in assembly, such implementations have depended largely on the skill and intuition[2] of individuals, rather than an application of systematic rules. On the other hand, general purpose solutions, emerging usually from academic institutions, have retained academic properties from which many potential industrial users still shy away.

It is this general background against which the work presented in this thesis has to be seen. The assembly of products has remained one of the areas in which manual techniques have dominated and in which there is a very large potential for improvement of productivity through the introduction of automation. The computer controlled robot together with some form of sensing offers a potential base from which the assembly process itself can be automated. It is argued in this work that the concept of the universal machine (or assembly cell for that matter) is not a viable proposition but that the machines and sensors have to be matched to the manufacturing processes under consideration.

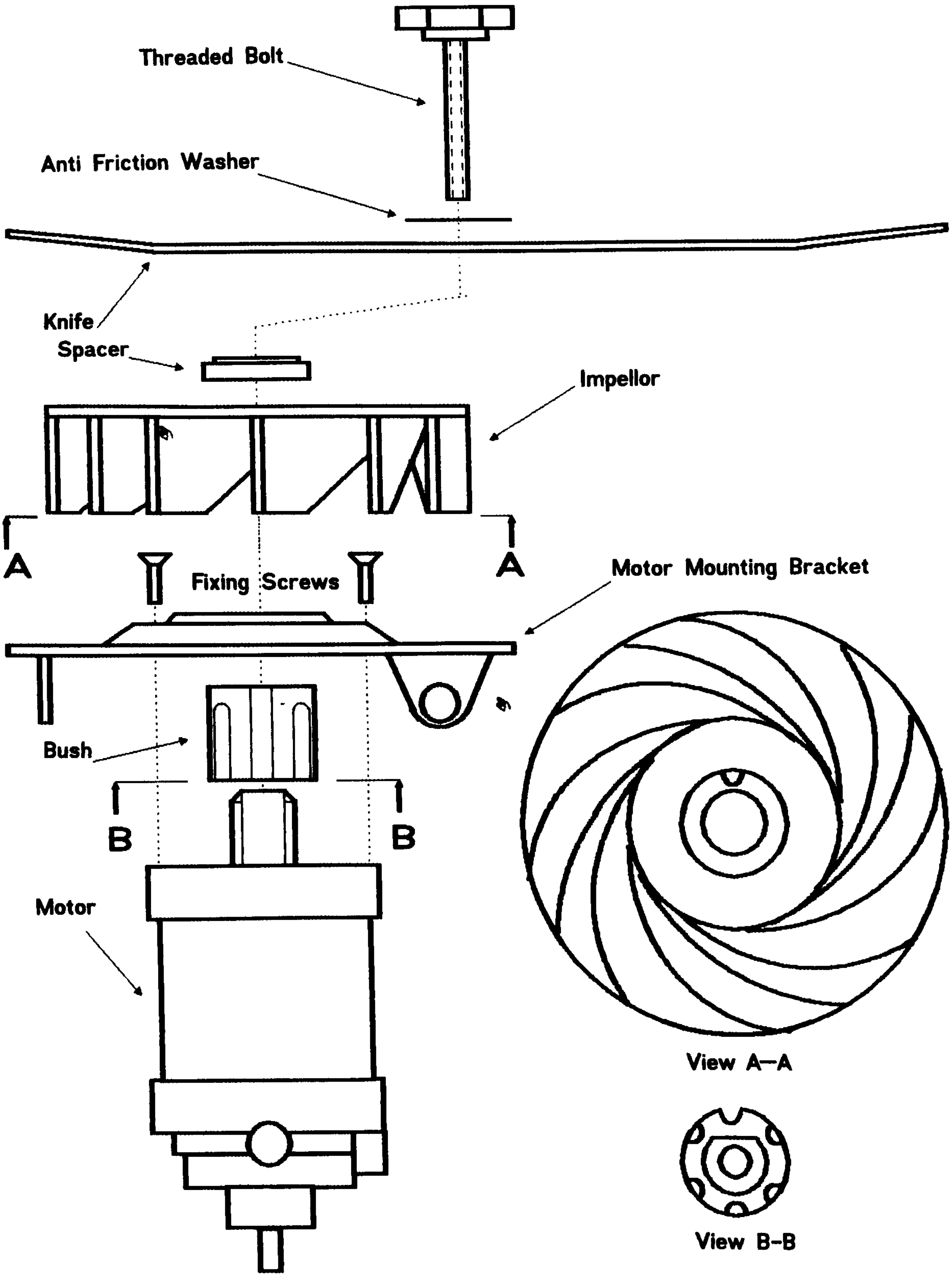


Figure 1.1: FLYMO Lawn Mower: Power Unit Assembly
(Courtesy of Electrolux, Flymo, Darlington)

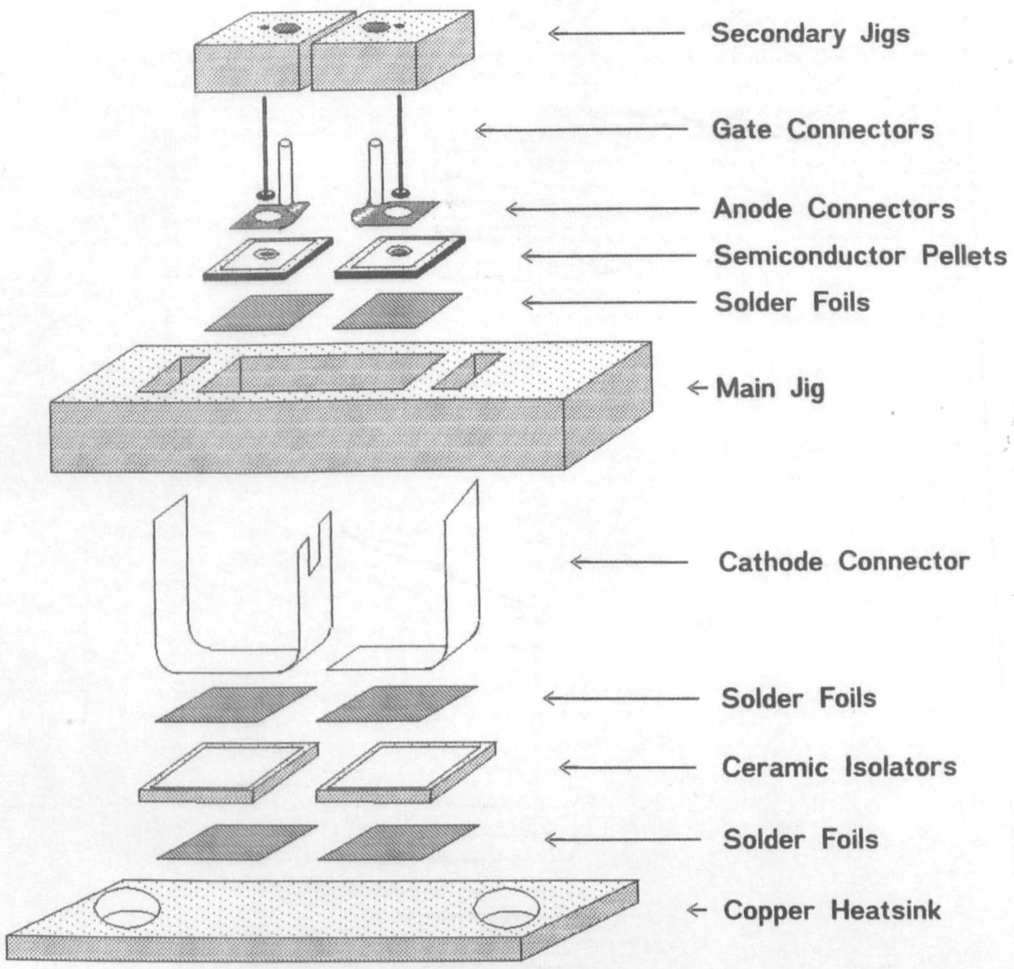


Figure 1.2: Power Thyristor Sub-Assembly (Courtesy of GEC, MEDL, Lincoln)

1.4 The Assembly Process

Assembly is one such process. The many different ways in which products are being put together appear to have strong and obvious similarities[3], although there is no relation in the final function of the product (compare figure 1.1 to figure 1.3). This thesis suggests a structure for a cell which is based on generic assembly processes which formalise the similarities. The range of applications can now be defined at the cell's design stage and is not dictated by a unique combination of individual pieces of auxiliary equipment which may have been necessary for the assembly of one batch.

Through an extension of the knowledge about generic processes, sensors are incorporated, chosen for their applicability in a range of generic assembly processes, rather than because of subjective preferences or even misjudgements.

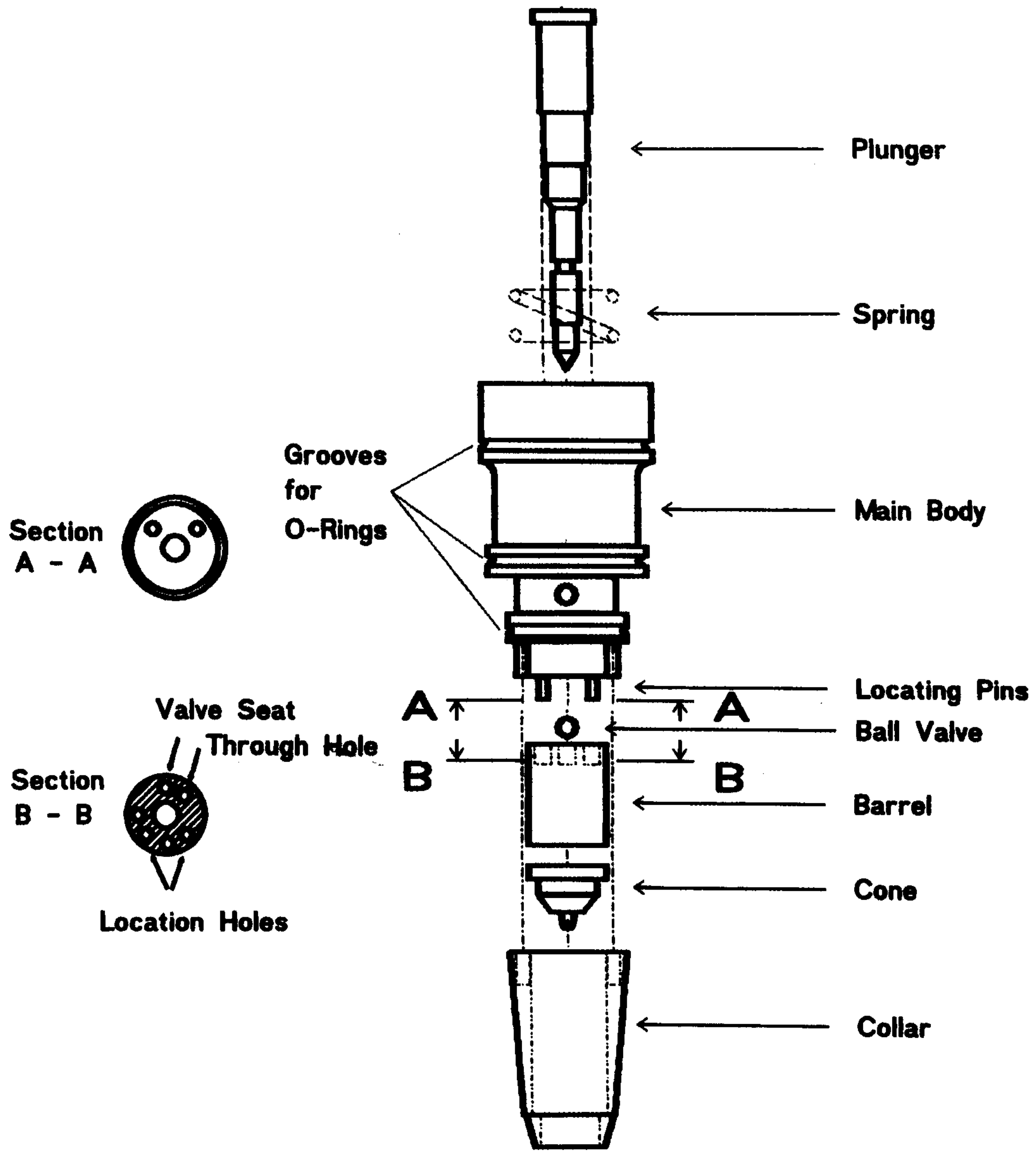


Figure 1.3: Diesel Fuel Injector Valve
(Courtesy of Cummings Diesel Engines, Darlington)

This thesis argues that the characteristic parameters for each stage in the assembly process have to be identified allowing conclusions to be drawn as to its progress (that is, whether it proceeds as planned or not). Such an approach has to consider the parameters which identify the success. Errors can then be regarded as a Boolean negation of that success. This is in contrast to the conventional way of contemplating and programming error routines in order of highest probability. A relatively limited set of conditions emerges for successful assembly. Not all combinations of error states - which could be identified by pre-determined sensors - need to be interpreted before run time.

To the author's knowledge such a structure is new and has not been proposed elsewhere. However, the many *good ideas* referred to earlier and available at any one time, are very often *good*. They must be able to be integrated and, where appropriate, used as easily as possible in the cell. The use of existing *good ideas* is certainly true for the demonstration cell described in Chapter 6, and they have contributed to its successful operation. Thus it is not proposed to dispose of existing techniques, but ideas which help increase the flexibility of the application are welcome. Flexibility to incorporate new solutions is essential for a basic structure. It must not be dependent on technologies or fashions at the time of its introduction. It is, incidentally, much more likely that partial solutions emerge for some problems, rather than frequently proposing radically new systems in a developing technology.

1.5 This Thesis and Its Subjects

In this thesis, it is attempted to present the development of the assembly structure in such a way that the reader is able to follow the thought processes leading to the proposed system. This has influenced the order of subject choice and the following sections explain the contents of the individual chapters.

In Chapter 2, the observations of the assembly cell at Darlington[2] are described and the most important conclusions with respect to the requirements for a general assembly cell are given. The assembly of a lawn mower's power unit is described because of its importance to the demonstration cell in which the presented ideas are implemented. Although the sample rate of assembly cells is not sufficient in a true statistical sense, the value of this data is considerable, partially due to its scarcity and partially due to the fact that it gives real insight into the nature of errors on the shop floor. Essentially, the data suggests that any successful and autonomous assembly has to be able to incorporate unforeseen circumstances in order to ensure that disruptive influences on the production flow are kept to a minimum.

The subject of Chapter 3 is the assembly process itself and a framework for its decomposition is developed. The objective is the identification of those parameters in each stage which can be measured in order to monitor the successful completion of each stage. It is suggested that the divisions into Feeding, Transport and Mating are highly suitable. The generation of appropriate rules can then be seen as the outcome of this analysis. The use of this methodology by less experienced personnel will provide advantages in a systematic and hence more repeatable execution of the production.

Decomposing tasks in an assembly cell into its principal parts highlights separate areas of control. Chapter 4 is assigned to describe the knowledge-based approach to the sequence control of the Mating stage and the concept of Information Spaces is developed. A multi-dimensional, orthogonal space is created which incorporates all information relevant to the mating process. It is explained how each of the axes represents *interpreted* sensor data, and special sub-spaces are assigned to the successful progress. All complementary sub-spaces thus represent unforeseen (or error) situations. The advantage is highlighted as a) the ability to accommodate any amount of sensory information in the structure and b) the provision of the correct context for its interpretation.

Chapter 5 develops the data structures which are needed to represent the information contained in FTM. By differentiating between generic and product specific data, the structures become implementation independent and the automatic execution can be supervised by operators at a conceptual level. Operators do not need to concern themselves with low level details. In addition, the concept of *Local Accuracy* plays a particularly important role for self-calibration and reduced operator supervision.

The basic elements for applying intelligent monitoring and control to an assembly situation are established in the chapters preceding Chapter 6. It describes the practical implementation of a demonstration cell. The description is kept relatively brief, because many details are not necessary to implement the concepts. Where the information is thought to be of interest, explanations have been given, for example, the role of the master-slave configuration. Many details are contained in the Appendices, however, and the interested reader is referred to them at appropriate places. Individual components in the cell are described as well as the realisation of the structure chosen for the cell. Finally, results are presented in the form of accumulated error situations and the behaviour of the cell in general.

Chapter 7 contains a critique of the proposed system and its implementation in the demonstration cell. A newly emerged view of errors is explained. The necessity of retaining sensing is discussed, even though they cause an increase in system complexity. The aspect of *intelligence* as contained in this cell is also discussed.

The final chapter, Chapter 8, highlights implications for further work to be carried out so that the context of this structure within the complete assembly process can be established. The wealth of run time data is seen as a key to the assessment of the cell performance, whether this is through sophisticated Man-Machine-Interfaces, or by incorporating elaborate algorithms for choosing sensors or planning run time actions. The automatic generation of the data which is required for this method would complete the chain for integrated assembly automation.

Chapter 2

Investigation Into The FLYMO Assembly Cell

This chapter presents the observations of a shop-floor assembly cell which was installed at Electrolux in Darlington to produce the FLYMO lawn mower partially by robotic techniques. A description of the parts involved in the assembly of the power-unit is given in detail, since the practical implementation of the system suggested in this thesis uses the same components in the demonstration cell described in Chapter 6. Observations are given as a summary of the most significant results for the assembly process. For the purpose of this work, any associated administrative aspects, such as managerial and financial problems, are not considered. The major conclusions drawn from observations made during the course of this project resulted in viewing error recovery as the ability of the system to learn new routines at run time. Many of these circumstances cannot be foreseen at the design stage and equipment installation. Nevertheless, such unpredictable errors have a significantly disruptive influence upon the flow of production.

2.1 The FLYMO Product

The product under observation was a domestic lawn mower which is sold under the brand name of FLYMO. At the time of observation - during the periods October 1985 to March 1987, robots were employed in the assembly of the smallest model, E25. Only the power module was assembled automatically; it was then placed into the orange hood manually. The latter has become familiar through the success of this product and frequent advertising in the media. The connection of power cables and packaging were also carried out manually because commercially available robotic (and other) systems did not offer viable solutions for these processes. It is mainly due to the fact that the E25 model was designed for manual assembly that those stages were not considered to be suitable for automation and models later than 1987 have been adjusted for easier assembly by machines.

Figure 1.1 shows the twelve parts needed for the power unit, with nine different components. It is already apparent at this stage that the cell has been designed for large volume production and not for flexible changes of products. This does not distract from the value of the information obtained from the factory floor, since the observations were collected from a *real* manufacturing environment. The numbers attained are far higher than those which could ever be achieved by a demonstration cell in a research laboratory.

The complete manufacture involves a variety of additional processes apart from assembly, such as injection moulding of hoods and bending plus coating of handles. Within the context of this work, these can be regarded as peripheral. The assembly process is of interest here and only the sequence of events in the robot cell itself is described as it was operated at the time of observation:

1. *Motor*

The electric motor is buffered ready for further processing. Since it is too heavy to be handled by the main assembly robot (Unimation PUMA 560), a second manipulator (Cincinati Milacron T3) is used. This lacks precision but is capable of handling up to 50kg payloads. It picks up the motor and places it into a rotary jig, which subsequently turns by 180° in order to present the new motor to the PUMA. While the new power unit is being assembled, the T3 is free to carry out other tasks. For example, it removes the finished sub-assembly and palletises the packaged lawn mowers.

2. *Motor Mount*

The PUMA collects the motor mount from its feeder and places it onto the motor. The mount provides the main mechanical link to the hood.

3. *Screws (four)*

Four screws are applied to fasten the motor mount to the motor. The weakness of the PUMA 560 makes necessary the use of an auxiliary tool which is able to withstand the reaction forces exerted during the operation of the pneumatic screwdriver. In these circumstances, the robot is used merely to guide the mounting mechanism of the auxiliary tool in the x-y plane, while the vertical motion and force is controlled by an additional pneumatic actuator. The screws themselves are supplied from a centre hopper feeder and forced along a plastic tube by compressed air to the active point of the automatic screwdriver.

4. *Bush*

The bush serves as a mechanical interface between the motor shaft and the impellor which generates the airflow necessary for the lift of the mower from the ground. It has a key (see figure 1.1, view B-B) on the inside in the form of a flat and a key for the impellor on the outside in the form of a semi-circular groove.

It is interesting to note in this context, that initially, the assembly of this component was also attempted with the robot. The shaft of the motor was rotated with reduced power while the robot applied a downward pressure. When the keys on the motor shaft and the bush were aligned, the bush dropped into place. This action was detected by a simple compression switch. The whole procedure worked well for an initial period of just over a year, but then the success rate dropped so drastically that the resulting failures could no longer be tolerated. After several unsuccessful attempts to rectify the error, it was decided by the FLYMO production management to employ an easier solution using a manual operation. The operator placed the bush onto the shaft before it was placed into the buffer. This did not increase the load of the operator noticeably and has restored the desired levels of productivity.

5. *Impellor*

The keys of the bush and the impellor also had to be aligned, again without prior knowledge of their relative orientation. The same technique as for the initial automation of the bush has been used and this has had sustained and acceptable success rates throughout the period of operation.

6. *Blade, Anti-Friction Washer and Bolt*

These are assembled as a separate unit at the feeding point of the blade. The robot collects a suction tool, then places the anti-friction washer onto the blade such that the holes are sufficiently aligned for the bolt to be inserted. An auxiliary tool at the blade feeding point makes sure that the alignment is within the required tolerances. The bolt is then picked up from a bowlfeeder and guided through the concentric holes of the blade and washer, securing the latter between bolt and blade. At the correct depth, permanent magnets attach themselves automatically to the blade so that the completed sub-assembly can be carried over to the motorshaft.

The thread of the bolt is then screwed to the internal thread of the shaft, again by turning the motor with reduced power until the required depth is reached. At this point the assembly is finished and the rotary jig can again swing back, simultaneously presenting a new motor to the PUMA and offering the completed sub-assembly to the hydraulic robot. The latter can remove the power unit and place it onto a shoot which carries it outside the cell. Further operations are now continued by manual assembly processes outside the cell.

This particular sequence of the production was monitored with respect to the failures that manifested themselves during the production run.

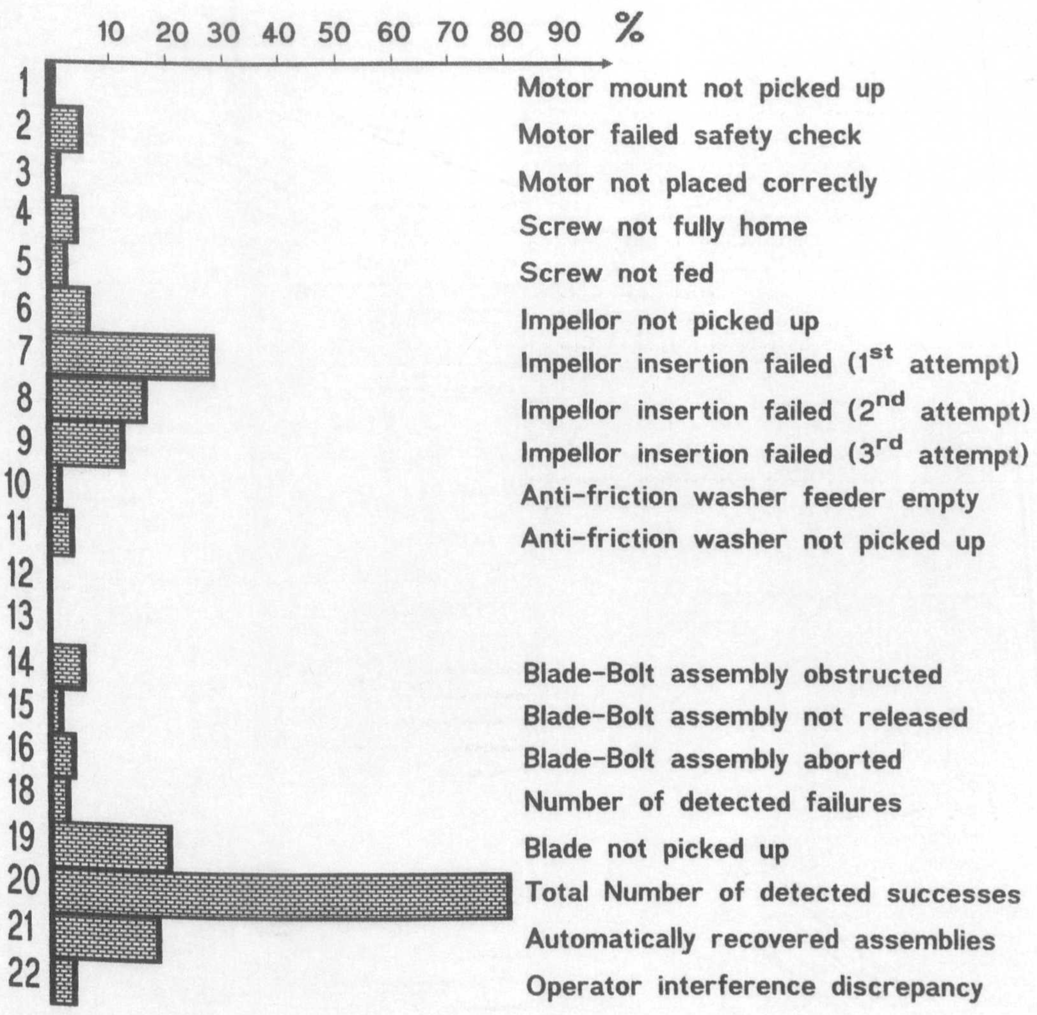
2.2 Power Unit Assembly Failures

The observations were taken in two ways:

1. Modifications to the production line programs running in the supervisor computer logged the entries into those 'error recovery' routines which could be invoked through automated detection. This relied on a number of simple sensors, such as proximity for the presence of the blade after it had been picked up by the magnetic grippers. Another simple switch detected the compression of the wrist, indicating a fixed force and displacement, such as might be encountered when the shaft and bush are not aligned. The change of state of the compression switch would then indicate the engaging of the key features after alignment, provided it was known that the robot itself did not move down. Thus it was possible to keep a constant observation of the cell over such a long period without the sustained attention of an operator .

2. Separate periods of one week each were spent at the factory itself to verify and interpret the situations arising from the automatically logged errors. It was important to be able to observe the situation, since the simple sensing arrangements only provided an indication of anticipated circumstances. The sensors and their associated software are not capable, however, of generating or processing enough information from which it would then be possible to analyse the situation correctly.

The number of products leaving the cell during the period of observation are company confidential and hence it is not possible to reproduce them. However, to support the validity of the investigations, it is permissible to say that the assembly processes logged were in excess of 50,000 over a period of several months and hence the observations are 'statistically significant'.



Note: Number 17 is the total number of units produced (100%)
Number 12 and number 13 are expected errors which did not happen

Figure 2.1: Summary of the Error Distribution in the FLYMO Cell

The summary of the error distribution is shown in figure 2.1. Effectively, this distribution represents the actual occurrence of anticipated errors. Consequently, some effort was invested in the pre-programming of certain routines, such as a limited number of re-tries for the impellor placement. These data and their distribution have thus validated most of the production engineer's efforts, particularly with respect to the number of repeated trials for the impellor placement. About a fifth of the products would have failed on account of that error condition alone, if it had not been for the re-try.

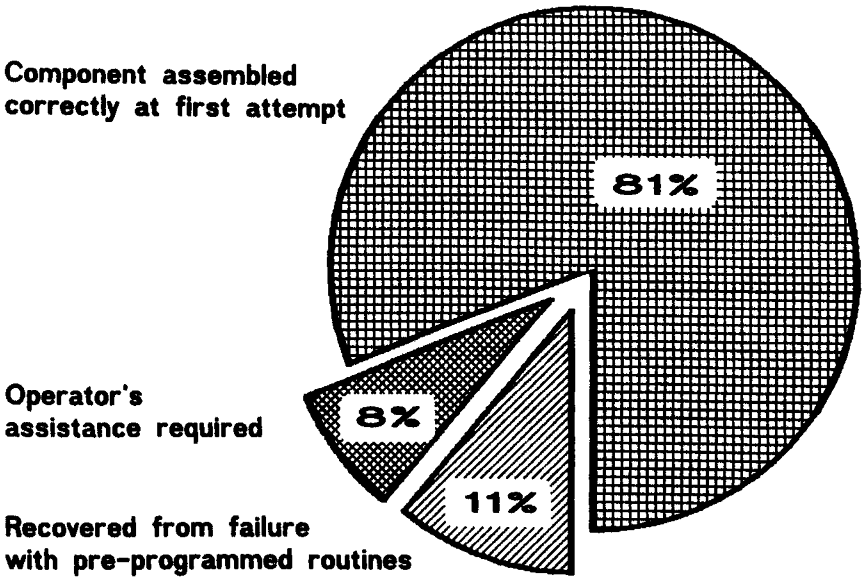


Figure 2.2: Summary of Error Recovery

2.3 Critique of Observations

Before the collected data can be interpreted, the context of that interpretation has to be set.

Firstly, although the actual number of products is sufficient, the product sample rate itself is definitely too low, just one. However, such data is extremely difficult to obtain in general, mainly because of the aspects of 'company confidentiality'. In that respect alone, acknowledgements are opportune here. Jim Timulty (Production Manager) and Kevin Stobbs (Senior Production Engineer) have enabled virtually free access to the production cell and supplied the parts necessary for dimensional investigations. The additions to the software were of special significance since it would not have been possible to collect such detailed information over this prolonged period.

Secondly, the cell itself has obviously been set up for a single product, rather than for batch production or even variations within the one product. Since the aim of this project is to provide exactly such flexibility in a systematic way, it is important to appreciate this fact.

Nevertheless, the data is valid as a look at an example for typical production performances. The exact proportions of the three main situations may vary (see figure 2.2), but most visitors from industry have confirmed that these figures correspond roughly to their own in similar circumstances.

2.4 Interpretations From The Observations

The most striking part of the pie chart in figure 2.2 is the largest portion of about 80% during which there are no errors at all. This reflects the careful design of the cell and good choice of fixturing for sub-assemblies under these circumstances.

Undoubtedly, the success of the cell is also due to the skill of the designers[2] who were able to foresee the most likely errors. They chose the most appropriate sensors, put them at the most suitable places and then were able to interpret the information in such a way that they could identify certain procedures for the recovery or at least for an attempt at recovery. Thus a further 12% to 15% (variations did arise throughout the total period) of all the products were able to be assembled without interruption of the manufacturing process or any interference by a human operator.

2.4.1 Manually Recovered Errors

In the last portion of assemblies, 5% to 7% of all attempts, it was necessary to ask for human assistance. Either the number of re-tries was exceeded or the condition detected was so complex that the unaided robot would not have been able to recover from that situation. In any case, recovery usually consisted of a 'clearing up' operation in which all unwanted parts were removed. This included - most times - the partially completed sub-assembly. The operation itself was then restarted from a known point in the sequence, usually from the beginning.

Economic considerations made it necessary to employ the operators of the cell elsewhere while the cell was running and consequently the conditions which lead up to a specific failure situation were not always noticed by them. Due to that fact, individual error causes were not of great importance to the operators, but the symptoms were sufficient to provide them with enough information as to what course of actions was needed in order to restore production.

For those reasons, it was not very easy to extract numerically precise and detailed information from the operators. Initially, it was hoped that the exact conditions leading up to a specific failure could be monitored. In fact, the concern of the operator was very rarely with the causes of the errors but almost exclusively with the uninterrupted output flow of lawn mower power units. This attitude existed to such an extent that often the appearance of good flow was sufficient. That lead sometimes - during error conditions - to the operator stopping the cell before the condition could actually manifest itself as an error in the monitoring program (entry 21 in the histogram in figure 2.1: Operator interference discrepancy).

2.4.2 Role and Quality of Human Supervision

Such activities do not contradict the previous statements about the inability of operators to observe the cell constantly, but reflect the learning capability of humans to react to certain changes in the environment. For example, the attention of the operator is not required constantly for the action he is currently trying to perform. He will have repeated them so often that some of the handling and motions become 'second nature' to him. So, as long as he is within the physical proximity, he is able to divert his attentions momentarily and will then notice any deviations in the state of the cell.

As well as expecting to see the right conditions during certain parts of the assembly sequence, the operator will also be getting used to a certain pattern of noise which is generated by the robots and the actuation of pneumatics and other moving equipment. Consequently, some conditions can be heard; and as this listening process is executed independently of current activities, some conditions are often noticed sub-consciously while the operator is engaged in other tasks.

During the personal observation of the cell, informal interviews were conducted with different operators. Unfortunately, the emphasis had to be placed onto 'informal', since there was some danger in giving the impression to the operators that they themselves and not the cell were under observation. Nevertheless, one particular piece of information concerning types of errors did emerge from these interviews and deserves further attention.

2.4.3 The Nature of Unexpected Errors

Most of the actions required from operators to restore the production flow resulted from errors which could not usually be predicted by the operator or designer. However, despite such unpredictability, different sets of error conditions repeated themselves regularly throughout a limited period; sometimes this period was a day or even less, at other times it was a fortnight.

The name to these conditions given by the operators was the 'Monday Morning Blues'. This concept encapsulates the fact that the errors appeared from somewhere - usually causes were not known - and they also disappeared again after some time - again, most of the time the reasons for the disappearance were not known either.

2.5 Implications from Error Observations

It has become apparent from the behaviour patterns discovered during the cell's operation that, in practice, it will not be possible to produce a flexible cell which can be programmed for most error conditions. There will always be some circumstances which the designing and maintaining engineers will not be able to foresee.

On the other hand, by far the largest part of assembly operations is already executed without any interruption and there is every reason to assume the continued high quality of future cell designs. Hence any additional procedures to check the fact that there is no error will result in an inevitable increase in production time. This is particularly true for the more sophisticated sensing mechanisms such as vision, and effects can only be kept negligible for simple sensors, for example binary sensors like cross fire or proximity. It is therefore highly desirable for a practical system to reduce the sensing to very essential points in the process, if it is not possible to execute them in parallel. Although these points of reasoning have been accepted for some time and really arise from common sense, they are worth re-stating in this context.

The main requirement imposed on a more general and more flexible assembly cell can really be expressed as the ability of a cell to cope with unforeseen circumstances. In general, it is not necessary to give the cell enough 'intelligence' to be able to conclude itself what the actual error condition is under all circumstances, nor is it always necessary to generate new recovery strategies automatically - in the first instance. For any practical application, the ability to remember some conditions and how to recover from them will already go a long way towards the increase in productivity and flexibility of this automation process.

To summarise, the cell:

1. needs to be able to recognise the encounter of unforeseen circumstances,

2. needs to be able to remember similar circumstances from its sensory information and then to remember what actions it has used before to accomplish a recovery in those circumstances,

3. must not allow time to be wasted on the identification of errors which are not there,

4. must not exclude the operator from the decision making process.

The last point is particularly important in the non-technical context. It keeps the expertise of the operator available for the smooth running of the cell. From a social point of view, the cell's and the human's capabilities can complement each other. In practice, that means that the 'power of the human brain' can provide solutions which would otherwise only be possible to be found with uneconomical computing (and sensing) equipment.

On the other hand, many relatively simple actions can be executed automatically without relying on a human for tasks for which he is not as well suited. It is a well known fact that repetitive tasks can lead to a lack of concentration and associated reduction in productivity. Benefits can thus be gained for both the operator on the shop floor and the company employing the flexible assembly cell technology.

The requirements set out above are addressed in the following chapters and a systematic approach to the technical solution of many assembly situations is suggested. It is based on the analysis of the 'general' assembly process and incorporates these conclusions from the observations of a 'real life' cell performance.

Chapter 3

Generic Assembly Processes

The main area of concern in this chapter is the analysis and decomposition of the general assembly process. A suitable framework is suggested which allows a systematic approach to the various decision stages required in the design, maintenance and operation of assembly cells. This framework is based on the separation of assembly processes into the stages of Feeding, Transport and Mating. Such a representation as distinct sub-processes makes feasible the idea of identifying *success criteria* to ensure the correct outcome of each stage. The general nature of the analysis implies that this framework is not restricted to robotic assembly systems but is valid for all automated and manual assembly.

3.1 Assembly As Positional Transformation

The constituent elements of an assembly process, Feeding, Transport and Mating, are actually present during the manipulation of each component, either during automatic or manual manipulation, but are more accentuated in flexible assembly. Figure 3.1 shows the two basic situations which are encountered. Initially (Figure 3.1a), the components under consideration are apart, the very reason for the desire to assemble them. Finally, the parts have a very well defined relative positioning (Figure 3.1b) and an assembly process provides the means for this positional transformation. This chapter attempts to show that the apparently bewildering variety of assembly actions can be grouped together into generic assembly processes if an appropriate framework for a general description is established.

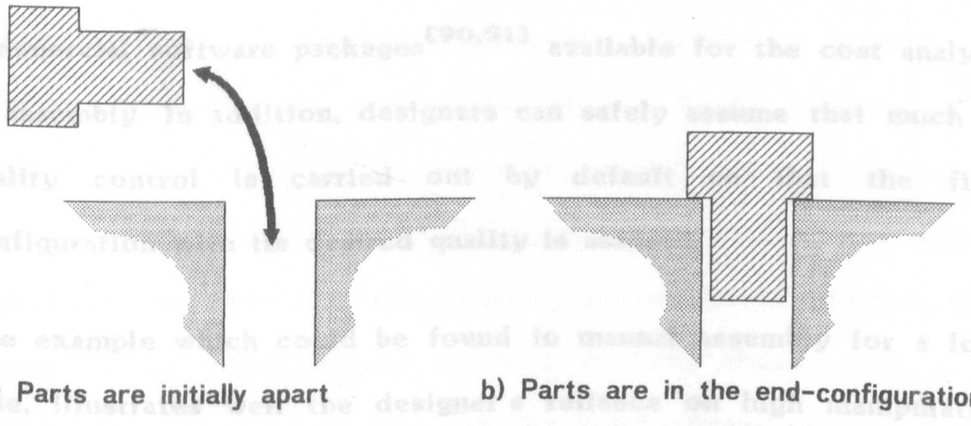

a) Parts are initially apart
b) Parts are in the end-configuration

Figure 3.1: Assembly As Locational Transformation

The basic problem which has to be addressed in any implementation, arises from the fact that the desired end-configuration is actually well defined, whereas the initial and intermediate configurations are not. Any system has to be matched appropriately to provide the ability to generate that end-configuration.

3.1.1 Characteristics of Manual Assembly

A closer look at the manual assembly process reveals quickly the superb ability of most human operators to learn a task from an imprecise description. The word superb has been chosen deliberately, because it expresses the human's superiority of being able to perceive and interpret a given situation. Thus any factory supervisor can usually rely on simple showing of the task and then operators are capable themselves to work out how to generate the missing information and to improve any deficient skills. Such sensing and learning is taken so much for granted up to now that it is very often incorporated into the design of products without a 'second thought'. There was also no need to provide separation and detailed analysis of individual sub-tasks in the assembly process. Much work[78] carried out in the field of assembly automation has concentrated on economic aspects, and has resulted by now even in commercial software packages[90,91] available for the cost analysis in assembly. In addition, designers can safely assume that much of quality control is carried out by default so that the final configuration with its desired quality is assured.

One example which could be found in manual assembly for a long time, illustrates well the designer's reliance on high manipulative skills:

In order to secure printed circuit boards (PCB) at a clearance distance above some base plate in a casing, spacers are used which are inserted between the PCB and the case. The whole arrangement is then secured by bolts which have to be inserted through the corresponding holes in the PCB, the spacer and the case, before a nut can be tightened onto the other end. The main problem is the uncertainty of the relative positioning of the components to each other, as well as their temporarily unsecured positioning, but even this exerts only a relatively minor influence upon the overall efficiency of humans. The advent of automation techniques quickly establishes the difficulties associated with these types of procedures and this particular example has virtually been eliminated in modern assembly systems.

3.1.2 Characteristics of Dedicated Assembly

At the other end of the scale, the design of fully automated and dedicated assembly machinery has also blurred the distinctions between any sub-processes which may be involved in reaching a desired state. It is quite obvious that a general consideration of assembly is not required for a single component, and machines will work at their best when designers are able to *cut corners.* The majority of dedicated assembly equipment consists of a sequence of individual actuators; at each stage, only a single component is added to the product. Only one - or possibly two - degrees of freedom are necessary, since fixtures can be arranged with adequate precision to accommodate the limited movements. In addition, the complexity of single action stations is kept deliberately at a minimum; for example, a part can be acquired on the up-stroke (through appropriate mechanical or pneumatic control arrangements) and inserted into its planned position on the down-stroke.

The resulting improvements in productivity make the efforts spent in product specialisation worthwhile, but they prevent the use of that equipment for other products, or even for variations in the same product. Of course, even specialised equipment can eventually be made to be adaptable, but this requires a relatively large effort in most cases and the degree of adaptability still remains too low for the ever changing demands of the market place.

A typical example of dedicated assembly can be found in rotary transfer mechanisms which have many (typically up to ten[2]) single stations attached to them. A component is fed initially and then a very simple action is performed on it at each stage;

[2] In-line indexing and free transfer is preferred for more than ten parts

examples would be a fixed rotation, an addition of another component or application of adhesives. The final station removes the sub-assembly from the system. The use of such techniques usually implies a deeper understanding of assembly automation by the designers. Such dedicated assembly may be accompanied by further advanced methods which are well established in areas like computer-aided drafting. Increasingly, design rules for assembly are also incorporated[56,57,58], exploiting the experience accumulated in one particular company - or elsewhere - in order to install more economical production systems.

One of the areas of improvement which still has a large scope for increased utilisation, is the integration of demands of production systems at the time of the product design[41,56,57.58].

3.1.3 Characteristics of Robotic Assembly

Robotic assembly, in contrast to both manual and dedicated processes, is very much concerned with general applicability, including the ability to change from one product to another as quickly as possible. Although a truly universal assembly cell would be highly desirable from a planning point of view, the prohibitive costs incurred by providing system devices to cater for all assembly situations would, purely on economic grounds, rule out such a cell. Further, even optimistic estimates of the computing power available to such a cell would incorporate still only a small fraction of human intelligence, and consequently very little could be taken for granted or assumed *to be done anyway*. Thus it is still necessary to describe the actual process in sufficient detail to a cell so that the robot is able to execute the tasks correctly and reliably. Providing this information for the successfully completed assembly process is the aim here.

In any event, economic considerations will eventually contribute to the final design of the cell, but such financial justifications are not addressed here. Over the years, the latter has proven to be a research topic in its own right and the reader is referred to the relevant literature for further information[13,29].

3.2 Problem Decomposition

In order to identify those parameters which contain the essential information, the total task of assembling one product has to be broken down to reduce the potentially bewildering complexity.

At each stage in the execution of an assembly task, the cell has to be able to identify successful completion of the preceding action in order to continue with the manufacture. Assuming such a capability, the cell can be left to run unsupervised with much greater confidence than is currently possible. In the event that the cell halts execution, the production engineer can be assured that products are not made with faults nor that a wrong course of actions is embarked on. Thus the destruction of expensive systems caused may be only by a comparatively minor error, is avoided. In the worst case, the overall productivity is reduced. Periodic checking and confirmation of continuation implies a sequential nature of the assembly, and it is precisely this argument which provides the basis for the investigations into the order and nature of generic assembly tasks.

3.2.1 Components

Each product leaving the assembly cell will consist of a number of components which have been *put together* in the right order and with the correct relationship to each other. The number of components is usually quite small[13] for a single station, typically in the order of one to two dozen. The upper limit is mainly imposed by the amount of space available[46] for the placement of feeding devices around a robot. Also, the total number is quite often influenced by related manufacturing considerations. For example, costs associated with adding a high value to a product may require a brief interruption of the strict assembly process in order to allow for additional, more stringent quality checks. Equally, a reduction of tasks per cell may help in avoiding expensive re-working, or may be due to an overall increased factory output by assigning further tasks to machines operating in parallel.

Component Definition:

A component can be regarded as being a single unit which does not require any specific assembly actions within the context of a single station. Naturally, they can be products of other assembly stations, either having been produced in the same factory or quite often may have been bought from a supplier as ready-made units.

The motor of the FLYMO lawn mower (Figure 1.1) provides a typical example for latter type of unit. It is itself a complex arrangement of individual parts, and is only produced economically for the lawn mower because the expertise and facilities exist at a different company.

Component Sequence:

Lastly, the assembly station may have to produce a unit (or sub-assembly) as part of its own operations before it can proceed to incorporate the newly produced component into the main product. The point at which the sub-assembly is complete, is taken as the change from several individual components to a single component, and it can be treated in the same consistent manner as those which are obviously single units.

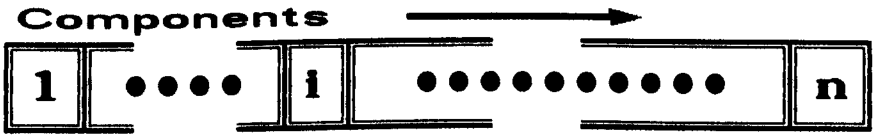


Figure 3.2: Component Sequence

Thus a succession of components is dealt with by the robot (compare figure 3.2), where at any instant there is only one component which is being manipulated. Consequently, the assembly analysis can now investigate the generic manipulation tasks. These tasks in turn are extendable to each of the individual components.

3.2.2 Feeding

The first important stage in any assembly process is the feeding of a component. This is not new, as feeding[46] has been identified as a major obstacle to flexible, economic robotic assembly. It effectively encompasses the majority of the auxiliary equipment. Nevertheless, without fail, the component has to be within the working envelope of the robot as shown in figure 3.3, otherwise it has no meaning as far as the assembly is concerned. The description of the feeding stage has to contain the specifications of those parameters which are clearly identifiable within the cell - by whatever means - and give a measure of the success of the feeding stage.

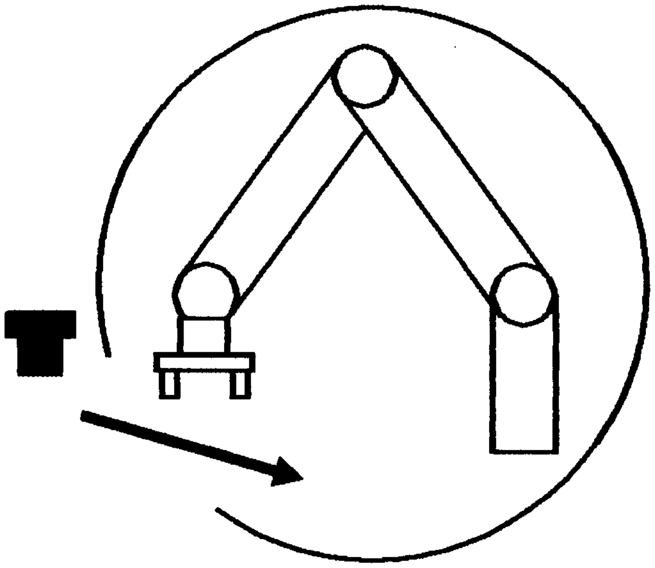


Figure 3.3: Component Feeding

The single most important parameter in this context is the final position of the new component once it is within the reach of the robot. Implicitly, the information concerning the component's presence at the expected pick-up point is also contained in it. Positional variations arise from the uncertainties associated with such a feeding point, including a variety of stable positions and the number of degrees of freedom within which the component is able to move given the current feeding arrangement.

For example, a pallet of components which has been carefully machined to fit the shape of the component will probably be able to specify all six degrees of freedom with sufficient accuracy for subsequent operations (compare figure 6.7). So the cell controller will know where to send the robot in order to achieve the correct gripping position. On the other hand, if the components can enter the cell with a very high degree of disorder, then additional actions may have to be taken in order to reduce the uncertainty to a more acceptable level.

That was the case in the diode assembly of Hill and Burgess[1] where the pn-junctions of high power semi-conductor diodes entered the cell without any pre-determined order: The area in which the pellets could be positioned was restricted, but the dimensions were very large with respect to the size of the silicon disks. The additional action in the feeding stage consisted of utilising the combined information from overhead and gripper-mounted cameras. The exact position could thus be determined by involving the robot itself to acquire the relevant information. Success was achieved by locating the disks repeatedly at the same point in the field of view of the gripper mounted camera so that the final pick-up point was always at the same taught offset.

The knowledge about the final position for pick-up can now be utilised by the cell control system in order to send the robot to acquire the component. After it has been gripped, the transport phase starts.

3.2.3 Transport

Although the transport of a new component to the point where it can be joined to an existing sub-assembly (see Figure 3.4) is not actually part of assembly itself, there exists an obvious connection to the successful outcome of the assembly. This separate stage is justified because of the different emphasis placed onto it during its execution. Here, the manipulator control system assumes the highest importance, as it is generally desired to spend a minimum amount of time in these *unproductive* transition periods.

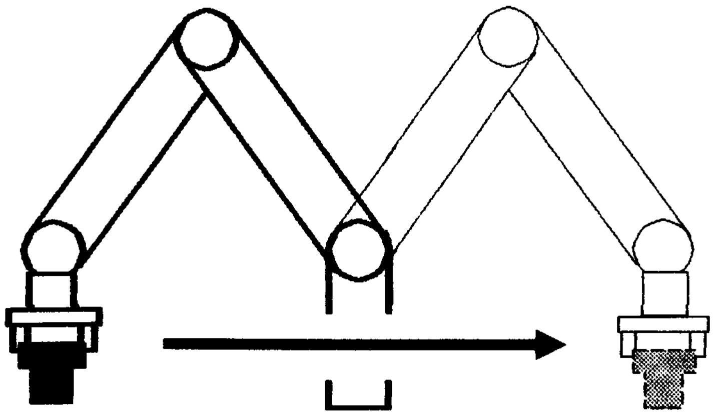


Figure 3.4: Component Transport

Dynamic Control

The important aspects of dynamic and kinematic control are clearly and fully acknowledged in this work. Most of the manipulator's time[3] is spent in transportation[1] and the excellent work carried out throughout the world[26,52,59,60] will undoubtedly contribute to a significant reduction in the overall production time. However, in this work, the manipulator is regarded as a machine which can be *bought in*. There will be little opportunity for most production engineers or cell designers to adjust or improve the performance of the robot itself.

Without doubt, the eventually achieved cell performance is strongly related to the devices which are available to the control structure and whereas the choice of equipment assumes a basic understanding of its functions, the various levels of complexity through a

[3] Correction: Transportation is usually one third of the total time

complete installation demand a shift in emphasis away from individual devices and towards applications. Assembly is the application under investigation here.

Trajectory Generation

Closely related to the aspect of control is that of trajectory generation, in particular, obstacle avoidance. Clearly, the moving devices must not be allowed to collide with any other parts in the cell, or the resulting errors will almost certainly be irrecoverable, even by the most sophisticated Artificial Intelligence techniques. Again, for the purpose of assembly, it is valid to assume that paths are either taught manually, in which case the obstacles are avoided by visual checking during the teaching phase, or the trajectory will result from an automatic path-planner. In any case, this field of research is defined distinctly from the topics in this thesis, and relevant work is also carried out throughout the world[82].

Transport Parameters

The transportation phase also needs to be parameterised to the extent that its success - or failure - can be identified. In fact, this is one of the more difficult aspects, since the requirements are not as easily associated with physical parameters as they are in the feeding and the mating stages. Nevertheless, the one single parameter which can be said to characterise this situation is the one of *configuration*. The intentions by using that expression are clarified in the following example.

If a robot has picked up a part from the feeding point, then the relative positioning between the part and any gripping device will be known. Given this information, the actual parameter sensed may be only the gripper opening of a parallel jaw gripper. The interpretation of this one parameter is carried out within the context of *a priori* information; for example, if the jaw opening is zero at the end of the transport stage then the part may well have been lost on the way. Any other opening apart from that measured at the time of pick-up, indicates a change in the configuration.

In any specific case, the designer of the cell has to decide - possibly with appropriate software tools[31,47,48] - what physical characteristics can be measured so that the configuration can be deduced. In the final instance, the choice of sensors to identify the configuration will be a compromise between cost and sophistication of sensing mechanisms and the ability to maintain the relative positioning by mechanical means. Typical examples for the latter can be found in automated tool and/or gripper changes which show a comparatively high degree of flexibility while maintaining the advantages of mechanical restraining.

The eventual interpretation of *configuration* is certainly very dependent on the products and the cell design. This is another instant at which appropriate software tools, such as expert systems, are highly desirable in order to assist the cell designer and finally the operator. Efforts in determining a maintained or constant configuration are also a function of the range in which the robot and its gripper are operated. For example, the FLYMO lawn mower power unit and gripper have a combined weight which approaches the load carrying capability of the ADEPT robot. In the implementation of the demonstration cell, the gripper/motor configuration was associated with the weight and slower motion with effectively more frequent force readings assured the maintained configuration.

3.2.4 Mating

The successful termination of the transport stage is also the start of the part mating, which is the assembly act itself (compare Figure 3.5). A variety of solutions is offered in the research literature[8,61] and important results and conclusion have been published[62]. Most of the previous approaches have attempted to model the assembly process by mathematical means, with varying degrees of success. This thesis offers a slightly different approach to the mating processes, associated more with the knowledge of HOW TO DO it, rather than a precise but also very complex description in equations.

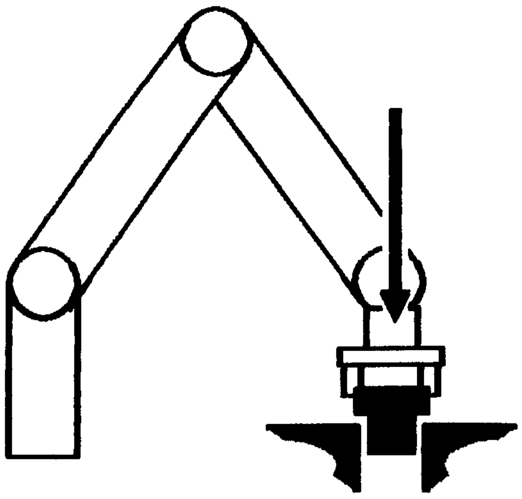


Figure 3.5: Component Mating

The following chapter on *Information Spaces* will discuss the details about this knowledge-based approach in detail, but it is appropriate to outline the ideas here in order to provide the correct context for the discussions. In general, the mating stage it characterised by the fact that the parts are actually in contact, but are still moving with respect to each other. This provides the main difficulty for the mathematical modelling approach, since a number of assumptions must be made, especially with respect to frictional forces. These assumptions are difficult to specify and may change significantly throughout different batches and even within a single batch.

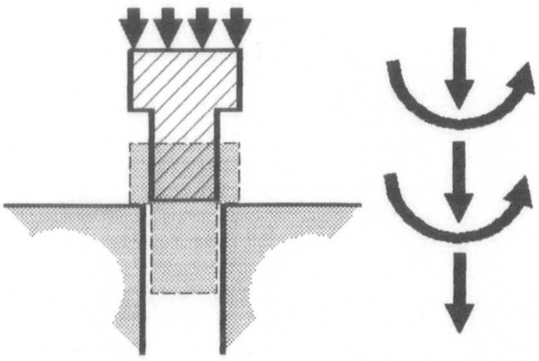


Figure 3.6: Force/Position Segments During Mating

However, contact and motion allow a specification of the physical parameters of force and position within certain limits so that mating is characterised very effectively by force/position segments (see figure 3.6). In specific cases, many segments may have to be traversed in order to arrive at the final, desired end-configuration. Naturally, force/position may equally well be torque/rotation segments. The segments themselves can be derived from the geometry of the mating parts. They are significant, in that they offer facility for *educated guesses* in discretising the forces and positions, thus removing the dependence on precise values during system set-up. During the design of assembly processes any special requirements on tolerance or similar restrictions can easily be incorporated.

3.2.5 Feed-Transport-Mate (FTM) Concept

By providing such a systematic description (see figure 3.7), a method has been established which is applicable to all *engineering-type components.* These are defined as those which keep their shape during manipulation, and do not undergo significant distortions as might be the case for textile parts in the assembly of garments.

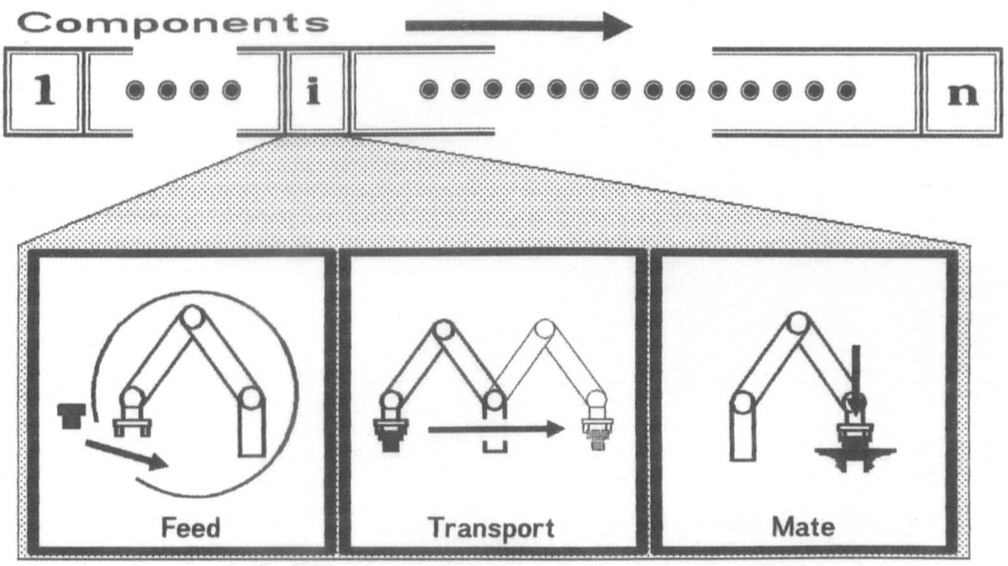


Figure 3.7: Feed-Transport-Mate Concept for All Components

At this stage in the research, it is not possible to make any further definite statements about the applicability to components other than the rigid ones.

A summary at this point is:

Each component has to be

fed, transported and mated

to the product which is assembled in the cell.

3.3 Extension of Component FTM

The approach to assembly as set out before is very similar to 'object oriented' programming of industrial robots[31,44,45,47,48]. In these systems, components in the cell are also considered with respect to their physical relationship to each other. Actions are a result from various properties of the objects (predominantly geometrical, like surfaces) and the desire to change the relative positioning in order to achieve a desired configuration.

Such a description suffices for many components arriving in the cell because it, too, follows the assembly process throughout all its stages. However, the emphasis is placed on the component and its properties, neglecting additional auxiliary actions which form an essential part in achieving successful assembly. Shifting the emphasis onto the assembly *process* does not dismiss the properties of objects, rather considers the whole area, objects and associated actions, including sensing.

Any complete description of a process performed within an assembly cell must therefore be able to accommodate such a variety of tasks. This is best achieved by developing the system in a top-down fashion, but before the proposed system is developed further, two commonly used methods will be given in order to illustrate some of those important auxiliary tasks.

For example, one way of reducing the total amount of time spent in travel, is the provision of multiple grippers on one robot. Consequently, the feeding, transport and mating stages for a single component can be sufficiently split up so that they cannot be executed sequentially, but several components need to be fed and picked up before transportation commences. A second example for additional processes to be performed 'en route' is the need for auxiliary tooling. In this instance, the cell has to feed and pick up the tool, then feed and pick up the component before its transportation can begin. This is a valid argument for the general case, as it is very difficult to provide a single gripping mechanism which will be able to grasp the different shapes encountered in individual components.

Thus this method has to be augmented to look at assembly in a slightly different way. In fact, it is the *process* which will be considered again, as was the original intention, rather than the *components* of the product.

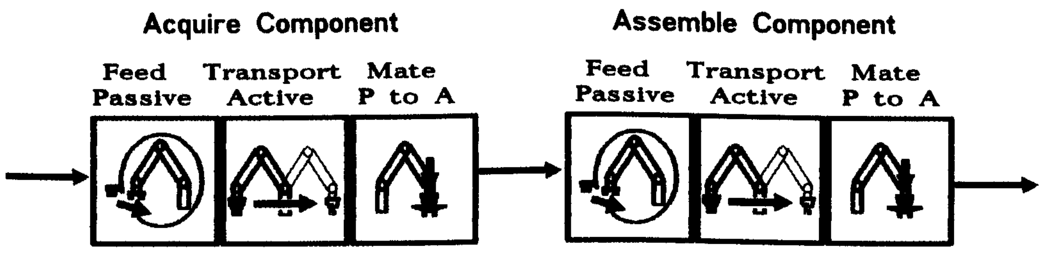


Figure 3.8: FTM-Pair Concept

3.3.1 FTM-Pairs

As an alternative to FTM stages for each component, the process can be viewed with respect to actions which the cell has carry out. After the feeding of one component has been performed satisfactorily, the robot has to execute two separate actions associated with each component: *pick it up* and assemble it by *putting it down* at the correct place (compare Figure 3.8).

3.3.2 Active and Passive Regions as Mating Features

Although this viewpoint introduces new arguments, the previous description is wholly included. These views are not mutually exclusive, but in fact complement each other well for the reasons discussed in the following paragraphs.

It was established in this chapter that assembly is a positional transformation with a specific desired end-configuration. That argument is valid in exactly the same manner for both *pick-up* and *put-down* tasks, if both are considered to be different generic forms of the assembly process. Each time we do have to 'mate' one part to another, the gripping mechanism is mated in the first instance to the part at its feeding point, resulting in a new configuration. The second time the process is executed in the conventional way, assembling the new component to the existing structure.

An example given in a *pseudo* programming language illustrates these ideas:

```
BEGIN—FTM-PAIR
   FEED(Component-A);  (* present component at its feeding place *)
   TRANSPORT(Default-Gripper); (* Move robot to the pick-up position *)
   MATE(Default-Gripper, Component-A); (* Hold component in current gripper*)

   FEED(Component-B); (*Already in place - no physical operation *)
   TRANSPORT(Component-A); (* So that it can be assembled *)
   MATE(Component-A, Component-B);
                              (* This achieves the desired end-configuration *)
END—FTM-PAIR;
```

In this context, it is now convenient to talk about *active* and *passive* regions within assembly processes. *Active* is defined as that region which is being manipulated by the robot and conversely, *passive* is the region to which something is done. In general, this nomenclature may not reflect the true situation, but it is very convenient to retain this distinction of the two regions which are important during the mating.

3.3.3 Regions of Importance

This now raises the questions about what the regions of importance actually are. A general and 'watertight' definition is outside the scope of this work, but in order to establish the meaning of the regions within this context, an explanation (by example) will follow. It is obviously worthwhile considering the work of other researchers who are investigating related subjects in a complementary way to this system[31,49].

The end-configuration is usually very well defined in the Cartesian space, particularly, if the product has been designed with the aid of a CAD system. Due to that requirement, very specific geometric restraints have to be fulfilled in order to achieve that configuration. This will involve only a limited number of 'faces'[31] and their physical relationship. A couple of examples may help to clarify this assertion.

1. **Placement:**

   Amongst the many different assembly action described and classified by K.G. Swift[27], placement is one of them. Suppose a block - like those encountered in the standard 'block' world in AI examples[50] - is to be placed onto a table. Now one of the faces of the block will be manipulated such that it will be flat against the top surface of the table. The regions of importance arise from these two faces, and the fact that there will be a quantifiable force exerted between them which can be measured at the point of contact.

2. **Insertion:**

   The next common, but slightly more complicated assembly action is insertion (like that in Figure 3.1). Now there are additional constraints[59,63], which change during the mating procedure. Without exception, the motion of the peg has to be executed such that it will be clear of any material surrounding the hole. Once contact is established, then the planar restrictions will assume importance, in that only very small sideways motions are allowed, but the peg will be free (assuming sufficient clearance) to move in the vertical direction. The region of importance is the physical region throughout which the peg and the sides of the hole could come into contact. The limits are set by the relative dimensioning of the hole, the peg diameter and the depth to which the peg has to be inserted.

Clearly, the precise definition what constitutes regions of importance is again very product specific, but it is also clear that any assembly will always involve only one *region* at any one time on each of the components which have to be mated. This holds regardless of how complicated the overall product assembly appears to be.

3.3.4 Recursive FTM-Pairs

Keeping the explanations of active and passive regions in mind, the FTM stages are very suitable to describe the pick-up operation as:

1. ***Feed*** the ***passive*** region. It is necessary to establish the point at which mating can be executed.

2. ***Transport*** the ***active*** region to the point of mating. The precise positions at which the mating starts or transport ends are debatable, but it is argued in this work that the approach and departure distances form part of the mating process (see chapter 5). The reason for that can be found in the geometry of the components which dictates not only the end-configuration, but also the collision avoidance in the *close vicinity* of the mating point.

3. Finally, ***mating*** between the ***active*** and the ***passive*** regions can take place, possibly monitored by the *Information Space* method (see chapter 4).

The case for the *put-down* activity can now be argued in the same way, with the proviso that the regions of importance are adjusted to the new component as the active one, instead of the gripper and any existing, partially completed sub-assembly as the passive one.

Being able to view the addition of a single component as two separate generic assembly actions (see figure 3.8) allows for the inclusion of the various situations which were not possible within the previous limitations at the object level. Each *pick-up FTM* is adding a piece to a conceptual *stack* at the end of the robot, and the success is only given when this extra piece has also been removed. Again, some examples (comparee section 3.3) may clarify this method.

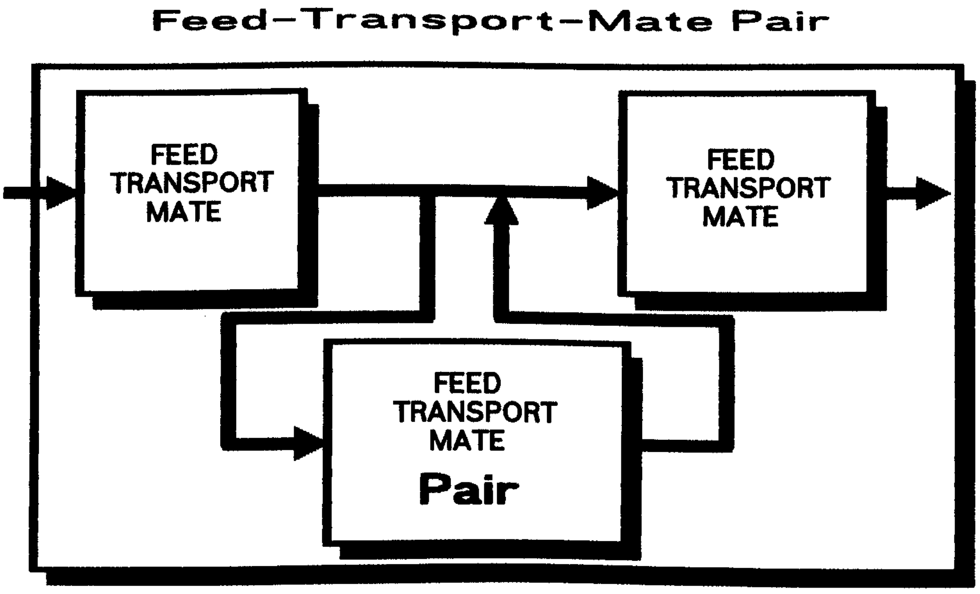


Figure 3.9: Recursion In FTM-Pair Method

1. ***Multiple grippers:***

   Suppose, a turret gripper is mounted at the end of a manipulator. This arrangement has more than one, usually three, sometimes up to eight, end-effectors on it. The latter has been proven useful in PCB population systems. For each component handled, the feeding stage will be carried out as before, presenting the passive regions ready for mating. The first *picking up* will mate the first gripper to the first component. The second passive region is fed and the second active region - now a different gripper - is transported to the second passive region and mated. Transport, in this case, also involves the rotation of the turret gripper.

Now there are two components on the robot, both of them have to be removed from the grippers and mated to the correct place on the PCB, for example. Each physical addition has also increased the *stack* in the computer which keeps track of the sequencing.

It is probably worth noting at this point that the vast majority of assembly operation will be symmetrical with respect to the removal of added parts on the principle of *last-in-first-out*. That is convenient and easy to follow but not necessarily the most efficient way to execute the process with respect to overall productivity. This line of thought can - unfortunately - not be followed up within the context of this research.

2. *Auxiliary Tools:*

This case is relatively easy to see, since the physical configuration during the assembly process does not allow for alternative executions.

When a second tool is needed (see section 6.1.4), then the default gripper will be the active region and the gripping area on the tool the passive part. Once these are mated, the active region is considered to be the gripping area on the tool and the passive area is on another component such as the motor. The new configuration can be expressed in very familiar terms from the descriptions of serial robot manipulators[26], with compound transformations[73] being added to the gripper frame.

Once the component has been mated to its rightful place in the product, the tool is left on the gripper. Designating the tool holder as the passive region, it is fed (almost certainly without any physical action) and the tool can be mated to the tool holder as it is put back into its resting place.

The *pseudo* programming language is useful here again to illustrate the second of the above cases slightly different form:

```
BEGIN—FTM-PAIR (* This pair deals with the Motor Tool *)
   FEED(Motor—Tool); (* Put auxiliary tool for motor within reach (NULL) *)
   TRANSPORT(Default—Gripper); (* This is the one picking up the tool *)
   MATE(Default—Gripper, Motor—Tool); (* Pick up tool *)
   BEGIN—FTM—PAIR (*This one deals with the motor *)
      FEED(Motor); (* Identify position within the pallet *)
      TRANSPORT(Motor—Tool); (* Send robot plus tool to the motor *)
      MATE(Motor—Tool, Motor); (* Pick up motor with Tool *)
      FEED(Motor—Jig); (* Jig is fixed: Physical NULL operation *)
      TRANSPORT(Motor); (* Using 'other end' of motor as active region *)
      MATE(Motor, Motor—Jig); (* Place motor into temporary fixture *)
   END—FTM-PAIR; (* Motor placed OK, tool left on the robot *)
   FEED(Motor—Tool—Holder); (* Also fixed: NULL operation *)
   TRANSPORT(Motor-Tool); (* Move tool back to its resting place *)
   MATE(Motor—Tool, Motor—Tool—Holder); (* Tool put back for next time *)
END—FTM—PAIR; (* Default gripper now free for next Dual FTM*)
```

3.4 Generic Mating

The last example, in particular, highlights the importance of being able to establish the success of the mating phase as a crucial part of the overall success. Since individual operations within the assembly cell have now been sufficiently separated to identify their own characteristics, the work is ready to attempt to provide criteria for the classification of generic mating processes, which clearly form the heart of any assembly system. Without these criteria, classifications remain essentially guess work, even if the proposals are quite often based on the valuable experience of individuals.

The next chapter will introduce such criteria and explain the involvement of sensors in a generic way, using the assembly information rather than individual sensing methods. It thereby differentiates between solutions which are valid at specific instances and methods which are usable throughout a significantly wider range of applications.

Chapter 4

Information Spaces

The outline for assembly processes, which are at a basic level common to all assembly methods, has been established in the previous chapters. The recursive frame work of FTM pairs accommodates the complete description of robotic assembly tasks. Although **Feeding** and **Transport** are regarded as separate subjects in their own right, the identification of their essential parameters in assembly (see chapter 5) defines the interfaces and hence establishes clear boundaries and transitions between the phases.

The remaining phase of **Mating** is at the heart of success for the overall process. The idea itself about using Information Spaces for the Mating Phase has already been introduced in a crude form in chapter 3, and in this chapter, the Information Space concept is explained in detail. Systematic sensor integration is emphasised and its relevance to error recovery discussed. Potential benefits of employing sensors are to be found in their **generic** use, by removing the dependency on individual products and individual cell implementations.

The originality of this work lies in the completeness of assembly descriptions as well as the ability to integrate any type and any number of sensors without compromising the structure. This promises to overcome many drawbacks experienced in current industrial applications where sophisticated sensor systems are usually specially designed for one task and cannot be easily transferred to other circumstances.

4.1 Sensor Integration

The system proposed here introduces a description of mating which allows systematic integration of sensing systems into assembly procedures. Even with the current state-of-the-art of sensors, the variety of commercial and proposed mechanisms is impressively large. However, the usefulness of specific items is not always obvious, except with respect to that single process which it has been designed to monitor. Many sensors involved in the control of large scale production facilities are manufactured with relatively simple measurements in mind, quite often for the identification of a single physical quantity. Temperature and pressure are typical examples, and complexity is introduced often by the scale of the operation alone. As far as robotic assembly is concerned, popular vision and tactile sensors are of course only a sub-set in that set of currently available sensors, but they probably represent the most sophisticated ones.

4.1.1 Information Content of Sensing

Sophistication bears with it its greatest promise, as well as the stumbling block, for more wide spread application. This apparent contradiction is best explained taking the example of vision systems. On the one hand, there is a large amount of information contained in a captured image on a frame store. But on the other hand, the interpretation is not at all well defined, unless the environmental circumstances in which the picture was captured are well known. This phenomenon can be partially attributed to ill defined requirements of the tasks, because it is often not clear what should be measured. To some extent, the ambiguities can also be attributed to the fluid boundaries between physical parameters. The latter can be seen in another aspect of vision systems.

Images are acquired from scenes by focusing a pattern of varying light intensity onto a light sensitive array and then converting this into a computer readable form. Hence the light pattern contains information about shapes and light intensity. This leads to significant problems when the information is extracted from sensed parameters which are themselves influenced by more than one variable, and are thus not clearly separable. Hence, for part of an object, uneven illumination across a single image or the presence of colour may produce the same grey level as the background - confusing the distillation of the desired information.

4.1.2 Confidence in Sensor Signals

Even if such information is clearly identifiable, not all parameters can be sensed by a single measurement mechanism. A variety of signal sources contribute to the information, and there are several important issues which must not be overlooked when sensors are employed.

- ***How reliable are sensor signals?***

  The desire to work in a real environment must take into account that sensor signals can be corrupted by noise or even that particular mechanisms do not provide reliable data[32,53]. Excluding the possibility of a bad choice of sensor (which cannot be dismissed entirely), there still remain problems such as physical contamination of the sensing element or deterioration due to age, wear or damage. For example, a crossfire sensor may be blocked by some environmental condition like dust accumulation or dirt deposited by components themselves.

- ***How do individual sensors contribute to the desired information?***

  In this context, it is feasible to assume that noise and other forms of corruption can be dealt with, but the question of significance remains. Sensors with a high information content can be interrogated with respect to a variety of different aspects, whereas simple binary sensors (such as proximity) give only a direct Boolean result, related to a single event.

  Nevertheless, this does not impose an automatic choice or significance, since the Boolean result can assume more importance in certain circumstances. For example, consider again the representation of colour information within commonly encountered vision systems. If it so happens that colour is not important for the assembly, but that an arbitrary succession of different coloured objects have to be located, then it may well be the case that some objects are merged into the background by assuming the same greylevel. An interrupted light beam can now be interpreted as the presence of a component with a higher degree of confidence than would be provided by a grey level vision system.

- ***How is conflicting interpretation resolved?***

  If two sensing systems have the same significance, then there may well be the situation in which the interpretation of the signals is contradictory. The resolution of this problem depends on characteristics of individual sensors, such as past reliability, or could solved by a pre-determined resolution to this conflict. Sometimes, additional information from a third sensor may have to be acquired or - where applicable - the possibility of equal weight could be avoided at the design stage. However, such assumptions are not valid in general, particularly if the system is automated in the ideal sense when almost all stages are generated without any further intervention by humans.

Despite the importance of these questions, the investigation of such issues could not be carried out within the scope of this project. However, later explanations show the existence of clearly defined interfaces between the stages of information handling, and these problems can be taken into account effectively *on the other side* of the interfaces. It was therefore decided that it is justifiable to concentrate only on the assembly process and that the previous lines of research can be followed up at a later stage.

4.1.3 Complete Information Integration

Interfaces like those above identify areas of *responsibility* within a systematic description, and thus the important issues within those areas are emphasised. In addition, it is reasonable to assume that - as technology develops - new solutions will emerge for many problems which appear to be insurmountable at present. As requirements for generic processes (especially in assembly) crystallise and as the body of experience grows, new sensing mechanisms may be proposed which provide suitable information more directly or which are potentially more economical. So, the set of currently available sensors is by no means exhaustive.

Consequently, the proposed integration method must be *complete*, at least in the sense that

- the relation between any distinct information sources can be described unambiguously within areas of responsibility in a process description,
- this system can accommodate additional (or less) information sources and,
- measured values can be interpreted within the varying context of a specific task.

Information Spaces provide a basis from which these requirements can be built up and the concepts will be explained in the remainder of this chapter.

4.2 Information Space Concept

The Information Space system forms the heart of generic assembly tasks and is based on the sub-division of process specific information into ranges of values which assume significance at some point in the assembly. Any information which can be sensed in the cell and is quantifiable can be assigned to an axis onto which the relevant sections are mapped. There are as many axes as there are *information* sources, and as the question of *sensor contribution* has discussed in section 4.1.2, a single axis does not necessarily represent just a single physical value, but may well map out some combination of measurements.

4.2.1 Interpreted Meaning of Sensing

Information axes represent some *meaning* of the sensor values from which it has been derived. Since the interpretation is carried out within the context of a well-known process, generality is not lost, and emphasis is placed on the identification of those parameters which can be used to determine the success of an operation.

For instance, the distance between a workpiece and its first point of contact with the mating part can be worked out from their physical relation. The component's location in the gripper and the target position are known from the system set up. Taking the manipulator's current Cartesian position, a single value (in this case distance) can be formed. Although this is a simple example, the specific information of importance is the *distance* in this context, and no direct measurement from present day sensors would be obtained easily.

As long as the active and passive regions do not touch prior to the first mating attempt, any values falling between $-\infty$ and the position corresponding to the start of mating can be grouped together into one single section (see Figure 4.1). This is usually the pre-contact configuration in the assembly process, ensuring sufficient clearance at the end of the transport phase and just before the mating procedure itself.

For convenient referencing, names may be assigned to each of the sections, here *pre-contact* is suitable for the situation just described. Figure 4.1 also offers suitable nomenclature for further sections during assembly. **Contact** describes the situation in which the components are moving with respect to each other, but are touching, with **Overshoot** being a third, but undesirable state.

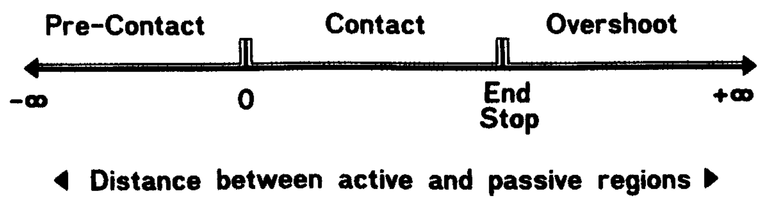


Figure 4.1: Example for Information Sub-Division

Sub-divisions on their own are usually not very meaningful on a single axis; only when they are placed into the correct context is it possible to interpret them (see section 4.3). One typical example for context dependent readings are those experienced in force sensing. The actual value of the forces will vary substantially throughout the mating operation and, whereas no significant forces are expected during the pre-contact phase, the same value at the final placement has a different meaning and may well indicate an error.

Already, the section boundaries on axes provide well defined decision criteria for the cell controller. If a value is encountered which does not fall within the set limits, then clearly a potential error situation has arisen. The *specific* causes can only be determined for *specific* circumstances, and an example for the deviation from successful sensor space regions may well be a component fault. Although no more statements can be made about the errors in the general sense, the axes themselves cover any error condition which may arise.

4.2.2 Range of Sensor Information

Sub-divisions also integrate and relate *all* assembly situations systematically, including those which have not been anticipated during the design of the cell. Even more importantly, those combinations of values are included which were not considered important enough for further investigation with current programming systems, or which have simply not been foreseen as potential error sources.

Obviously, limits are set by the physical range of sensor values. Theoretically, though, its expansion is infinite and in practice only limited by the range of the data representation in the computer, which usually does not impose any practical limits. Also, there are no pre-conditions on the nature of information measurements, and discrete values - including Boolean information - can be grouped together unambiguously, as well as sub-ranges for continuous signals.

4.2.3 Empirical Division into Sub-Spaces

Before any assembly can be attempted, axes and their sections need to be specified. At this stage of the research, there are no sophisticated, automated tools available which could help during the generation of information spaces. Such support would undoubtedly be necessary for use within a practical CIM system - particularly for uninitiated or inexperienced users. For the demonstration cell, it suffices that the sections have been identified empirically.

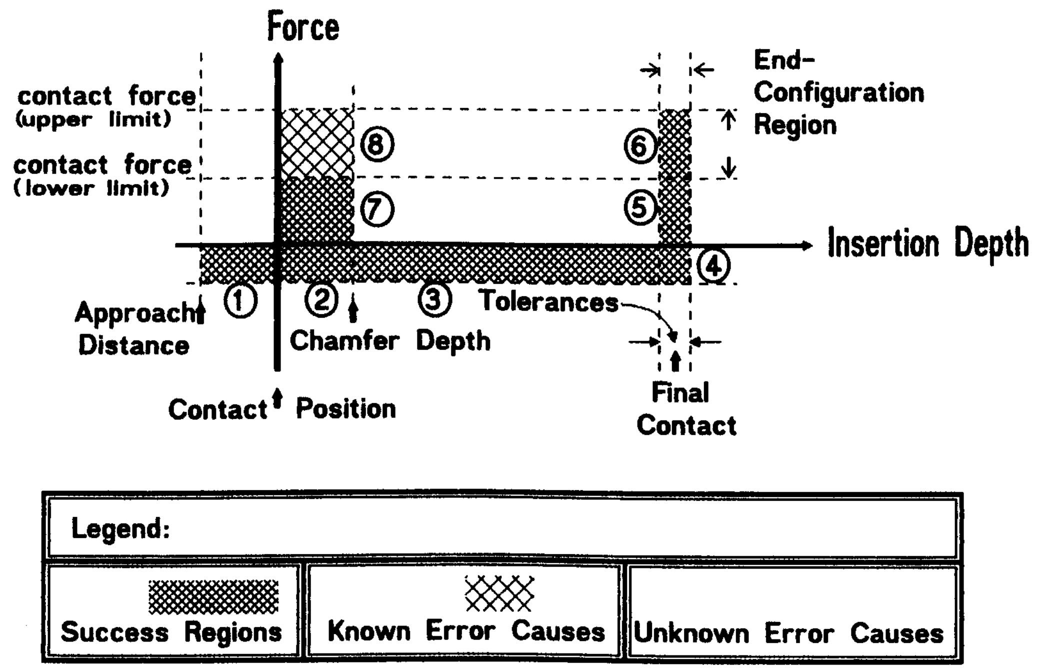


Figure 4.2: Force/Position Space for Peg-In-Hole Example

Figure 4.2 shows an example for the very frequently encountered type of assembly operation: *'peg into hole'*, aided by a chamfer. The relative positioning of the active and passive regions are logged on the horizontal axis using the first contact point as reference. The force experienced in line with the assembly insertion direction during the mating process is plotted along the vertical axis. This example shows a direct correspondence of the sensor measurement with the information desired.

In order to be successful, the positioning has to reach a very specific area partition ⑥ , which corresponds to the desired end-configuration. The progress of the assembly process is charted as the travel along a certain path across the Information Space where specific force ranges are expected during individual position sections.

With only two axes involved, the Cartesian cross products of the sections form *areas*, and in Figure 4.2, those representing the known and desired states are shaded, As long as the travel takes place within these *success* areas (①- ⑥), the state of the assembly is exactly as expected. Region ⑦ is slightly different, in that the forces encountered during the initial alignment contact represent a known condition. If the system compliance can accommodate the resulting lateral motion, then these relatively small values are acceptable and no special recovery action has to be taken. Logically, regions ② and ⑦ could therefore be merged and treated as a single region.

4.3 Inclusion of Error Situations

Should region ⑧ be entered, then clearly the forces are not acceptable, and in this case, a condition has arisen which is not within the pre-specified acceptable sub-space. Treating this sub-space as an error region, the identification mechanism has specific and context dependent information available, which it could use to attempt a search for the actual cause of the error.

4.3.1 Choice of Additional Information

In general, the information associated only with the success of assembly progress is insufficient to recover from the error, although it is quite conceivable that simple conditions are already pin pointed. The data from the shop floor (see chapter 2) suggests that the lack of information at this stage does not constitute a disadvantage within the overall automation process, rather that there is now scope for invoking other sensing mechanisms.

These may not necessarily be physically different sensors, but information ignored previously now assumes importance. Again, force sensing is a typical example. As long as the force along the insertion direction is within the pre-specified limits, other force components are not relevant. If a deviation is detected, then either sideways forces or moments or a combination of all give a more detailed picture about the precise nature of the error. However, since it was not required to monitor that information during the successful progress, time needed to be spent only on the essential information. It can be seen at this point, that the system of information spaces complies with one of the demands derived from the observations of the FLYMO cell (see section 2.5). No effort is wasted on detecting error conditions which do not arise during the significant majority of assembly operations.

4.3.2 Monitoring of Essential Information

Once an error situation has arisen, the emphasis within the control system must change. Although lengthy and time wasting operations are still to be avoided, the desire to get back to the successful state, in which products continue to be manufactured, dominates. So, some effort must be spent in the eventual removal of this problem and it is admissible to use additional sensing operations[35]. These can take even more time than would be permissible in constant monitoring situations, since only a relatively short period will be spent in the recovery routine.

This use of additonal sensing operations does not imply a lengthy delay every time a deviation has been detected. At this stage, the severity of the situation will influence the resources (including time) necessary for the recovery. During the design stage of the cell, the library of generic mating operations will assist in the choice of sensors so that mechanisms available from other parts (for example vision for inspection) can be recruited for the identification process. Clearly, any automated error recovery must, therefore, have some access to the resources available to the cell[16,17], as well as being capable of generating the correct actions. Unfortunately, such topics have to be postponed to further investigations. Work which is suitable for contemplation in this context and which concerns itself with the automatic generation of recovery actions[20,22,25], exists elsewhere.

4.3.3 Handling Assembly Complexity

The point of these discussions is to show the Information Space concept to be able to accommodate unknown error situations, and to provide well defined interfaces to other parts of the systems. These, in turn, are able to address different and equally important aspects of the overall system.

In fact, the example given in figure 4.2 has arisen from the *bush-onto-motorshaft* assembly (see figure 1.1). The error region ⑧ represents the misalignment of the keys, and error recovery is achieved by entering a new Information Space, in which the rotation and vertical forces are characterising the alignment of the keys. The relative positioning establishes its own success sub-space and once encountered, the control system can return to determine where the process has arrived in the original information space.

Naturally, error recovery is designed to return the system from an error sub-space to a success sub-space, but the reality of factory environments does not necessarily ensure this. However, since all information for the successful progress is contained in the processes' Information Space, the new situation can be identified and appropriate actions taken.

4.4 About Cell Control Stability

It is necessary at this point to consider possible consequences of these actions, once error recovery has been attempted. In the first instance, any new state can conceivably return to the error state from which it tried to recover. In that case an infinite loop can arise. This is not too difficult to avoid, since the cell will also retain knowledge about its own past performance. Any error identification can take repetitions into account and draw new conclusions from such a behaviour.

A slightly more difficult problem arises from the situation in which the system may get *stuck* in a particular sub-space. Again, it is conceivable that the actions in error recovery and Information Spaces may not lead to the desired *progress*. No definite solution can be given, although practical implementations could consider time as an additional factor. The lines of thoughts contemplated during this project were directed towards a possibility of *learning* the time during the initialisation phase and then using this value (plus an added margin of safety) for a time-out check.

In general, such features are considered here as being part of the implementation of this structure rather than an essential part of it. One difficulty with Information Spaces could arise from the choice of axes, which currently remains the subject of experience and *good ideas*. However, the provision of adequate libraries would also form part of support for the design of one particular implementation so that there are either sufficient examples or even software tools for automatic or computer aided generation.

The necessity of the latter has already been mentioned, and becomes especially important when the complexity grows beyond the direct visualisation capacity of human beings.

4.5 Multi-Dimensional Hyper Space

Many complications arise in current assembly installations because of the need to specify everything in great detail in order to accomplish a task. As explained in section 3.1.2, the complexity for those products for which the batch size makes automatic assembly economical, is kept to a manageable level by utilising many dedicated stations, each of which is only designed for a limited task. This need for explicit detail will persist throughout any automation system, because of the nature of the underlying control based on computers and their peripherals.

The Information Space concept is no exception. Eventually, any information which is important to the identification of the assembly's success *must* be able to be incorporated. If this results in a high degree of complexity, then it merely reflects the fact that the process itself is complex. The aim of *user-friendly* automation on the factory floor should be the removal of too much implementation detail from the operator, and instead allow for interaction on a conceptual level[34].

4.5.1 Orthogonal Information Relation

With only two essential parameters, as given in the example in section 4.2.3 and Figure 4.2, the resulting relation between force and position is easily visualised as a plane. By making the axes orthogonal, rectangular areas are formed and the Cartesian cross products of the axes' defined segments identify the success sub-spaces.

At this point, the choice of right angles between the axes is more or less arbitrary, except that it

- aids the visualisation process,
- is a convenient method of relating the information sources and forming a space in which the context of run-time data can be clearly established.

4.5.2 N Dimensions

The logical extension of assigning the individual axes to related information sources is the creation of a multi-dimensional, orthogonal hyper space. It allows for any number of important parameters, and despite visualisation being possible only up to three dimensions, there is no practical limit on the implementation of more than three axes.

Figure 4.3 gives an attempt at a three-dimensional representation of the hyper space, in which the general sub-division of information is overlaid as a grid on each of the three planes. Sub-spaces can be identified clearly as hyper volume elements. Above three dimensions, tools for representation become theoretical.

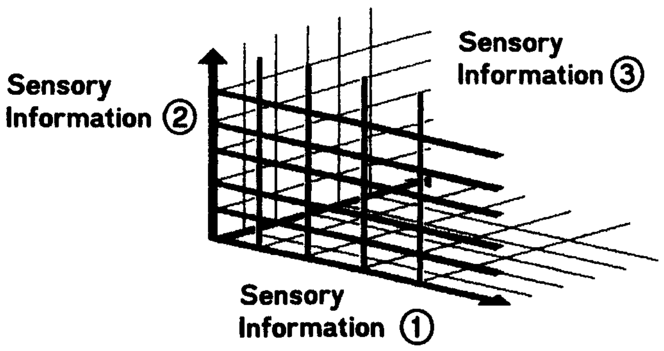


Figure 4.3: Example For A Three-Dimensional Information Space

Fortunately, implementation of hyper volumes is not difficult, since computer programs only have to carry out a number of comparisons (= N) to establish the hyperspace point. N is any integer and its size is determined by the complexity of the generic assembly task.

4.5.3 Success Sub-Spaces

Figure 4.3 also gives a good impression of the potential complexity of trying to specify all error conditions, even if they are related to the process parameters. Within practical systems, the extension is not infinite, but contained by the range of sensor values (see section 4.2.2). These natural limitations and the intrinsic value of concentrating on those parameters which are closely identified with the success of the assembly operation, are illustrated in figure 4.4. Still only three dimensions are used, for obvious ease of explanation.

The individual (hyper-) volume elements form only a small part of the total volume. They are determined by the choice of the process parameters and thus reflect the understanding of the process. For example, a bad choice of axes will not allow the same degree of monitoring as a good choice will.

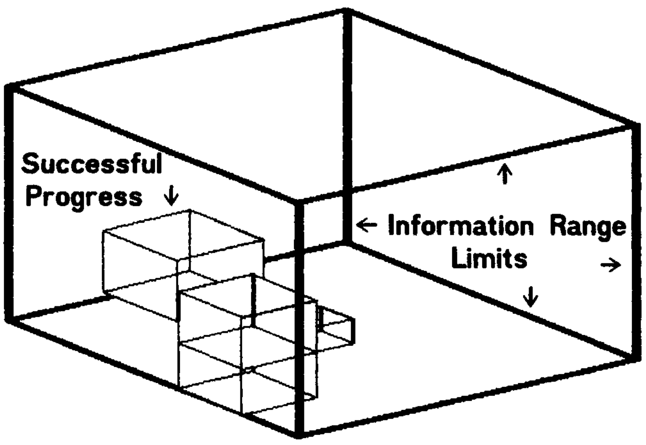


Figure 4.4: Volume Elements Forming Success Regions

Further, Figure 4.3 is slightly misleading, in that the choice of division is by no means limited to even patterns, as the grid implies. The elements are chosen so that they do not overlap, but otherwise there are no limits on the size or relationship to each other. Figure 4.4 shows one example of how the success regions of a hypothetical task may be arranged.

4.6 Assembly Rules

Every sub-space is formed by the corresponding limits of the divisions on the information axes. There are no other restrictions placed on the choice on the divisions, except that they are not allowed to overlap. In general, they are not even required to form continuous spaces by having common boundaries, but as in reality one of the axes will almost certainly be positional information (compare section 3.1), continuity without gaps is imposed by some physical conditions.

4.6.1 Assignment of *Actions* to Sub-Spaces

Concentrating only on the sub-spaces which represent the successful progress has the major advantage of well known relationships between active and passive regions in each individual sub-space. This knowledge is established at the definition stage of the generic mating task by choosing suitable information axes. Consequently, a rule can be formed for each sub-space which contains the action for this given situation. As long as the assembly proceeds as expected, the cell knows exactly what actions it has to execute. In general, these rules are not unique to each sub-space, and despite the fact that different sub-spaces identify specific (and usually unique) situations, progress may well be accomplished by using the same action rule.

Take the force/position plane figure 4.2 as an example again. As long as the situation is found to be in regions ①, ② and ③, the insertion depth needs to be increased. This means that the robot should continue to push down in all cases.

Eventually, all rules which desire to change the current state between active and passive regions will result in a command to a robot, or to some other device which is capable of performing an action which effects a change in the physical state of the cell.

4.6.2 Conditions for the Choice of Sub-Spaces

The reason for the demand for non-overlapping sub-spaces has now become apparent, because it resolves any ambiguities of different rules (in general) for the same situation. This cannot (and should not) be resolved at this level, and generic mating operations must be defined in such a way that there is only one rule for one specific situation. Otherwise, axes are chosen to reflect the process parameters so that the specific mating procedure is adequately (or completely) characterised. In that manner, the assembly of a specific product is synthesised from elements which have their own well defined characteristics, and each generic mating procedure is expressed within the systematic framework of Information Spaces.

The question of sensor choice is as yet unanswered, as the information remains the result of the interpretation of the sensor signals. Planning systems - either automatic or manual - are responsible for the correct matching of the sensors to their ability to provide the desired information.

A library can be built up, using the information axes as criteria to decide whether the assembly operation in question belongs to a particular class or not. Again, the complexity of the task is reflected in the complexity of a particular Information Space. The decisions where to place the boundaries on the divisions need not be taken until the cell also obtains the information about the specific product it has to assemble.

Investigations into manual and automated assembly[27,11] have already suggested some forms of classification systems. Although their criteria were chosen with respect to a different aim, useful information is contained in such work and can be enlisted when the final system is generated. Clearly, features in the geometry of components have a strong influence and must be used as the basis from which the generic sub-spaces can be derived. Finally, the specific product data such as the size of features is also the result from analysing the geometry of the design.

4.6.3 Setting Uncommitted Boundaries

Generic applications are made possible through the uncommitted assignment of boundaries and actions associated with the resulting sub-space. Such generic operations can then be incorporated into a new implementation, without having to reconsider the details of the mating operation. Once the correspondence of the intended operation with an existing generic library entry is established, only the explicit data for the sub-division needs to be provided.

Only after that data is merged with the cell design specifics (see chapter 5), does the interpretation become specific to the cell and the product. Cell specific data contains information about which sensor drivers give the desired information and which actuator drivers can achieve specified goals. In programming terms, the stage reached at this point resembles very closely the stage at which a VAL program (or any other commercial robot language) can start execution.

4.6.4 Run Time Consequences

As far as error recovery is concerned, there is, however, one effective difference from robot oriented languages. The whole operation is guided by information gained from its environment and error detection is enabled automatically because any deviation from these very specific success sub-spaces is interpreted as an error. The conventional way of pre-programming recovery routines for specific situations is not excluded from this framework, since those routines dealing with obvious error situations can be entered before the invocation of the data. These error situations are identified by designers as their own sub-spaces.

Treating the detection of unknown error conditions as the entry into sub-spaces which contain no knowledge about the process, provides a platform for learning how to recover from the situation encountered. Should the implementation of FTM pairs not include an automatic error diagnostic facility and recovery strategy generator, then it is still feasible to enter new routines *manually*. In this case, diagnostics and recovery is performed by the operator.

This contradicts the desire for fully automated systems only at a superficial level, because of two main factors; both of them are very important in practical implementations:

- New errors do not appear randomly (see section 2.4.3), but tend to repeat themselves over an unspecifiable period of time. Giving a cell the capability to detect and react to certain conditions, even if only through an *on-line extension of pre-programmed routines*, then the operator's assistance will be required less as the experience of the cell grows.

- Specific error conditions can only be recovered if the cell has the ability to discover them or at the very least is able to deduce the reasons for the encounter of the new situation. Consequently, a natural compromise arises between the provision of sensing mechanisms, supporting computing power and the cost of employing operators. It is unrealistic to expect to recover correctly from all error conditions automatically, and in the final instance, some assistance will always be required from an operator.

4.6.5 Robustness

So far, the implicit assumption has been made that any deviation from expected sub-spaces represents an execution error. Viewed from a different angle, it is attributable to some environmental condition and the process description itself is not at fault. Whereas such a state is certainly desirable, no guarantees can be given that this is true.

Information Spaces allow for those process modelling errors by their uncommitted nature (see section 4.6.3). Not only is a generic approach enabled through them, it is also possible to change specific parameters on line. For example, a typical situation for the above to happen could be encountered when the operator is not able to discover any *real* errors. In this case, he might decide that the boundary which was crossed, could be extended by the newly measured amount because it is really part of the success sub-space. The level of trust in the system implementation and any Artificial Intelligence techniques used therein will set the eventual autonomy in the decision making process.

It is easy to see that extensions of boundaries should be treated with due respect, and safeguards have to be built in. On the other hand, reductions of sub-spaces could be left much more to the system, particularly if a large number of assembly actions have already been executed and part of the sub-space in question has never been entered.

Any implementations of FTM descriptions can address themselves also to the real possibility of system learning. Not only are some parameters changed to achieve automatic improvements, but there now exist criteria against which the performance of individual cells can be measured. If the data about the sequences of sub-spaces were to be kept for later analysis, production management may well decide that the invocation of one particular error routine is better dealt with by some previous sensing in the process, or even by providing a more specialised jigging which reduces the errors.

Additionally, if there are no errors during distinct divisions of sub-spaces, then the equipment which might have been provided especially for additional information gathering during anticipated errors could be used in other cells. Furthermore, as the confidence in operations increases, many sub-spaces can also be timed and new conclusions drawn from this new information. For example, the robot's speed can be increased in one sub-space, or has to be reduced to avoid errors during another.

4.7 Summary

This chapter has explained the concept of Information Spaces and discussed how it can be used to form a useful framework in which sensor data can be merged and interpreted in its correct context. It encompasses the complete description of any assembly process but only requires actions for those sub-spaces which represent the successful progress. Contrary to error situations, 'normal' circumstances are known already at the design time and can thus be specified without major difficulties. Any errors are detected at run time by establishing the deviation from the 'norm'.

The knowledge about the performance of a cell within circumstances which are often unique to the installation, allows a high degree of robustness with respect to errors at run time such as changes in the environment. Furthermore, the framework is also robust with respect to errors in the specification of information sub-spaces, as these will manifest themselves as specific error conditions at run time. The ability to change the assembly behaviour allows the cell to learn to cope with unforeseen situations and thus the most common error situations are eliminated very quickly.

Interfaces to error identification and recovery strategy generation are provided so that some automation of these two important aspects can be linked with this structure. The operator's expertise is also included, because it can be entered as manually determined recovery actions.

The existence of relatively large amounts of data such as past performance and instant data, enables a more detailed investigation of the actual situations encountered. The structure of the data representation therefore plays a vital role in this examination process. Chapter 5 will discuss the principal and practical implications of the proposed data organisation and how the FTM description with active and passive regions is consistent in its treatment of the assembly process as an interpretation of sensory information.

Chapter 5

Data Structures

The *FTM Pair* description of assembly processes has established a systematic framework which contains the identification of all significant stages by representing them as logical sub-divisions with well defined interfaces. Using this approach, areas of responsibilities have crystallised for each sub-division, and the necessary parameters which determine the accomplishment of the assembly task, are defined. The most critical stage is found in the mating of the *regions of importance*, where the *Information Space* description provides a suitable system to determine the outcome of assembly.

This method has broken down the potential complexity, but since the process itself can remain very intricate, the overall complexity cannot be reduced; particularly, if the system must be constructed so that it can cater for a general solution. Consequently, the actions which are to be carried out, still have to be specified in sufficient detail, and the availability of a systematic approach opens the way for automatic generation of most details. Although a large part of the final implementation of individual steps would thus be hidden from the operator, the necessary amount of detail, whatever size, still needs to be generated for successful run time control. In order to be manageable, this data must be organised so that the basic ideas of the framework are represented. Any implementations can use the data definitions in suitable algorithms. These algorithms have to be written to suit the data and need not be adjusted for different tasks. Once both ingredients for the cell's control software are established separately, the assembly of another product will then involve only the loading of new data, rather than rewriting programs (see Figure 5.1).

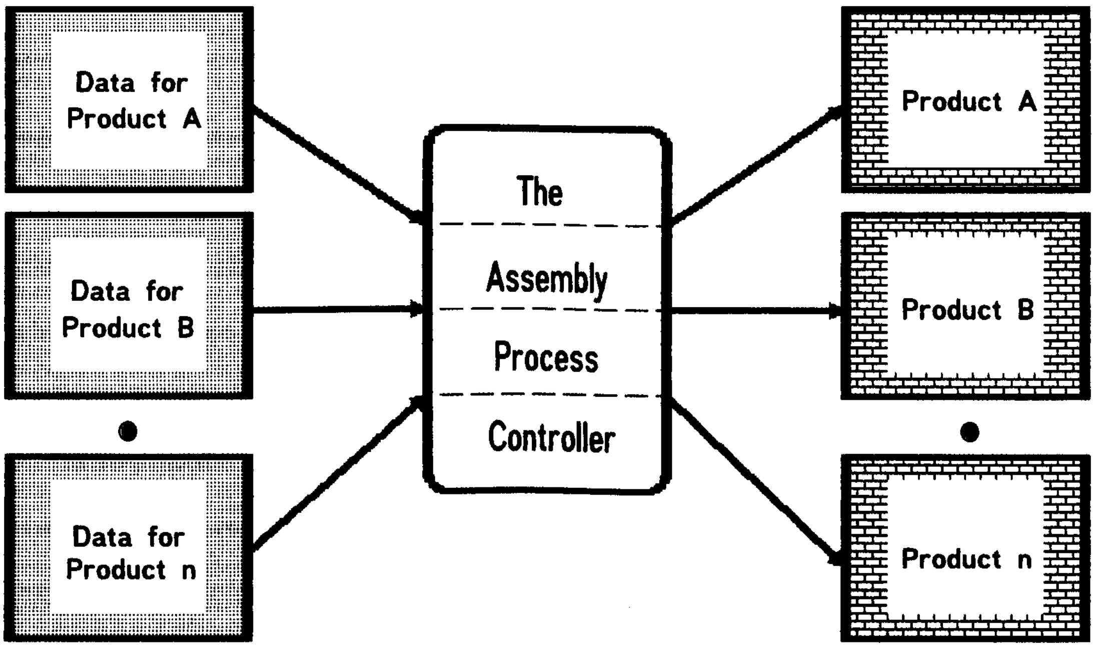


Figure 5.1: Data-Driven Assembly Control

This chapter explains the significance of data organisation for assembly and shows how individual sub-processes in FTM are reflected in data structures rather than in procedures. The detailed representation of information needed at run time - that is for a very specific, setup-dependent control system - is given and then the separation into generic and product specific data is explained. In the context of information and its data representation, the importance of *Local Accuracy* is highlighted. Its role is linked strongly with a TOP-DOWN design approach, where the product or cell designers are free to concentrate on the conceptual aspects of assembly. The modularity of generic assembly also encourages BOTTOM-UP implementation because each part in the assembly process can be addressed without fear of consequences elsewhere in the cell. Implications towards the general teaching or programming of the cell are also explained.

5.1 Information for Cell Teaching

Before any product can be assembled, the cell controller must be informed about the tasks it has to execute. In conventional systems, the robot is programmed to move to certain locations, which it was shown before and which were recorded in its memory (this is the so called Teach-In procedure[36]). More sophisticated programs can be entered textually (VAL, VALII AL etc), thereby including more run time checking and also some sensor interaction. This method is only a relatively small advance over the 'show and replay' type, since all variations have to be included at programming time and the task execution remains essentially that of a replay action robot, albeit with some variations.

5.1.1 On- and Off-Line Programming

Despite the substantial effort required to re-program manually, this method has the considerable advantage that locations are adjusted implicitly to suit those actually encountered in the cell. On-line programming is thus inherently more precise at run time compared to off-line programming. Nevertheless, the advantages gained by increasing the flexibility through off-line programming tend to outweigh the drawbacks in the long term. The main problem arises from a potential mismatch between the model of the environment contained in the computer and the real surroundings. On the other hand in off-line programming, the generation of new programs can take place while the machinery is still productive, whereas the teach-in method must halt the production flow.

In general, the efforts involved in off-line programming to keep the robot's actions to the desired accuracy are significant. Therefore, it is only common sense to provide a facility to combine the advantages of both systems and choose the most suitable one where appropriate. In this work, teaching will be regarded as the act of *providing all assembly information,* regardless of the source or method.

In the FTM Pair description, these two convenient ways remain the main (and only likely) methods by which the robot assembly cell will be taught about the task. Both involve the teaching of only one mating position, where the final end-configuration is recorded:

- If programming is done manually, then the robot is guided in Teach-In procedures to the point at which the generic mating procedure would eventually terminate, representing the successful end-configuration. Any other configurations during the mating are temporary and already known from generic routines, removing the need to specify many intermediate positions explicitly.

- If the system is programmed off-line, then the *geometric* relationship between the active and passive regions determines the final robot position. This part of assembly has attracted considerable research throughout the world[31]. In the structure proposed here, the assumption can be imposed that the final specification of active and passive regions and their relationship has to be obtained from design and simulation procedures.

The outcome from these arguments is the fact that there will be a *single* robot location which is of importance. This location contains position and orientation and can of course be made up of several transformations. Although the global location needs to be known at the final instant, this global knowledge is not relevant to the assembly process.

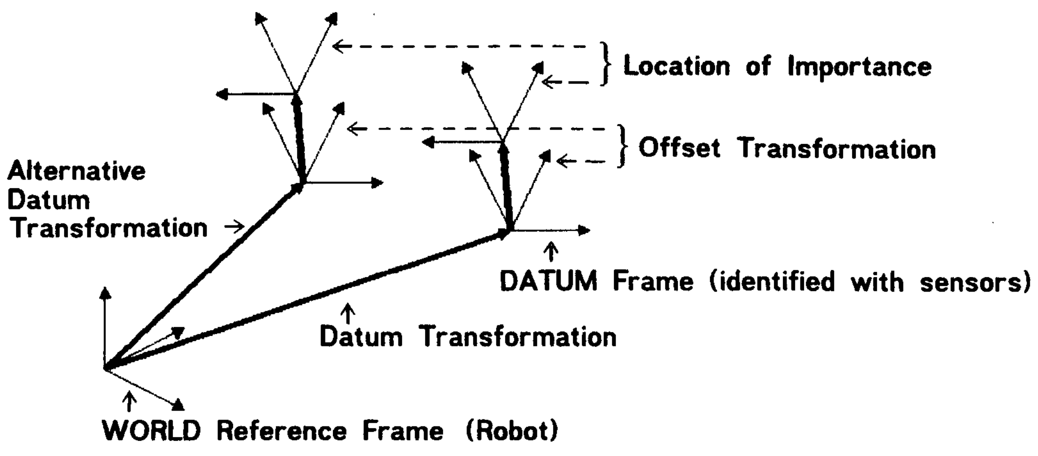

Location of Importance
Offset Transformation
Alternative Datum Transformation
DATUM Frame (identified with sensors)
Datum Transformation
WORLD Reference Frame (Robot)

Figure 5.2: Independence of Global Accuracy

5.1.2 Significance of *Local* Accuracy

The off-line programming disadvantages arise mainly from the inability of mass production machines to maintain a precisely known environment. It would be uneconomical to produce robots to a very high precision requirement. Currently available robots with a claimed repeatability of 0.05mm tend to be more than twice as expensive as those with 0.1mm repeatability. Although these expensive robots usually also claim a relatively good accuracy of less than 0.5mm, other factors probably have more a significant influence upon the ability to find the important locations. Examples of the latter may be found during the reconfiguration of cells for different products; when jigs and feeders may not be put back exactly at the same position, (compare Appendix H - datum frame calibrations).

The discussions in section 3.1 have led to the conclusion that global knowledge about the assembly environment is not necessary, only the relative positioning (compare Figure 3.1) of the components is important. Consequently, where within the reach of the robot the operation takes place has no influence on the mating procedure itself (see Figure 5.2).

Composition of the end-configuration into more than one transformation can reduce errors substantially. There is no need for global accuracy of any of those transformations, except one, a datum frame. Offsets become very small (compared to the working envelope of the robot) and subsequently any errors will be correspondingly small. Again, the decomposition of locations provides advantages over both teaching methods. Mismatches in the computer model are relative to that datum frame and remain small, whereas teach-in methods require the 'showing' of the offset only once.

Generally, the manipulator's repeatability over a small distance is sufficient for an attempt at mating even with less precise robots. This is particularly relevant when the 'capture' range of sensors employed in identifying the DATUM frame is significantly larger than any likely errors. Therefore, it should be possible for the robot itself to establish all important locations. Otherwise, warnings can be issued at run time to make the operator aware of the calibration failure.

5.1.3 Self-Calibration

Of course, datum frames still have to be located globally, but sensors already needed elsewhere in the assembly - such as vision systems[64] or cheap sentinel probes[10] - can be drawn in for this calibration. It is also desirable to employ a method common to all locations, as this will reduce the overall set-up complexity and make the cell more easily operated by inexperienced personnel. The precise locating of datum frames with the aid of suitable sensing mechanisms can take place during system initialisation, and it is reasonable to assume that no further changes in positions occur during *normal* operation. Nevertheless, should the error diagnosis suspect a misplaced component, the calibration routine can be repeated.

Even the points at which calibration of datum frames takes place need to be known only roughly[4], thus the time spent on teaching by the operator is reduced. When programmed off-line, quite large errors can be tolerated, even to an extent which would lead to unacceptable stoppages in sensorless systems. Furthermore, the potential for tuning the computer's model exists through sensory re-calibration, since the newly acquired data can be fed back into the model. Obviously, such an ability for self-calibration is particularly important for rapidly changing batches, where batches of the same product may be assembled by the cell repeatedly, but not continuously.

[4] as long as the 'capture' range is not exceeded

For example, datum frames and offsets are easily taught in VAL II. Assuming that the frame is already defined as FRAME, then the offset is defined simply by entering

```
HERE FRAME:OFFSET
```

after the active and passive regions have been placed into the correct end-configuration.

Assembly sequences which are to arrive at this configuration must at some point be derived from the geometric features of mating regions. However, by interpreting features which are relevant to the assembly process as information, implications towards the structure of the descriptive data have to be observed. The emphasis is placed on process information and not geometry. The following paragraphs will show that even the regions of importance (compare section 3.3.3) are much more suited to the FTM method when viewed as data rather than as geometric features which has been conventionally the case.

5.2 Active and Passive Regions as Information Structures

In Chapter 3, the introduction of *regions of importance* departed from the approach of geometric reasoning about each component. In fact, any component can be regarded as a collection of active and passive regions. Naturally, these are derivatives from the geometry, and planning stages must take shapes and sizes of components into account to produce the information necessary for these regions.

5.2.1 Information for Passive Regions

Once the feeding stage has been informed about the passive region which is to be fed using a suitable reference, then the feeding procedure can associate a component with that region (defined previously in the cell design stage). This leads to a further step which identifies the component with a physical feeding mechanism. This mechanism is activated from the feeder driver, which provides the physical interface between the desired information and the real world.

For example, there will be many instances where a binary signal will have to be set in order to switch on a bowlfeeder, until another sensor (proximity, light barrier or similar) provides the signal to switch off again. On the other hand, large uncertainties in the feeding[1,4] may need sophisticated sensing, and even some manipulation, if the most suitable sensor is mounted on the robot.

The driver's tasks are the association of the passive region with the necessary physical events and the provision of information wanted by the control system. This includes both the DATUM frame and the OFFSET, since both are in general variable. The role of the DATUM frame has previously been explained, and the variations in the OFFSET can be seen in the following examples:

Bowlfeeders are usually associated with only a single location, from which components can be picked up. However, if the components are presented on a stack or in a tray, then the location will change each time a component has been picked up. Keeping track of the sequences can be made *entirely* the responsibility of the feeding stage, preserving consistency with the FTM stages of this description. Also, the same component may be chosen to be fed by different methods (for reasons which cannot be foreseen at the time of the cell design), leaving a clearly defined responsibility for the driver. Finally, the feeding stage will result in that *single* location (compare section 5.1.1) which can be used later on. The information for passive regions is summarised in figure 5.3.

Passive Region Information	
Identifier	For reference and association with components
Datum Frame	To be returned by driver
Offset Transformation	To be returned by driver
Driver	Physical interface

Figure 5.3: Information for Passive Regions

The presence of the component at the feeding point is contained implicitly in the positional information. The method of communicating presence or absence is implementation dependent and not necessary for fundamental considerations. Hence, it is possible to provide a suitable error message as part of a larger error handling scheme, use an additional Boolean variable, or adopt a convention such as a NULL DATUM location. The latter makes the assumption that such values will never occur in practice.

5.2.2 Information for Active Regions

Once the feeding of a passive regions has been successfully completed, the information can be used in the transport phase to guide the active region to the passive region. So far, the assumption has been made that the OFFSET defines the end-configuration, but without due attention to intermediate configurations, a simple 'MOVE' command will almost certainly lead to collision situations. However, if the transport phase has access to the Information Space data about the Mating, then it is able to determine a suitable approach transformation automatically.

This approach frame does not have to be calculated explicitly for every transport phase, but is initialised at the same time as DATUM frames are defined. This would indeed be more sensible, since the likelihood of continuously changing configurations is very small. Nevertheless, automating the generation of this data further reduces the details which have to be provided by the programmer. It should be born in mind that explicit teaching is not excluded.

Of course, access is needed by the run time controller to the result of the feeding operation. The transport phase is terminated at the location resulting from the combination of approach distances and the location of the passive region. The approach problem can be made part of the general obstacle avoidance task, but it is special in this context as motions take place very close to mating. Therefore, these sections in the geometric configuration have been defined to be part of the mating procedure. The remaining part of generating the path from the start of the transport phase to the passive region must also be provided.

In the simplest case, a single 'MOVE' instruction can be issued to the robot, but very sophisticated routines may be employed here[9]. The degree of automation will depend on the overall automation and planning of the cell. In the case of teach-in procedures, 'VIA' points[36] represent a very convenient compromise.

Finally, a driver routine is necessary to provide the ability to interface physical devices which will perform the transport. In multiple robot cells, an identification of the robot itself could be one of the parameters given to the driver. The advantages of retaining a driver in the transport phase - as in all other phases - are found in the independence of implementation, and in the independence of robots themselves. Sophisticated high-level languages (such as VAL II) as well as simple point to point motion robots are acceptable. Figure 5.4 summarises the information for passive regions.

Active Region Information	
Identifier	For reference and association with components
Passive Region	To extract the target position
Generic Mating	To extract approach transformation
Path Generation	To avoid collisions
Driver	To provide the physical interface

Figure 5.4: Information for Active Regions

So far in this chapter, the FTM methodology has established the basic requirements for data organisation. The next sections will address more specific problem of data management at run time, where all data - generic and specific - has to be available to perform the task.

5.3 Run Time Data Organisation

One of the advantages contained in the FTM method has been highlighted as the systematic approach to error recovery[3,12,20]. Although such procedures play a very important role, particularly at run time, the system is directed towards error detection and not towards identification or recovery synthesis[21,22,23]. Hence the structures proposed here will identify the necessity for a link into error recovery, but will reserve their detailed implementation for future work. It is important that data for the detection of the successful progress is represented so that any error recovery routines can make use of the available information.

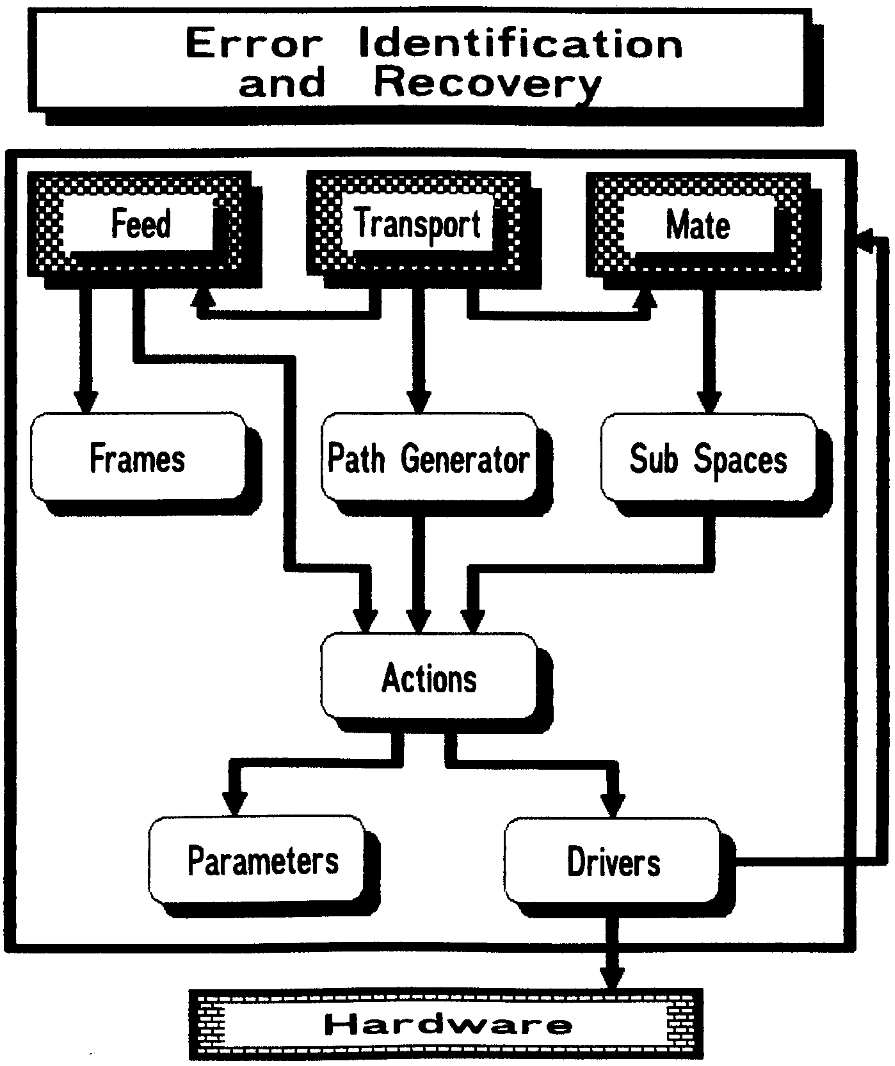
5.3.1 House-Keeping Information

Any practical cell controller will almost certainly need access to implementation specific information which can generally be regarded as 'house-keeping'. For example, the initialisation of the assembly cell must know, somehow, whether Frames and Offsets have been defined. This can be determined by interrogating the 'house-keeping' data associated with each passive region. The same can be said for the Drivers, ensuring, for example, that the desired routine is actually loaded.

House-keeping is a matter of implementation and - to a degree - apparently arbitrary decisions which have to be taken in order to define the most convenient structures for given facilities. The final form depends on the cell controller and its sophistication, but not on the principal specification of assembly information. House-keeping will be given only minimal attention within the context of this thesis.

5.3.2 Principal Data Support Structures

Although house-keeping data is necessary for implementing the FTM structure, it is not part thereof. However, there are some details which must be discussed and which represent the information - itself repeatedly emphasised as the essential aspect of sensory assembly. Figure 5.5 attempts to give a general overview of the most important relationships between individual structures.



where arrows mean: *has access to*

Figure 5.5: Principal Data Relationship

From previous explanations, Error Identification and Recovery is not directly connected with the remainder of the data. Obviously, it is not completely detached either, hence the close proximity. The precise nature of the links has yet to be established and due to its own complexity forms part of a further research topic. The next sections describe the links between the fundamental data structures.

The arrows in figure 5.5 signify access to and influence over data. Hence the Feed phase must know about and update the Datum and Offset Frames. Through feeding, the Transport phase is able to make use of the generated information. Additionally, the Mate phase also influences the transport by virtue of its approach configuration. Feed and Transport will both influence the physical state of the cell and must also have access to the information gathering devices (sensors). Collecting information and changing the cell's state is executed directly through the Actions, except in the case of Transport, where the influence is exerted only through an additional layer, the Path Generator.

Similarly, Mate changes and observes the cell, but also has to realise the rules contained in the Sub-Spaces. Actions themselves can interface to Drivers by calling appropriate routines with action and context specific Parameters. It is eventually up to the Driver to provide the final details of interfacing to the real world, leaving the choice of hardware such as the complexity of interconnections to a preferred implementation (for example, see chapter 6 and Appendix A for demonstration cell details).

5.3.3 Recursive Nature of DRIVERS

Drivers are special in one particular sense. The sequencing of assembly operations cannot always be described within one single Information Space, but may well demand the invocation of other Spaces during one assembly situation. An example of this can be found in the relative orientation between the bush and motor shaft. Initially, the encounter of a high level of force whilst still in the chamfer region can be interpreted as an error and dealt with in the conventional way.

Equally, the regularity of this error occurring allows an interpretation of the rotation adjustment as being part of a more complex assembly situation. Consequently, the encounter of this particular situation leads to a change to another Information Space in order to monitor and affect other parameters, including rotation, torque and in-line forces. When the keys are finally aligned, that is, when the secondary Information Space has reached its successful sub-space, the original Information Space can be returned to and the new situation be assessed. By giving Drivers the ability to call the whole system recursively, a large scope is enabled for complex as well as simple sensory assembly.

5.3.4 Information Representation

Having established the principal relationship between data parts, the descriptions of individual elements of the information representation follow. All explanations refer to *types* of data, and not to actual information. Hence the need for some information from other data structures does not indicate access to the same physical information, but it may be used if circumstances demand it. Context integrity is preserved by storing various information in different places and consequently the amount of storage is not reflected in this presentation.

● **Transformations:**

These represent the basic robot locations and offsets. They are exactly the same concept as the familiar transformations used throughout robot kinematics and dynamics[26,36]. The choice of Roll, Pitch and Yaw is essentially arbitrary, but is widely accepted by robot manufacturers and researchers. Strictly, an Identifier is not needed but, firstly, some commercial robot control systems allow reference to locations by name and, secondly, it is easier for human interaction to refer to names rather than values. The latter is true for all data, but for the sake of completeness and ease of manipulation, the actual values are also contained (see figure 5.6).

Transformations	
Identifier	For reference purposes
X Y Z	Numerical values of origin
Roll Pitch Yaw	Numerical values of orientation

Figure 5.6: DATA - Transformations

● **Datum Frames:**

Datum Frames contain the information which make up a datum frame. At least three points are necessary[73]:

- the origin,
- a point which defines the x-axis together with the origin,
- a point lying on a line parallel to the y-axis, defining the x-y plane.

Once the x-y plane is defined, the z-axis can be constructed from that data and a new transformation results from the calculations. The addition of an identifier completes the information for datum reference frames and the format is shown in figure 5.7.

Datum Frames		
	Identifier	For reference
Transformation ⇨	Origin	Calibrated transform. defining origin
Transformation ⇨	X-Axis	Calibrated transform. defining x-axis
Transformation ⇨	Y-Axis	Calibrated transform. defining x-y plane
	Resultant	Calculated transform. from points above

Figure 5.7: DATA - Datum Frames

A new feature is exploited within the data. Sometimes, it is advantageous to use locations which are physically the same. For that reason, Datum Frames do not actually contain the locations from which they are defined, but only refers to them. That way, each transformation can be defined individually. Should locations have to be physically identical points, then consistency is preserved. The symbol given to indicate a reference, rather than the values, is "⇨". The type of data referenced is given in small print.

● **Drivers:**

Drivers (figure 5.8) are hardware and implementation specific routines, which are called whenever the conceptual behaviour has to be executed as physical actions. In fact, the role of Drivers is rather more complex than a simple translation from conceptual demands to physical parameters (compare section 5.3.3). An indication is given in figure 5.5 that they have access to the complete FTM structure. In other words, a recursive call of FTM stages should be possible.

Drivers	
Identifier	For reference
Routine	The procedure to be called

Figure 5.8: DATA - Drivers

Although this could create a complex structure and potentially includes the danger of an unmanageable system, it merely reflects the complexities inherent in the assembly process itself. The general structure of FTM and generic mating procedures is not diminished but enhanced through this facility. An example may clarify this.

Consider the coordination of multiple robots within one cell, especially for a 'two-handed' operation. If one of the parts has to be secured by a robot, then the success of this process can be determined by regarding the 'fetching' of the first part as feeding the passive region. The first robot thus has to execute an FTM Pair:

- Pick up the first component,
- 'Mate' active region to the 'empty space' at which the second component needs to be added.

Now the active region changes its role to a passive region which has just been fed. The second robot, too, has to execute an FTM Pair so that the mating of both components can be carried out:

- Pick up the second component,
- Mate active region to the designated passive region, still attached to the first robot in the normal way.

The assembly process has been successful if the first mating (=feeding) and the second mating have been accomplished successfully.

The task of invoking the right FTM Pair falls upon the Driver. The obvious consequences for the cell controller concerning timing, coordination and concurrence are very important in these circumstances, but their detailed investigations are beyond the scope of this thesis. The example was chosen to highlight again the complex nature of Drivers. Drivers also fulfil more conventional functions such as combining, filtering and weighing sensor data. Eventually, the complexity is determined by the implementation, and it has to be mentioned that the experiences at the site of the industrial partner have not favoured a simple, rather than a complex approach (see Chapter 7 and section 8.1.1).

Parameter List	
Type	The parameter type
Parameter	The actual parameter
Parameter List → Next	Next parameter

Figure 5.9: DATA - Parameter List

● **Parameter List**

Parameters for Drivers are provided by a Parameter List (Figure 5.9). The type of parameter is specified so that some cross checking within the driver routine can take place as to the correctness of the parameter passed. The data structure of type 'list' has the advantage that it can be extended to suit the requirements.

Path Generator	
Identifier	For reference
Path Type	The type of path generated
Routine	The routine used to do it
Parameter List → Parameters	Necessary parameters

Figure 5.10: DATA - Path Generator

● **Path Generator:**

This information has a similar form to the Drivers. It also contains the type of path generated and must have its own parameters to be successful. The most important one is the final location of where the robot has to end up, whereas the start is obtained from the current robot location. The inclusion of this routine can remove the explicit teaching of motions specific to the cell design.

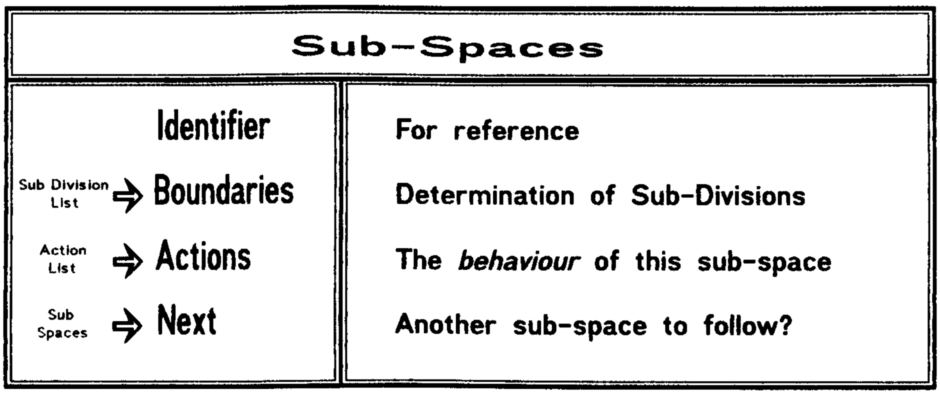


Figure 5.11: DATA - Sub-Spaces

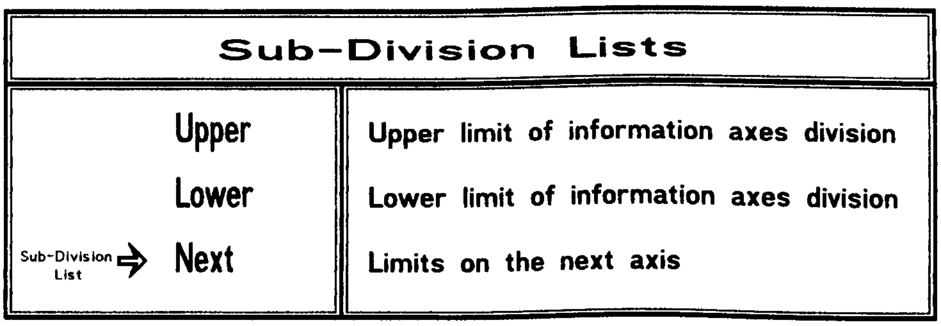
● **Sub-Spaces:**

Within the description of the Information Space, each sub-division must be clearly identified by limiting values on information axes. An individual sub-space is associated with a generic sub-space. It is characterised by the number of axes and the 'sensory' actions to be performed when the system searches for a situation to be identified with a particular sub space. Since there is a - generally - unknown number of axes and actions, the data within the sub-spaces represents the start of lists for both boundaries and actions (see figure 5.11). In fact, the organisation of the sub-spaces themselves is best done in a list, since their number is also not fixed in the general case, hence the pointer to itself.

● **Sub-Division Lists:**

The actual data contained in the sub-division list consists of the upper and lower boundaries of an information axis. The correspondence of these 'numbers' must be ensured at the time of initialisation and should not be a time consuming procedure at run time. For example, any measurement is only of value if the units are corresponding. In fact, there must be as many entries in the sub-division list as there are axes in this Information Space. An initial check can verify the correspondence of run time data relatively easily.

The complete description of all situations within the Information Space must be able to represent the conceptual $\pm\infty$. This does not pose any practical problems, since the range of values available in floating point representation usually exceeds those encountered in real applications. A *very large* value can thus be assigned to be ∞, such as 10^{10} or anything else suitable.



Sub–Division Lists		
	Upper	Upper limit of information axes division
	Lower	Lower limit of information axes division
Sub-Division List ⇨	Next	Limits on the next axis

Figure 5.12: DATA - Information Space Sub-Divisions

● **Action Lists:**

The role of the action list - there may well be more than one action - is the association of particular Driver routines with their parameters. Thus the action has to point to a driver and also to a parameter list. Again, the pointer to itself enables the construction of a list. In this way, driver routines can be called with many different parameters, especially with different functions which some drivers have to perform.

Action Lists	
Driver ⇒ Action Driver	Routine to invoke
Parameter List ⇒ Parameters	Parameters to pass to the routine
Action List ⇒ Next	A potential sequence of actions

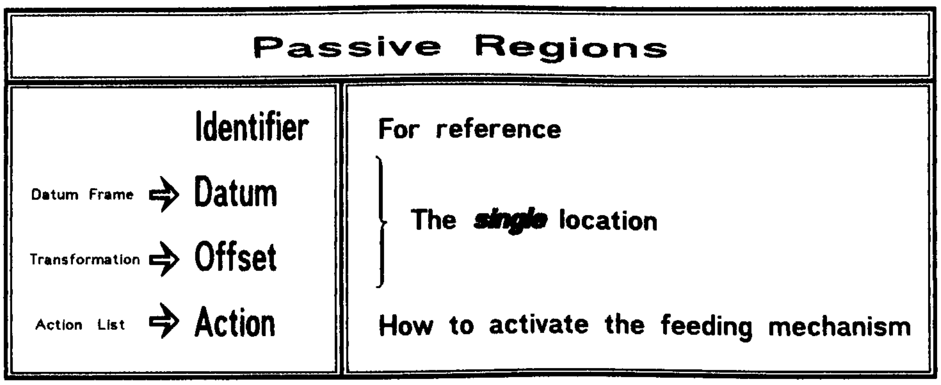
Figure 5.13: DATA - Action Lists

For example, robots and proportional grippers are not merely motion devices, they also act as a coordinate measurement devices and drivers have to initiate moves as well as interrogate these physical systems.

Passing of parameters must thus be bidirectional. In addition to providing certain commands and parameters for devices, results may be returned to the calling procedure. Handling sensory information is a matter for individual implementations, for example, the temporary storage of numbers for further processing. Although any comparisons are at the very heart of the decision making process and thus the interpretation of the data, the mechanism itself is outside the specification of successful assembly.

5.3.5 Top Layer Information

Section 5.3.4 has described the supporting data structures for the FTM method. These are now ready to be used in the top layer (compare figure 5.5) in the form of the Passive and Active Regions. Their implementation is very similar to the basic descriptions in section 5.2. The Mating operations differ from the previously discussed form only in that *all* information, generic and specific, has to be represented.

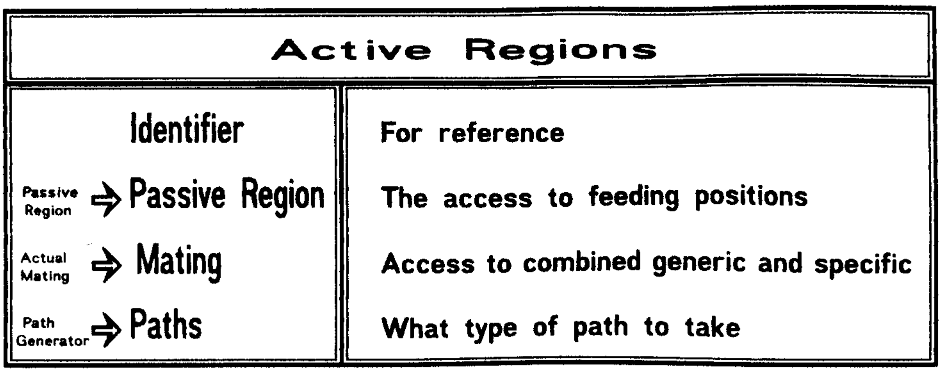


Passive Regions	
Identifier	For reference
Datum Frame ⇨ Datum	The single location
Transformation ⇨ Offset	
Action List ⇨ Action	How to activate the feeding mechanism

Figure 5.14: DATA - Passive Regions

● **Passive Region:**

The data structure for Passive Region is shown in figure 5.14 and the similarity to figure 5.3 is quite obvious.



Active Regions	
Identifier	For reference
Passive Region ⇨ Passive Region	The access to feeding positions
Actual Mating ⇨ Mating	Access to combined generic and specific
Path Generator ⇨ Paths	What type of path to take

Figure 5.15: DATA - Active Regions

● **Active Region:**

The similarity to principal considerations is the same for the Active Region, but, of course, figure 5.15 must be compared with figure 5.4.

● **Actual Mating:**

The execution of an assembly task needs both the generic data and the product specific information. These are combined into the single data structure of Actual Mating. It, too, has an identifier and a pointer which associates the known and successful sub-spaces with this mating operation. The actions assigned to each sub-space are contained in the sub-space data, but the manner of generation of sensor data has not been identified. The Information Axes point to a list of sensory actions which are executed every time an updated set of sensor values is demanded from the cell.

The success of the assembly is indicated by the arrival of the mating operation in one special sub-space, the success region. A separate pointer to this sub-space preserves the independence of the success region and can be assigned to any sub-space - as appropriate. Decisions about *Success* or *Failure* are also made slightly easier after the controller has determined that the current process happens to be in a known sub-space. Finally, the approach transformation itself is contained in the Actual Mating data, because it can be assumed that it will not change after it has been calculated at the beginning. The transport phase can thus simply *look it up*.

Although the mating success is given through the arrival in the end-configuration sub space, the procedure has not finished at this stage. Appropriate actions of clearing the manipulator from the mated components must be incorporated. Termination is identified by entering the Final sub space, and thus the controller can check whether the parts have been configured and whether the process itself has progressed as expected. Figure 5.16 summarises the Actual Mating data.

Actual Mating		
	Identifier	For reference
Sub Space	⇨ Sub-Spaces	Known Sub-Spaces in this Mating
Action List	⇨ Axes	Information Axes
Sub Space	⇨ Success	End-Configuration Sub-Space
Sub Space	⇨ Final	Termination of mating
Transformation	⇨ Approach	Clearance before mating begins

Figure 5.16: DATA - Actual Mating

The collation of data for the Actual Mating concludes the definitions for the supporting data structures so that all information for a complete control of the process at run time is assembled. It has to be stressed again that errors and their determination are not part of and are not contained in these structures. It is envisaged that similar arguments can be brought forward for the generation of error sub-spaces in an equivalent manner, but the search and organisation of error situations[18,19,24,25] can be regarded as being separate.

However, the merging of generic assembly process data and product specific information for the Actual Mating has temporarily obscured the importance of their separation which will be the subject of the next sections.

5.4 Generic and Product Specific Data

The development of the Information Spaces (see chapter 4) forms the basis from which a definition of generic mating processes can be generated. Different processes can be judged and compared against existing ones and classified into different groups. Three criteria have to be observed during classification:

1. Equality of information axes
2. Equivalence of sub spaces
3. Equivalence of success actions

In the following, the generic aspects of the mating phase in assembly are explained by discussing the case for membership in a generic process. These criteria can then be used for the evaluation of any new process with the view to its being assigned to an existing class or to establish a new one.

5.4.1 Equality of Information Axes

In section 4.2, the Information Space description of a mating process was defined through the identification of that special information which describes the success of the mating procedure. This is irrespective of any methods of sensing such information. Consequently, two processes which are the same are also described by the same information requiring an equal number of axes. If there are fewer, then some information is missing and the monitoring is incomplete.

On the other hand, if there are more axes than necessary, additional data has to be processed, and the description of the mating process is over-complicated. This situation is also undesirable, since extra sensing systems are likely to be required to provide the extended context for decision making. Additionally, any automated assembly cell is already complex enough from a user's point of view, and any simplifications help the user friendliness.

Overall, the complexity of the assembly operation is reflected in the number of information axes. Hence for a very simply case, a single axis may suffice, whereas a complicated procedure may need many axes in order to be described adequately. However, the latter point is addressed in the wider issue of Design for Assembly[27,90,91], in which complications (and subsequent expensive monitoring systems) can be avoided to a significant degree.

5.4.2 Equivalence of Sub Spaces

Having established the equality of the Information Axes, the equivalence of the success sub spaces must be determined. They do not have to be exactly the same; indeed, the advantage of generic classification lies in the equal treatment of processes which are *nearly* the same. Variations in the boundaries (compare section 4.6.3) express specific information, whereas the number and sequence of sub-spaces express generic information. Specific numbers may change without giving rise to a new interpretation of the situation and sensor data is given satisfactory meaning if the context of successful progress is reflected in equivalent sub-spaces.

Three sub-spaces are of particular importance here. In the first instance, the mating has to start from some point away from the first contact to avoid collisions. Although the Approach could be evaluated with suitable algorithms, solutions are not necessarily unique. In order to avoid such confusion, the design stage can specify this information more comprehensively.

Secondly, if the start of Mating must be defined, then the finish must also be specified. This second sub-space represents the overall success, as all operations have been executed according to expectations. The regions have departed sufficiently so that they will not interfere physically with each other. An explicit reference to it avoids ambiguity and is easily incorporated into any data structures.

Thirdly, the sub space corresponding to the final end-configuration must also be identified explicitly. The reason can be found in the potential complexity and ambiguity of error recovery situations. For example, if some actions are executed which bring the process into a defined state, but miss out the important final mating point, then it is possible that the controller will continue from that state onwards. To avoid such a miss, the cell must be able to check for the correct configuration by examining the history of the process. If the end-configuration region was reached, then the process was successful, but if it was missed, then some indication towards the error situation can be extracted. The necessary information for generic mating processes is summarised in figure 5.17.

Generic Mating	
Identifier	For reference
Start	Reference to the Approach Sub-Space
Finish	Reference to Final Sub-Space
End-Configuration	Reference to End-Configuration Sub-Space
Information Axes	List of Information Axes
Sub-Space	List of Success Sub-Spaces
Action	and their Associated Actions

Figure 5.17: Generic Mating Information

At this point, much of the very detailed information can be hidden from the operator, allowing him to concentrate on the process rather than automation machinery. If there is any doubt about the validity of the computer's model, all information can be recalled, as the control computer must have every detail available anyway in order to achieve the desired goal of the end-configuration.

5.4.3 Equivalence of Success Actions

The actions assigned to each of the sub spaces are also part of the criteria for the classification of generic mating processes. In a similar way to sub-spaces, actions are not required to be exactly the same, as some variations in action parameters do not necessarily change the context of the mating operation. However, for each sub-space, the action equivalence is important because actions are designed to guide the mating from one success state to the next. Thus they affect the sequence in which the success spaces are expected to be traversed.

These arguments arise from the derivation of information sub-spaces from geometric considerations. Usually, one route has been decided upon which has to be followed. Possible alternatives for that route have already been eliminated at previous stages. For example, the identification of one sub-space during insertion may require a *push-down*. The manner of achieving this may vary, such as differing robot speeds, but the concept of *push-down* must be observed. Any other action would lead to a different generation of the sub-space sequence.

An example for generic information is given in figure 5.18 for the 'standard' example of the bush-onto-motorshaft assembly:

```
Keyed_Hole_Onto_Peg;                            (* Identifier *)

Apart,                                          (* Start sub space *)
Ok,                                             (* Final sub space *)
Final;                                          (* End-configuration *)

Z_Force, Z_Translation, Gripper_Opening;(* The three info axes *)

Apart,          go_down;                        (* 7 regions with their *)
Chamfer,        go_down;                        (* actions              *)
Sliding,        go_down;
End_Stop,       go_down;
Final,          open_gripper;
Retract,        go_up;
Ok,             null_driver;
```

Figure 5.18: Example for Generic Data

The use of these three criteria enables a systematic building of a library for generic mating operations. The first two stages, Feed and Transport, are not contained in this description, as the manner in which they can be achieved successfully is not part of this research[5,6,7]. Previous explanations (see chapter 3) have highlighted their distinct characteristics and interfaces. Until some more research can identify the processes involved and their characteristic information systematically, Feed and Transport requirements are arbitrarily assigned to belong to the product specific data in the demonstration cell implementation.

5.4.4 Product Specific Data

Essentially, all information specific to the product itself, is represented by numerical (or sometimes symbolic) data which is used to *fill in* the empty variables in the generic description. As mentioned previously, the Feed and Transport for the active and passive regions remain an exception, as they will have to be specified explicitly. In practice, it is expected that the data will be stored in a library or even in a separate file, and hence the collection of product specific data can be referred to as the Product File.

A succession of FTM pairs is contained in the Product File, which at this stage does not need to consider the overall relationships of actions and geometry of components any longer. In effect, such determination assumes its importance elsewhere, usually on the software layer 'above' the FTM specification, such as design or as assembly planning. In fact, the corresponding pairs of FTM have no direct relationship to each other at this stage, and even the recursive nature is not obvious in the Product File. The requirement remaining is an even number of FTMs.

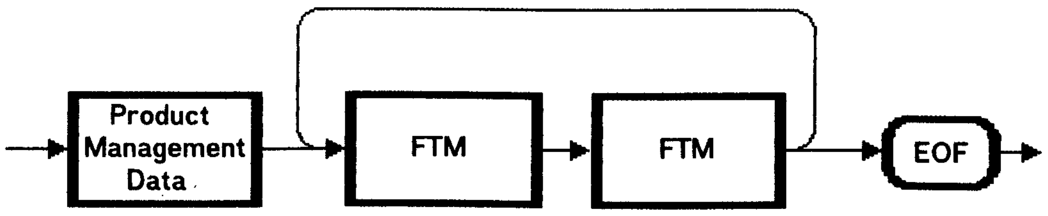


Figure 5.19: Product File Specification

Figure 5.19 shows the general form of the Product File, where the FTM information is expanded in figure 5.20. The first box containing the Product Management Data is a general collection concerning organisational requirements not directly needed for the assembly process. One obvious example would be the number of products to be completed in this batch. The precise content and format must be left to individual implementations and can even be omitted from the Product File. However, any practical system must concern itself with such aspects.

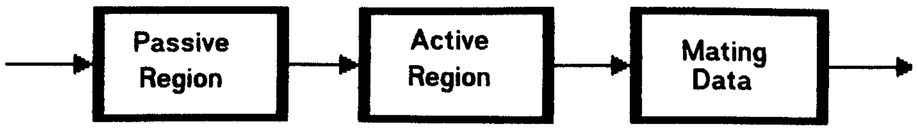


Figure 5.20: Product Specific FTM Information

The definition of specific data for active and passive regions and the Mating Data is implementation independent. The requirements are listed below:

i) Product Specific Passive Region

The specific information for passive regions consists of the identifier and driver for the feeding mechanism. The driver must assign the suitable frame and offset as well as the actual hardware to drive the physical mechanism.

ii) Product Specific Active Region

Again, only the identifier and the path generator are specific to a particular product, because the context of one FTM allows for the correct reference to the Passive Region as well as to the Mating process. The requirements for both active and passive regions are thus satisfied by the provision of suitable driver routines and their parameters.

The level of autonomy in the cell is not dictated by this general structure. For example, if there are only relatively simple routines available for feeding, then each passive region may have its own *customised* driver. Similarly, any path generation can be used, suited to the cell's circumstances. At the lowest level, simple VIA points may suffice, and thus only a list of these locations need to be supplied as parameters. More complex routines may be invoked by different driver routines.

iii) Product Specific Mating Data

A considerably more detailed account of the mating specifics must be available in this part. A close match is necessary with the generic information and thus the precise form of the specific mating information depends on the generic class. The latter must be referenced so that it can be looked up in the generic library. After that, the numerical and symbolic data is provided according to the *empty variables* in the referenced mating process.

An example for Product Specific Data is given in figure 5.21. It indicates the format of the specific data.

```
(* Passive Region: *)
Motor_Shaft, Feeder_Driver, Motor_Jig, Jig_To_Shaft;

Bush_Inside, Path_Generator, VIA; (* Active Region *)

Bush_Shaft_Approach, (0,0,29,0,0,0); (* Approach Transform *)

Keyed_Hole_Onto_peg; (* Generic Mating Operation *)

ASTEK_Driver, z_force,          (* Three Information axes *)
ADEPT_Driver, z_translation,
HULL_Gripper, gripper_opening;
(* Sub Spaces: *)
Apart,    -0.25, 4.0,   14.0, Pos_Infinite, 3870, 3920,
          ADEPT_Driver, go_down, moderate;
     (*  ←   Force   →   ← Position →  ← Gripper → *)
Chamfer,  -0.25, 10.0,  11.0, 14.0, 3870, 3920,
          ADEPT_Driver, go_down, slow;
      (* ←       Action        → ← Parameter → *)
Sliding,  -0.25, 10.0,  1.0, 11.0, 3870, 3920,
          ADEPT_Driver, go_down, fast;

End_Stop, -0.25, 10.0, -0.5, 1.0, 3870, 3920,
         ADEPT_Driver, go_down, slow;

Final,    10.0, Pos_Infinite, 1.0, -0.5, 3870, 3920,
          HULL_Gripper, Open, Fully;

Retract,  -1.5, -0.1, -0.5, 30.0, 4660, Pos_Infinite,
          ADEPT_Driver, go_up, fast;

Ok,       -1.5, -0.1,  30.0, Pos_Infinite, 4660, Pos_Infinite,
          Null_Driver;
```

Figure 5.21: Example for Product Specific Data

5.5 Control Structure

The cell's overall control structure is required to manipulate this data in an appropriate manner so that the assembly does progress as it was planned. Although the shape and sophistication of the software can vary significantly, two particular functions need to be included in order to create the correct environment for the system to perform correctly: initialisation and error handling.

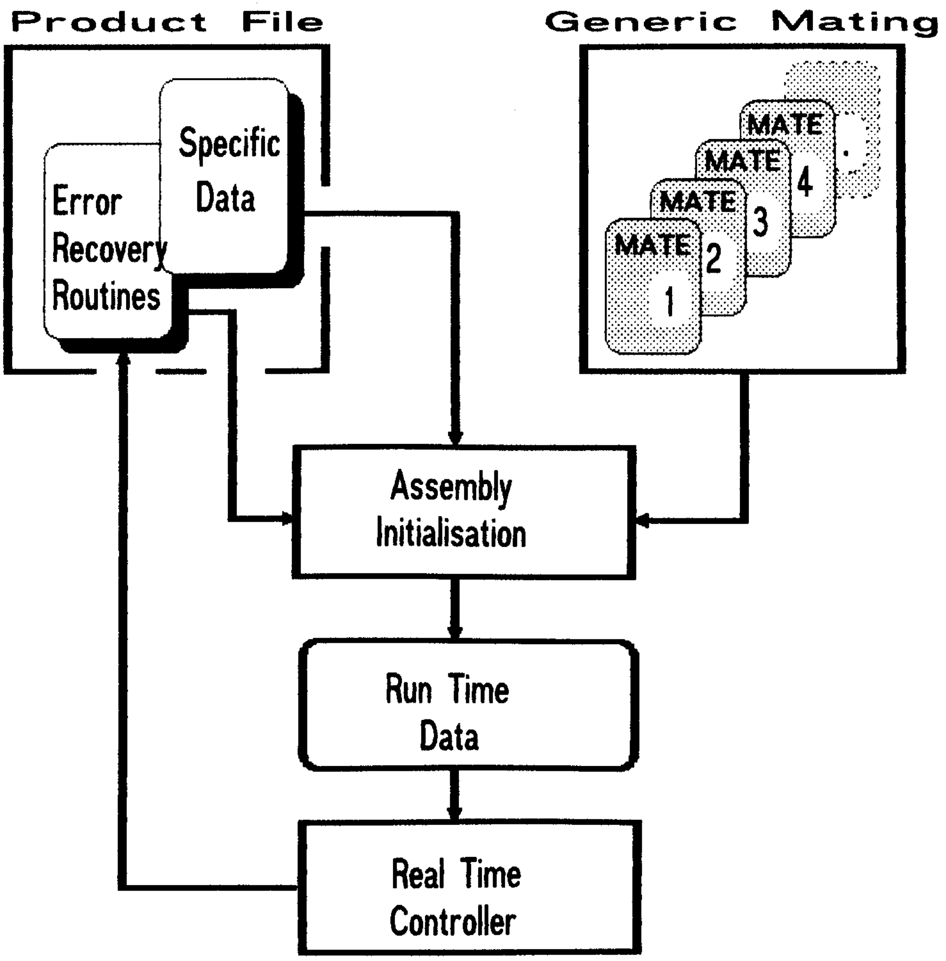


Figure 5.22: Relation between the Assembly Information

5.5.1 Cell Initialisation

A suitable environment must be created so that the data can be operated on correctly. It can be assumed that an appropriate program can be written and that such a program does not depend on set-up specific circumstances, but can use the data to adjust to specialities of the product.

Figure 5.22 shows the relationship between the product specific information and the generic mating library. Before the cell can produce any goods, the generic data and the product specific *numbers* must be combined. Only after this combination is carried out, the run time data becomes special to the cell's setup and a specific product can be manufactured. The real time controller can act upon the instructions encoded in the FTM data.

Naturally, any desired sensors for the information gathering process have to be accompanied by their drivers. Loading the correct ones is another step for the initialisation to perform. Much of the consistency and performance checking can also be carried out; for example, can all information required also be obtained because the correct drivers are loaded, all desired devices are physically attached and are able to communicate?

Before the assembly can be started, the cell controller must ensure that all Datum frames have been calibrated. House-keeping data has to be consulted now or the relevant routines have to be executed as a matter of course. By the same arguments, the manner of teaching Offsets has to be taken into account, as undefined Offsets will almost certainly lead to irrecoverable error situations.

Finally, another important aspect of self-calibration can be included at the time of initialisation. The first assembly in a new set-up can be run in a *trial mode*, where the performance can be observed by the operator and subsequently confirmed (or adjusted). Potential deviations are re-taught *automatically*, because the known successful progress in the mating stage together with recovery actions can cause changes in the final Offset once the Datum frame is defined. The only demand on the operator is made through the necessity to confirm the correct progress despite any self-invoked adjustments. Thereafter, the detailed calibration is incorporated without any further attention or checks by the operator and deviations are now identified as errors by the cell itself.

5.5.2 Error Handling

The additional aspect of error recovery routines is contained in figure 5.21. The experience gained from the observation of the cell on the shop floor indicates very strongly that any error recovery actions must be closely associated with the product itself and the specific setup of the cell (compare section 2.4.3). In particular, some circumstances are peculiar to a particular cell design as well as to temporary conditions under which it may operate. As a result, new error situations may emerge which would have been impossible to predict in practice.

However, the notion of *successful assembly* also requires some routines to recover from relative minor errors. These are those situations which would have been pre-programmed instinctively in conventional programs, since the inclusion of these exceptions would have been regarded as part of the assembly itself. Naturally, not only is it necessary to include obvious error situations even before the cell is started, but also those it has learnt while it was in operation previously.

The run time controller also produces data for handling new error situations and these are available as soon as the same error appears again. All this temporary data must be stored in the associated error file for use at the next setup, compare figure 5.22. Over a longer period, most error situations will manifest themselves so that the system will become its own expert, requiring less and less attention from operators. In this way, it is very likely that the most commonly occurring errors are eliminated very quickly.

5.6 Summary

This chapter has dealt with the basic data structures which are necessary to support the FTM system with respect to successful assembly. The advantages of data-driven assembly control have been explained, as well as how the viewing of active and passive regions as collections of information is consistent with tackling the assembly process itself in a sensor-rich environment.

In that context, the concept of Local Accuracy was explained, and the ways in which it aids in the reduction of detailed on-line teaching. The latter is the main reason for dividing the assembly data into generic and product specific parts, which must be combined at the time of initialisation to form the Actual Mating data for control at run time.

The structures proposed are implementation independent and have specified boundaries at which individual hardware and also software systems can be incorporated.

Although the structures also allow for handling of error situations, the main subject of this thesis is the systematic structure for error detection. Of course, such a feature is only useful if the apparent error situation can be rectified[14,15,19]. Hence the situation must be identified so that appropriate actions can be chosen. The organisation of the error identification and the generation of new (or old) recovery routines are two additional fields which have had considerable attention elsewhere[18,24,25] and have been shown to be non-trivial. As a result of those arguments, error identification and recovery have deliberately not been investigated any further.

Chapter 6

The Demonstration Cell

The concepts developed in chapters three, four and five have been implemented to a large degree in a demonstration cell. The cell incorporates a number of sensors, an ADEPT II robot, a DEC MicroVAX II as the main controller and an in-house communication structure called ROBUS. Work has concentrated on the implementation of one product, the power unit of the FLYMO E25 lawn mower.

This chapter explains the choice of the assembly together with the individual elements in the demonstration cell and their role in the assembly of the power unit. Not all of the ideas presented previously have actually been put into practice, but the general underlying principles have been observed. The differences between the implementation and the system proposed here are explained and the realisation of error results for the test of the local accuracy are presented. The combination of the residual errors with the ability to accommodate relatively large deviations are shown to provide a significant reduction in the complexity for explicit teaching of exceptional situations.

6.1 The FLYMO Power Unit Choice

The study of assembly and its errors has been presented in chapter two, where the FLYMO assembly cell[2] provided an excellent opportunity for this work to study *real life* behaviour. It was therefore a logical step to choose the same assembly for the demonstration cell, particularly since some additional factors highlighted the need for more sophisticated control of the automated cell. These reasons will be expanded upon in the following as the cell structure and the devices used in the cell are described.

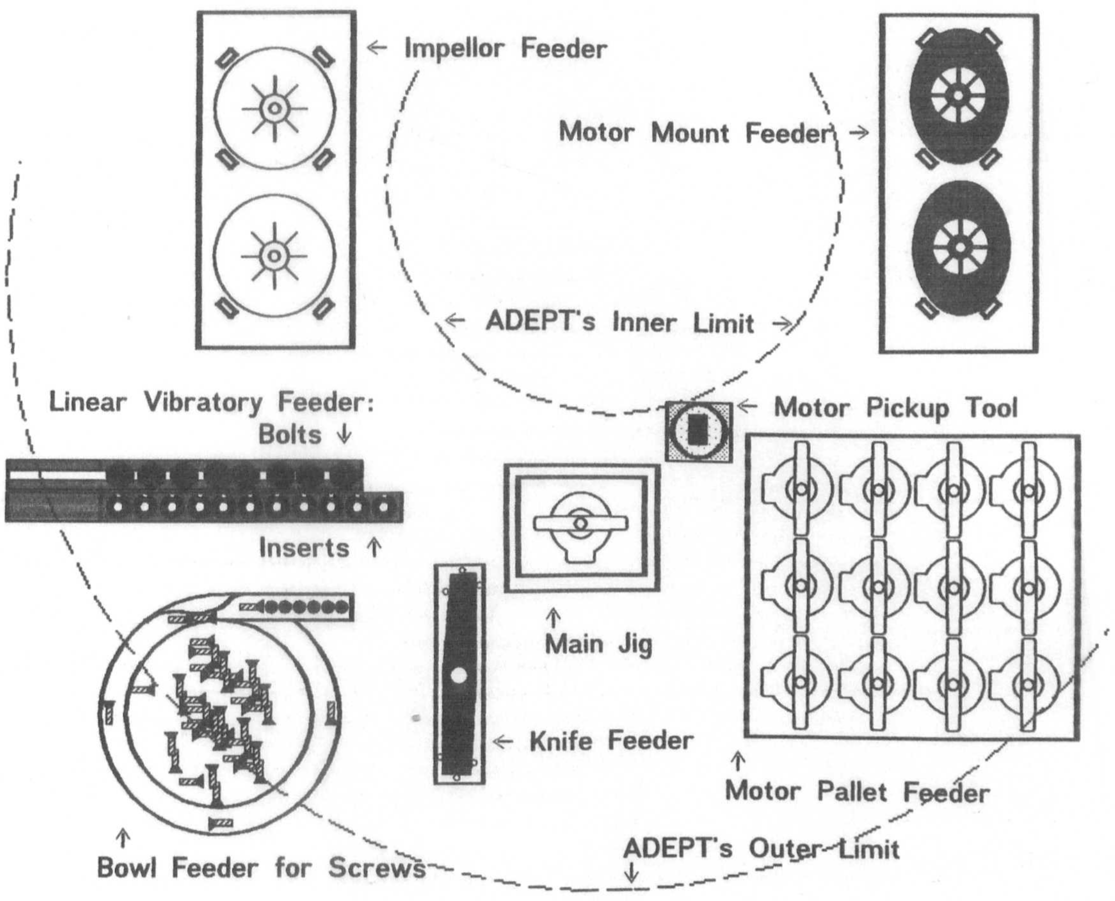


Figure 6.1: Demonstration Cell Layout

6.1.1 The Cell Layout

Before the characteristics, which make the power unit attractive for use in a demonstration can be explained, the physical components and their layout need to be given. Figure 6.1 gives a schematic overview of the placement of feeders. This layout has been chosen empirically in the absence of suitable tools. This does not detract from the main purpose of the work presented here, mainly because the aspects investigated did not require an optimal placement.

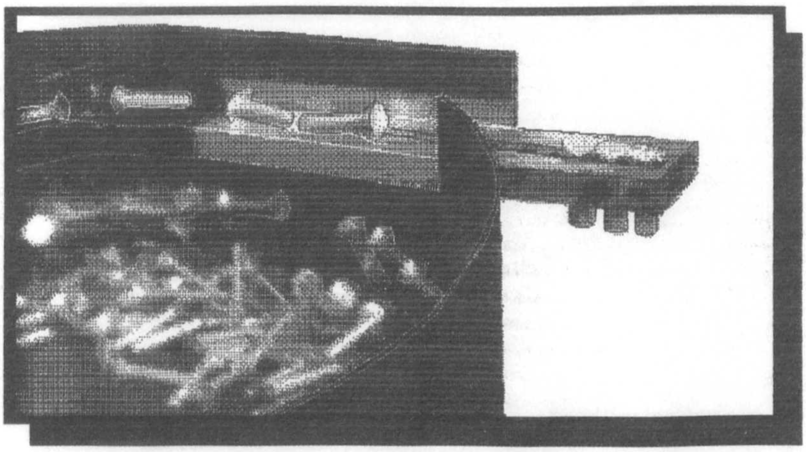


Figure 6.2: Vibratory Bowl Feeder for Screws

6.1.2 Variety in Feeding Mechanisms

One of the attractions in this assembly was the variety of feeding mechanisms necessary for the different components (compare chapter 2). The screws are fed by a standard bowl feeder (see figure 6.2), and although the final shape of break points and orientation guides is not a precisely quantifiable science, relevant expertise exists (from dedicated automation) and is widely accepted. Similarly, the linear vibratory feeder (as pictured in figure 6.3) which is used for the bolts and metal inserts, is also well used in industry. Although the number of components which can be held in reservoir at any one time is almost certainly too small for real production cells, the variation on bowl feeders was a potential source for different errors.

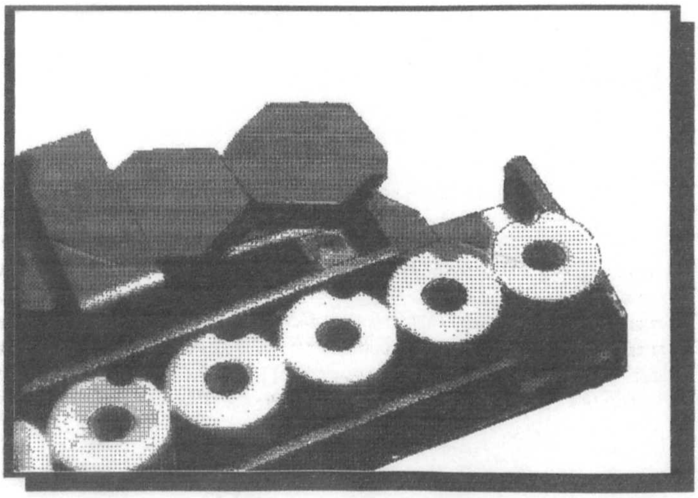


Figure 6.3: Linear Vibratory Feeder for Bolts and Inserts

The whole cell was designed to run in batches of about one dozen; a number thought to be high enough to simulate production flow. Since no real production was planned, the manual disassembly of products leaving the cell had a decisive influence over upper limit of products.

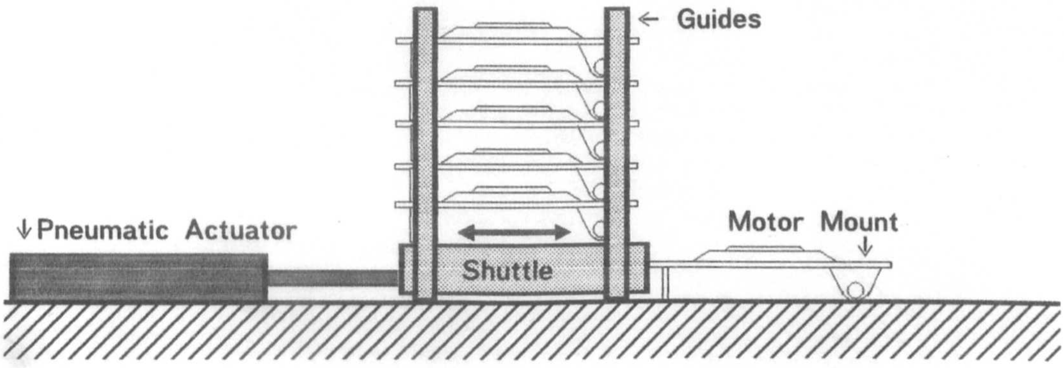


Figure 6.4: Stack Feeder for Motor Mount

The impellor and the motor mount were both fed by a stack mechanism, which used a shuttle to push the bottom component out to the point where the robot can pick up the part. Appropriate catches must be provided so that the weight of the stack does not cause jamming. The original feeders were designed at FLYMO in Darlington and were made available to the demonstration cell. Figure 6.4 shows the principal feeder mechanism as a side view, from which the feeding action can be seen more easily.

The feeder for the impellor functions in a very similar way, except that the dimensions and guides are different. The provisions for manipulating the falling stack are not shown. Figures 6.5 and 6.6 show pictures of the actual motor mount and impellor feeders respectively.

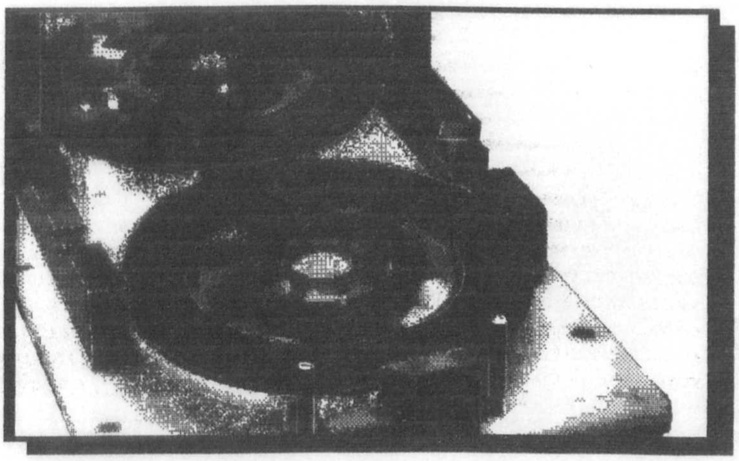


Figure 6.5: Motor Mount Feeding

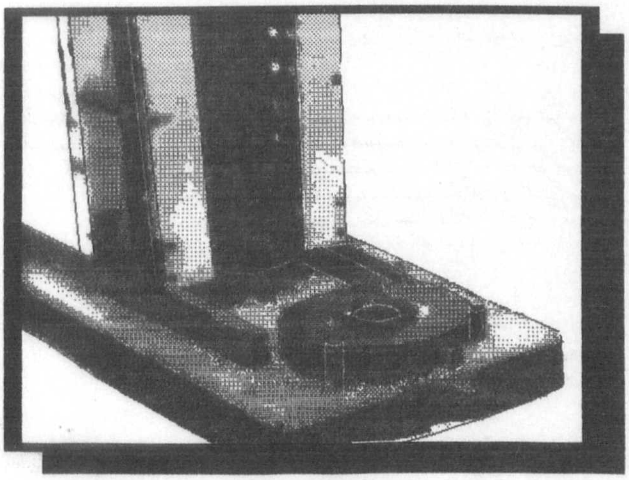


Figure 6.6: Impellor Feeder

In contrast to the previous feeding devices, the lawn mower's knives are fed on a stack which has no actuation on its own. Consequently, the robot must be capable of picking up the component from the right point which varies throughout the production run. The same considerations are valid for the pallet feeder in which the motors are presented to the manipulator. Figure 6.7 is a picture of a partially emptied motor pallet feeder and figure 6.8 shows a picture of the stack feeder for the knives.

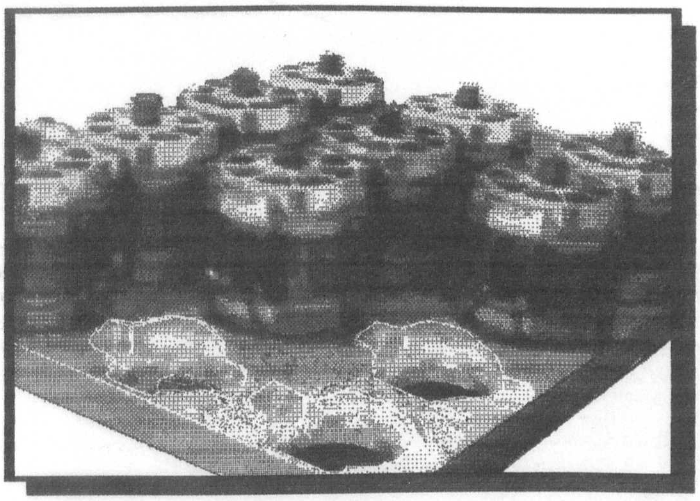


Figure 6.7: Motor Pallet Feeder

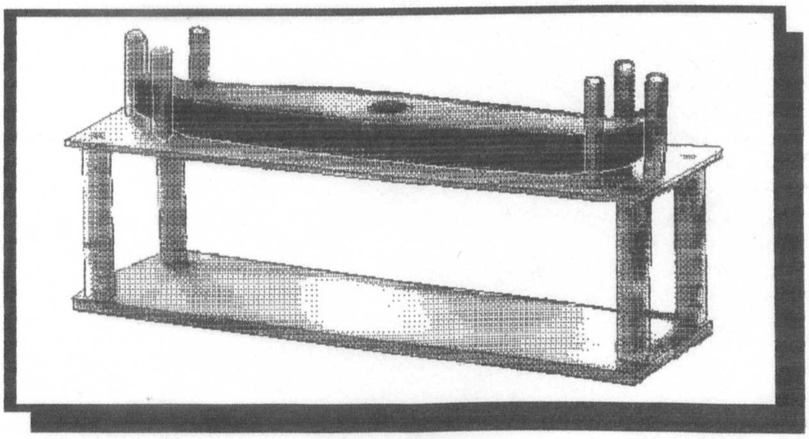


Figure 6.8: Stack Feeder for Lawn Mower Knives

These positional variations are handled by the feeder driver in addition to its basic responsibilities for the physical interfacing. The relevance of Datum frames and Offsets has been explained in chapter 5, where the Offset now contains more than one transformation. The feeder takes account of the number of components picked up and adjusts the eventual mating position by returning the original offset from the Datum frame, together with any additional offsets caused by the nature of this feeding method.

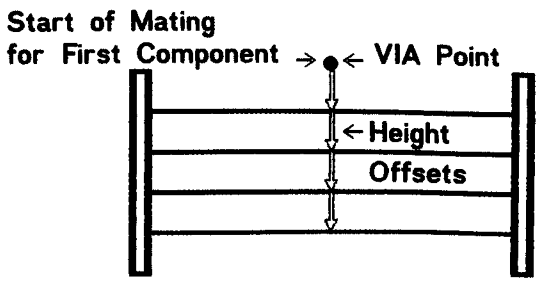


Figure 6.9: Offsets in Stack Feeders

The offsets are illustrated in figure 6.9 for the stack feeder and in figure 6.10 for the pallet feeder. It is interesting to note in this context that the division between obstacle avoidance and the approach to the mating procedure is not always obvious. The stack feeding must include a VIA point so that the knives can always be approached directly from above and no collisions with the guard posts (see figure 6.8) can occur.

In comparison to the array feeder, the equivalence in figure 6.10 is only true because the offsets are pure translations. In general, calculations are more complicated as the order of transformations is important[26], but it is unlikely that such 'complicated' feeding devices are acceptable in a factory environment.

An ambiguity could arise because the start of the mating and the VIA point are physically the same for the first knife, but they differ for subsequent ones. Clear division between the stages in the assembly process (FTM) reduces such ambiguities. This particular problem does not exist for the pallet feeder, provided the gripper is lifted high enough - and that is part of the familiar obstacle avoidance problem.

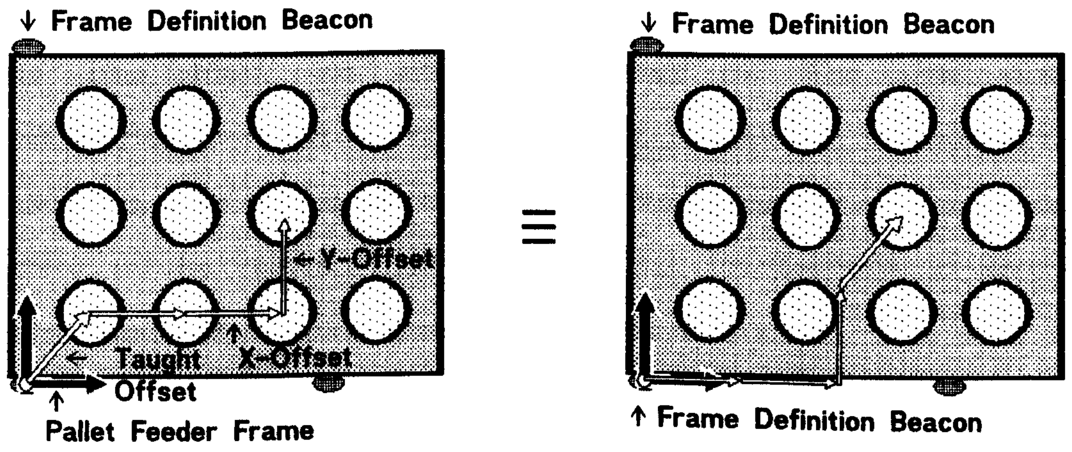


Figure 6.10: Equivalence of Taught and Driver Generated Offsets

Of course, any sensing associated with the feeders is handled by suitable drivers. It may even be necessary to take command over the robot as part of the FTM procedures (see section 5.3.4). This relevant sensors be mounted on the end effector[1], for example.

6.1.3 Varying Degrees of Assembly Difficulties

The variation in feeding was one consideration in the product choice for demonstration, and another one was the degree of difficulty in the assembly process itself. Most of the components in the power unit can be assembled without significant problems (compare section 2), just as would be expected in a designed product. However, since automation had only been introduced after production had started, some operations remained difficult to automate.

Almost all of these problem areas were not concerned with the power unit itself so that they could be tackled manually outside the robot's cell. Examples of difficult tasks include the threading of cable ends and the packing into boxes. The insertion of the bush onto the motor shaft was placed initially in the production cell. The procedure which rotated the shaft and waited for the bush to drop down, was successful during the first year after installation. For reasons which were not established, the operation then began to fail at an unacceptable rate. Since there was some slack in the operator's time, the easiest solution was a manual placement and avoiding problems with the machines.

A significant contribution to the eventual failure was made by the unknown relative orientation of the keys on the shaft and the bush. Solutions to determine the orientation during the feeding would have been uneconomical (if possible at all) and the inclusion of specialised sensing was also too expensive because of the previously described and much more obvious solution due to production pressures.

As a consequence for the demonstration cell, a relatively difficult area was ready to be exploited as an example without the need to introduce artificial errors.

6.1.4 The Need for Auxiliary Tools

Finally, many assembly operations which can be found on the shop floor, cannot be executed with a single gripper[10] and operations not directly relevant to the assembly itself need to be included. The transport of the motor to the main assembly jig is such an example. The only major aspect missing in the cell is the use of multiple grippers, but as this was not vital to the demonstration, this was not considered to be a disadvantage.

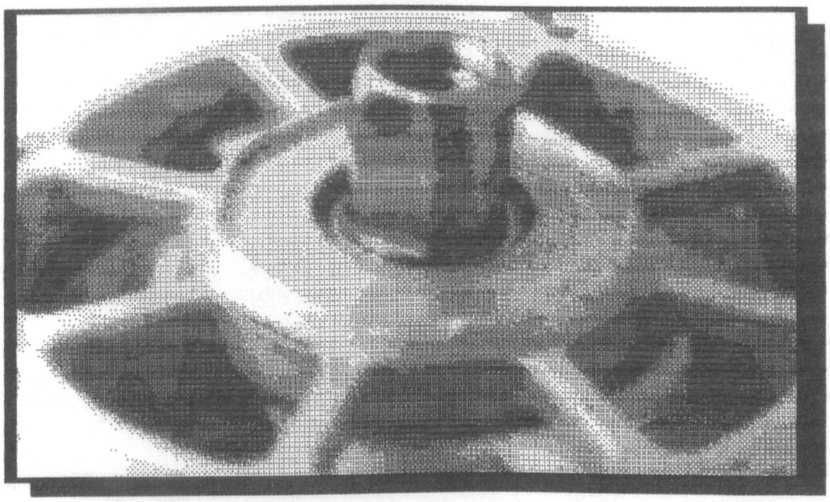


Figure 6.11: View of the Motor Top

Figure 6.11 shows the top of the motor, and an empirical choice was made to utilise the radial support features for transportation. The gripper would neither be wide enough to grasp any part of the motor housing to enable secure attachment, nor would it be strong enough to provide sufficient friction to avoid slippage. The tool is designed such that it could hook into the ribs, and the force of the motor weight is taken up by the mechanical stop securing the tool in the gripper's jaws. Figure 6.12 depicts the tool resting on its holder.

The features on the jaws of the proportional gripper will be described in section 6.3.3, but it suffices to state at this point that a variety of gripping features are accommodated on the jaws in order to avoid frequent changes of tools.

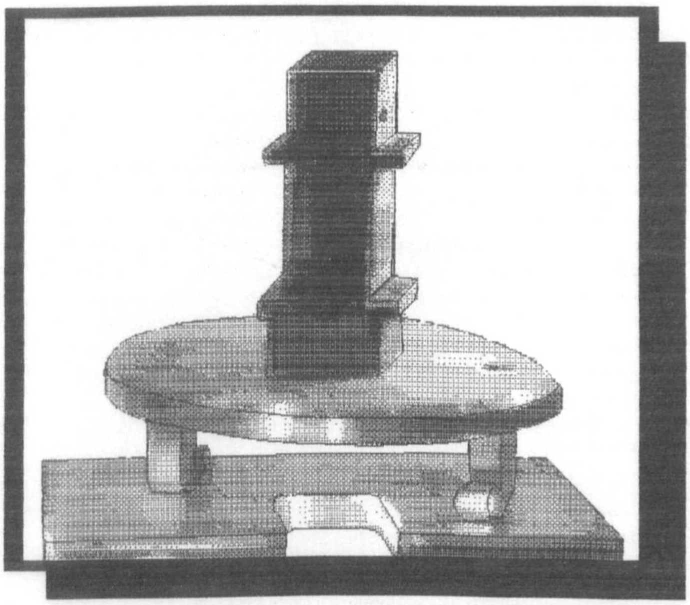


Figure 6.12: Motor Tool Holder

In addition, the way in which the motor is engaged into the hooks, is also a relatively complex assembly, requiring a greater number of sub-spaces to accomplish the correct position, and testing the validity of the system proposed even further.

6.1.5 Temporary Sub-Assembly

Although all products are made of individual components, the assembly does not always follow a strictly *linear* sequence. Sometimes, a sub-set of components must be put together before they can become part of the product. Most of the production processes assume well defined assembly processes to take place outside the cell so that the result can be regarded as a single component.

This is not so for relatively simple operations, or, for example, for temporary fixings arising from the act of gripping. The power unit of the Flymo lawn mower also contains such a requirement. Section 2.1 already contains a description of the assembly procedure as it is implemented in the production cell, and the changes in the demonstration cell are merely related to the activation of a suction gripper instead of permanent magnets. The main reason for these changes can be found in the more general application potential for suction grippers.

6.2 The Hardware Structure

The hardware structure adopted for the research has also been shaped significantly by the requirements for generality. The inclusion of many different devices (either sensing or moving) can potentially increase the system complexity to a point where the addition of one more vital sensor, for example, may result in unacceptable cost increases. Such aspects are clearly highlighted when the communication between devices is based on comonly encountered star-type connections, such as point-to-point wiring using the RS232 standard. In addition, the communication flow has to be directed in a well defined manner, as unexpected consequences can arise from an unpredicted exchange of data between individual slaves.

This is also a problem common with many other factories and institutions and several solutions were proposed[84-89] during the middle to late 1980s. At the time of the design of this cell, no suitable standard was available. The chosen Master-Slave structure addressed the needs in this situation adequately. The fundamental aspects have been published[30] and used successfully for the coordination of a variety of different robotic projects[1,32,68,69] not all of them concerned with this type of assembly.

6.2.1 The ROBUS concept

The main characteristics of a Master-Slave system can be summarised as follows:

- Hardware and software modularity:

  Each of the devices attached to the cell contains a certain number of functions. These vary significantly in complexity and are not necessarily under the control of the cell designer or even operator. Any commercially acquired equipment is likely to be self-contained so that only external functions are visible. Equipment developed during the research is obviously more accessible for modifications, but any practical production cell must incorporate readily available systems, such as robots and vision.

- Secure communications

  Directing all data exchange through the master ensures that the integrity of communication is preserved, and that none of the slaves need to be programmed to *understand* all the others. Any translation, control and interpretation is concentrated in the master, forcing a very desirable modularity.

The realisation of these demands resulted in ROBUS, a communications bus for the control of robotic environments. It is basically a parallel bus structure, designed for the transfer of token commands, mostly assuming a comparatively high degree of intelligence in each slave.

The choice of at most 254 slaves arose from the eight available primary address lines, and addresses 0 and 255 are conveniently defined as illegal. This eases the software control slightly.

In reality, the number of slaves is not expected to exceed much more than a tenth of the devices attached at any one time, since the general desire for modularity and ease of use would place too much a weight onto the Master. However, such a large address range will not limit any additions, should they be required. The data is byte-oriented and the maximum raw data rate over this bus is 1MHz.

Again, this value is at the theoretical end of the scale, since the signals in the demonstration cell are generated by software, achieving a maximum rate of just under 10 kBytes per second. This is deemed sufficient for token commands[79], since a single message will be in the order of 15 to 20 bytes long on average, allowing for a high level communication rate of several hundred times a second. Even if under these conservative estimates three or four slaves are involved at any one time in an operation, the sampling time remains well below that of commercial robot controllers, which is in the region of 50 to 60 samples a second. Limitations are thus not imposed by ROBUS but are likely to have their origins in the generation of data in the slaves.

Even at the low data rate of 10kHz, events can be reacted to within 1ms, allowing for emergency stops or some on-the-fly sensing. ROBUS itself does not demand any manufacturer specific hardware, nor any specific amounts of computing power, since the complexity of the task, availability and convenience to the user should determine such parameters. It is thus feasible to employ a very cheap micro-computer which is based on an eight bit processor as a master. It might act merely as a message router, whereas significant power is needed in the pre-processing of vision data, for example, which is carried out on large Mini computers such as DEC VAX systems.

For completeness, the specifications of ROBUS and its *philosophy* have been included in Appendix A.

6.2.2 Standard Slave Control Processor

It was necessary to develop a simple micro-processor system for the control of specialised equipment in the cell, such as the binary signals and the Hull gripper (see section 6.3). The choice of the Zilog Z80 micro processor resulted from economic arguments and the peripherals were added according to demands in the cell. It soon became obvious that it was more economical to use the same card for most interfacing requirements.

Apart from a small amount of memory (8k RAM and 8k of ROM), some other useful functions were included:

- two RS232 serial lines,
- 24 bit wide parallel interface,
- 3 timers.

The usefulness of these functions stood the test of the applications and it is possible to exploit the small amount of intelligence present for each interface. Sometimes, the slave card was programmed to be a *wire link* between ROBUS and the serial line, thus providing a useful interface to commercial devices. Other times, almost the full power of the system was exploited to control the proportional gripper or provide PLC type functions.

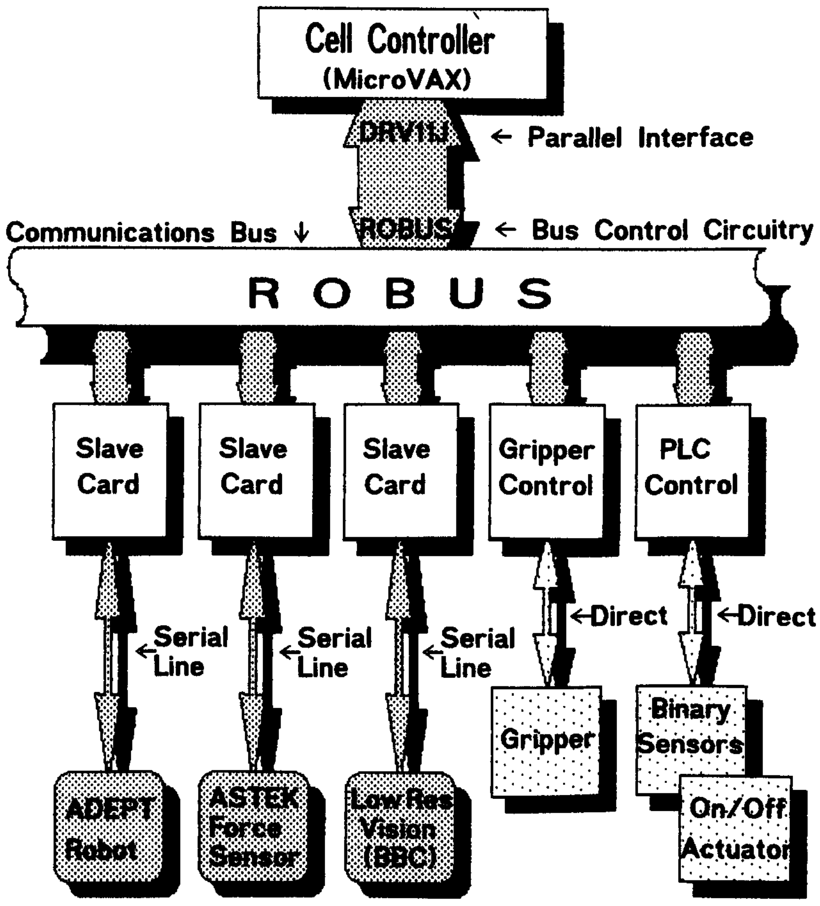


Figure 6.13: Hardware Structure of Demonstration Cell

6.2.3 Master Controller Hardware

Figure 6.13 shows the overall hardware structure of the cell. The master controller was chosen to be a DEC MicroVaxII for the following reasons:

- it is an economical machine powerful enough to run AI software,
- Suitable commercial interfaces allow the connection to ROBUS,
- the size and cost would make this computer acceptable to industry.

As ROBUS is not an industry standard, the electronic interface had to be developed for the MicroVax. A parallel interface was chosen (DRV11J[5]) and the master interface designed accordingly. It is not necessary to elaborate on hardware details here, but it is sufficient to explain that the control lines and the addresses are all generated by software. This simplifies the interface hardware somewhat, and has only a minor influence upon the overall system performance as has been explained before. Comparisons can be made with the specification of ROBUS in Appendix A and the circuit diagram of the DRV11J to ROBUS interface in Appendix B.

6.3 Individual Slaves

Having established the communication and control structure, individual devices can be attached to it. The demonstration cell contains the minimum number of actuators and sensors to perform the assembly. These are described in the following sections.

6.3.1 Manipulator

The most obvious device - from a visual point of view - is the manipulator itself. As has already emerged from previous chapters, an ADEPT-II robot was used for these experiments. It has a SCARA-type configuration and very good claimed performance characteristics. The only way to assume control the manipulator is via the supplied operating system[73], containing VALII as the only language. VALII looks like a sophisticated BASIC with robot-specific routines and data structures provided.

[5] Quad 16-bit parallel interface for DEC standard Q22-Bus

Although VALII is one of the more powerful and user friendly robot languages which are commercially available, its facilities are largely unnecessary within this control structure. In fact, one could argue that the sophistication necessary for controlling the robot and its environment is better separated anyway, but there are obvious advantages[6] when the robot controller contains a program controlling the whole process, including its peripheral devices.

This view of the robot as the centre - in all aspects - of the assembly cell has its origins in early systems, in which distributed computing power was too expensive to contemplate. Furthermore, standard control in industrial installations is exerted by PLC-type controllers, and in the early days of robot deployment, the provision of *'I/O lines'* for binary interfacing was necessary for compatibility reasons. Binary actuation and sensing could be easily provided as functions of the robot controller, particularly for synchronisation purposes. Comparatively powerful communication channels only emerged with increasing complexity and they exist on the ADEPT. Most of the communication functions can be controlled from VALII.

In this strategy, only a fraction of the robot's commercially supplied computing power is utilised, mainly to receive *orders* and send status and position data. One of the more desirable features for *external* control would have been the ability to interrogate the robot's position in world coordinates as well as relative positioning; at a rate comparable to the robot's own sampling rate of 60Hz. Unfortunately, only serial communications are readily available, and the console command WHERE[73] returns information

[6] Complexity and cost is reduced in conventional dedicated cells

tailored to a human operator. Eventually, the fastest method of position acquisition was achieved by writing a small VALII program which transmitted, on demand, only the four world positions (X, Y, Z and Yaw) in the shortest possible format without loosing accuracy. A sampling time of 0.1s could be achieved.

Nevertheless, some other facilities were also useful during the teaching phase when the ability to define and record absolute and relative positions were helpful. Debugging was aided by exploiting the robot controller in the absence of more sophisticated utilities in the master controller[7]. Since the latter was not the objective of the research, some commercial facilities in the robot controller were appreciated after all.

6.3.2 Wrist Force Sensor

Of the commercially available force/torque sensors, the ASTEK sensor, manufactured by the Barry Wright Corporation[70], was the most suitable for this work. The ranges were suitable (up to nearly 80N force, and about 1Nm torque) as well as sufficient resolution (12 bits). An on-board micro processor also gave comparatively sophisticated functions, such as re-definition of the point of origin from which moments are measured and a variety of data formats from which the user can choose the most suitable one.

Values are measured by an arrangement of six strain gauge transducers[71] and transformed into the six force components, three forces and three moments. Limits can be monitored automatically through a simple command, like overload and excess rate threshold. The latter proved to be important, not only because

[7] The first implementation does not contain 'user friendly' features

of absolute force value limits (which is acceptable in any real system), but also limits in the rate of change. As a consequence, contact positions can only be approached relatively slowly to avoid sharper *bumps* and the speed of the robot had to be reduced noticeably in all situations where contact was expected.

6.3.3 Parallel Jaw Gripper With Proportional Control

The force sensor was mounted between the robot's wrist and the gripper. In general, a proportionally controlled gripper with parallel jaws provides a comparatively large amount of flexibility. The gripper designed at the Unversity of Hull offers a very high performance solution by utilising the speed of a direct-drive, Somarium Cobalt DC motor. Using a standard PID controller, the full length of travel is accomplished in under 100ms, while it is able to exert a maximum force of about 10N.

The gripper's performance was adequate for this type of assembly, but an estimated increase in force of about a factor of two to three would be beneficial for a more general application; particularly since the force can be controlled with a large resolution. The power was supplied to the motor via a pulse-width modulated signal, with a repetition rate of 7kHz and resolution of 8 bits. In practice, a force resolution of 1/100 was achievable.

The controller was implemented directly on a standard slave card. Thus the communication did not have to rely on limitations imposed by serial communications. Only the power stage was physically separated, with the advantage that any heat generation could be placed out of range of delicate electronics. Because of the use of PWM, no problems of interference arose when controlling the power *remotely*.

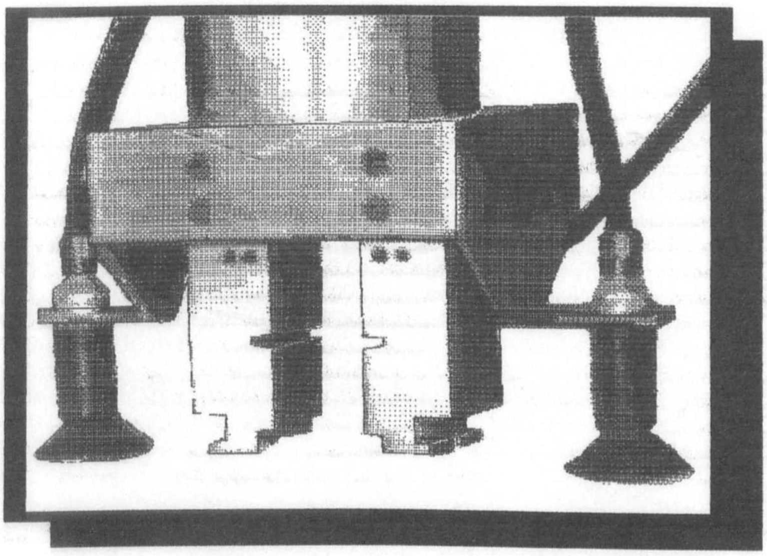


Figure 6.14: The HULL Gripper With Added Suction Cups

The jaws are actuated via a rack and pinion arrangement which converts the rotary motion of the shaft into opening and closing actions. The direct drive nature of the motor allowed the motor to be mounted on the pinion itself and so enables a tight control over backlash in the teeth of the rack, the only possible place where it can occur.

The jaw position is measured by a relative optical encoder with 1000 lines per revolution. By counting every edge of the quadrature signal, the maximum opening of 28mm could be resolved to better than 0.02mm. Such a resolution is well in excess of that required in practical assembly situations. For example, in order to avoid excessive gripping forces, increased friction must be provided on the gripper. Usually, this is done in form of an elastic material such as rubber lining on the surfaces of the jaws. Whereas the chances of slipping are now significantly reduced, any measurements of jaw separation are much more difficult to predict within the available resolution. The values measured at the time, though, provide a good guide line towards the presence or absence of the components in the jaws.

When the gripper was installed in the cell, a number of features were added (see figure 6.14) in order to accommodate the range of gripping mechanisms for this assembly. By careful design, gripper or finger changes did not have to be considered. Two suction pads were mounted one on each side so that they were at the correct distance to pick up the impellor, knife and eventually the completed assembly but also in such a way that they did not interfere in any other way with the normal operation of the jaws.

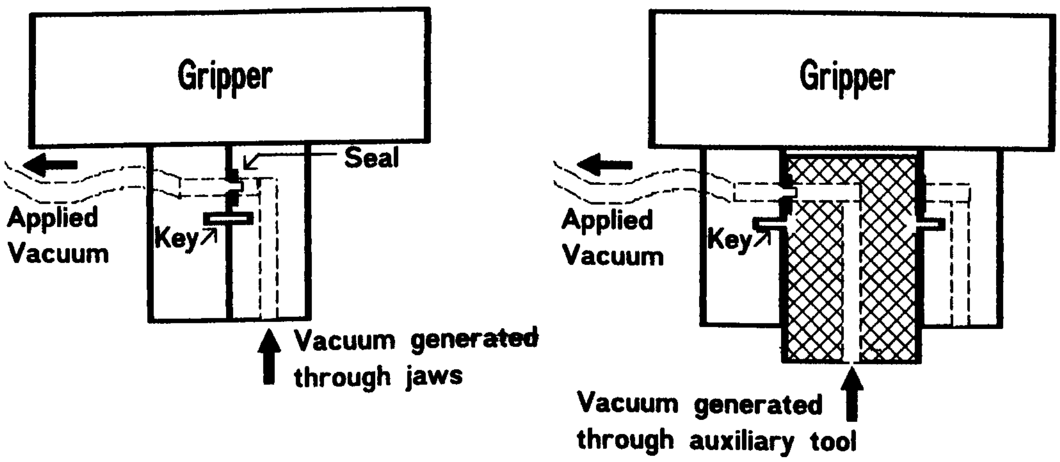

Gripper
Seal
Applied Vacuum
Key
Vacuum generated through jaws
Gripper
Applied Vacuum
Key
Vacuum generated through auxiliary tool

Figure 6.15: Suction Generation Through Jaws

In addition, a further vacuum line was provided which was routed through the jaws. Its objective was the provision of suction for the assembly of the screws and supplying suction to auxiliary tools which had to be picked up by the gripper. Figure 6.15 shows the vacuum path for the jaw suction and for additional tools.

6.3.4 Low Resolution Vision

The second major sensor system apart from the force sensor is a low resolution vision system, with a useable resolution of 64x64[66,67]. Its sensing element is an integrated circuit designed as a dynamic memory for computers. The principle exploits the physical construction of the memory cells, which rely on the storage of a charge for a certain amount of time. This charge will of course leak away in time, if it is not refreshed at regular intervals. However, if all cells are fully charged and the array is exposed to light by choosing an appropriate encapsulation for the chip, then the charge will leak away more or less proportional to the light intensity.

A specific pattern of light will thus discharge the individual memory cells unevenly. Reading back the contents of the memory opens up an interpretation of the cells as pixels which are either black or white and which are arranged according to the image projected onto it. The main advantage can be found in the significant price reduction compared to the usual CCD devices, it is an order of magnitude. There is, however, a loss in image quality, but the requirements for robot vision are such that the information obtainable from these sensors is adequate for many decision making processes. The principle has been shown to work in previous assembly work[65] with resolutions as low as 32x32.

The main role of this vision system is the identification of the frame location beacons (see section 5.1.2). The inadequacies in vision accuracy can be compensated when relying on measurement repeatability. The results of the frame definitions are discussed in section 6.6 .

6.3.5 Binary Functions

In contrast to the more sophisticated sensors, many process parameters are sufficiently represented by simple binary sensing[37] or actuation. For example, the linear feeders and bowlfeeders have no parameters which can be adjusted under computer control, nor do they need to provide this. It is enough to switch power on and then remove it again when the parts have arrived at the feeding point[13]. Similarly, the generation of vacuum is controlled by valves which allow pressurised air to pass through a Venturi type restriction or which shut the air off.

By the same arguments, several types of sensing can be achieved adequately through binary switches, such as proximity, standard micro switches or cross fire sensors (compare section 4.1.2). In fact, any general control system must be able to incorporate these binary functions because of the rich variety of sensing currently used in automation which is traditionally controlled by PLCs. Furthermore, sensor signals only assume meaning when it has been assigned to them; in other words, it is up to cell designers to choose the best equipment (in their circumstances) where, undoubtedly, familiarity as well as availability will influence the decisions. Sophisticated solutions are not always the best.

No assembly can take place without the necessary hardware. The constituent parts of the demonstration cell have been described in the previous sections. For any other assembly, some pieces of equipment may have to be changed, but the amount of flexibility can be increased significantly by making any changes largely independent of hardware. Some pieces of equipment have to remain specialised even in a truly flexible system, but as long as these are limited to only a few or are even exchangeable automatically while not incurring significant costs, the burden of adaptable manufacturing cells falls onto the software control.

6.4 Low Level Software

The control of the complete system is performed at various levels. At the lowest level, data is exchanged in an uninterpreted way. The hardware structure for this is provided by the ROBUS[72] implementation and the associated software binds all functions together. Low level software is very specific to the implementation and constitutes the support for all slave drivers in the master. Two major aspects must be addressed, and the first is the link into the existing operating system, which is not visible to the user. Secondly, the conceptual functions must be made available for the programmer so that the system can actually be used for its purpose.

6.4.1 VAX/VMS Interface

The binding of the ROBUS control into the VAX/VMS operating system was written in MODULA-2[54,55,80], a language well suited to the generation of large programs, but which also contains provisions for hardware access and real time sections. The parallel port interface card was installed and can be accessed in that part of the memory space which is reserved for input/output devices.

The MicroVaxII computer has a very sophisticated memory management system[78] and it is not appropriate to explain the details in this thesis. However, the hardware bus on which the peripheral devices are placed has been made to be compatible with

the older DEC standard of the Q22-Bus. This was retained by DEC for commercial reasons so that proven sub-systems and interfaces such as the DRV11J can be used with the MicroVax range of computers. By extracting the virtual memory address from the VMS operating system, a pointer is set up in the software so that reading and writing is accomplished easily in MODULA-2 by defining the registers in the DRV11J as a RECORD[54] data structure (compare figure 6.16).

```
TYPE
        SHORTSET  = SET OF [ 0..15 ]; (* additional types for control *)
        BYTESET   = SET OF [ 0..7 ];
        DRVWord   = RECORD (* logical split for ROBUS interface *)
                      DataByte    : BYTE,
                      StatusLines : BYTESET,
                    END,
        DRV11J    = RECORD (* FOUR 16-bit parallel ports *)
                      CSRA  : DRVWord,  (* Control Status Register A *)
                      DBRA  : SHORTSET, (* Data Buffer Register A *)
                      CSRB  : DRVWord,  (* Control Status Register B *)
                      DBRB  : SHORTSET, (* Data Buffer Register B *)
                      CSRC  : DRVWord,  (* Control Status Register C *)
                      DBRC  : SHORTSET, (* Data Buffer Register C *)
                      CSRD  : DRVWord,  (* Control Status Register D *)
                      DBRD  : SHORTSET, (* Data Buffer Register D *)
                    END,
```

Figure *6.16: Type Definition for Parallel Interface DRV11J*

Only two of the four sixteen-bit-ports needed to be used for the control of ROBUS, and these are set up via their own control registers, one as an input, the other as an output port. Thereafter, ROBUS is controlled merely by setting various bit patterns on sections of the port's StatusLines, **making the DRV11J part of the overall control structure.**

6.4.2 ROBUSIO Module

By exploiting the facilities offered in the low-level module ROBUSIO, all complications arising from the operating system are hidden from a potential user, who only has to include an initialisation routine at the beginning of the program. Appendix C contains a listing of the DEFINITION MODULE ROBUSIO in which details of the functions can be found. In general, various routines are made available to the user, who can choose the level of access. Under normal circumstances, the RobusRead and RobusWrite procedures suffice, as all generation of control signals is provided within these routines.

A further inspection of the module reveals the close relationship of the software and the hardware. Any Device Driver mentioned in chapter 5 will need to use these routines at some point, and thus the Device Drivers represent the interface between the general system and the specific hardware. It has to be remembered that the Device Drivers in chapter 5 have a more general role, but the principle of modularity is observed in the overall system. Different implementations on other master computers need to concern themselves only with the low level part of the Device Drivers.

6.5 Higher Level Software

The higher level software can thus concentrate on the conceptual aspects of assembly implementation and need not consider the detailed interfacing problems. Naturally, the better the understanding of the system and its integral parts, the more likely it is that the power is fully exploited. So there is not a clear division between *low* and *high* levels, but rather a grey area.

6.5.1 AI Languages

The heart of the assembly control program was written in PROLOG[33] which itself was supported by its own environment (POPLOG) within the VMS operating system. The justification for the use of an Artificial Intelligence language was given through the use of AI techniques, such as data base searches. An appraisal of this choice is given later in this chapter.

The main function of POPLOG was the interfacing of the AI language to the hardware controlling routines. The generation of assembly control, building of regions describing the actual mating, transport and feeding was written in PROLOG. Controlling routines were kept deliberately at a very simple level, partially because of the desire to control the assembly process from the data (compare section 5.2), and partially because *ideas* were to be proven in this project; commercially viable software was not the aim. Thus the data itself is of much greater importance; its use remains an implementation dependent aspect which can be improved.

6.5.2 The Data

In section 5.4, the arguments for dividing the assembly data into its generic and product specific parts are given. Generally, any proposed methods for classification of assembly processes[13,9] vary significantly with the criteria applied, for example, whether manual or automated (usually synonymous with dedicated) assembly is considered. Very often, it is the intention to provide aids at the design stage for more efficient products. For the purpose of this work, it is sufficient to apply the criteria outlined in the

Information Space description (Chapter 4) and Data Structures (Chapter 5) to the processes contained in the FLYMO cell. The suitability of this as a representative problem has been discussed in section 6.1.

The generic data used in the PROLOG control program is given in detail in Appendix E. There are presently only a few processes worked out in detail and their importance lies in the methodology leading to the generation of these structures. It is also not difficult to see how a more rigorous approach to *all* assembly processes can result in a library of generic mating procedures.

As one would expect logically, the product specific data detailed in Appendix F is more voluminous, and only one product is being studied. The correspondence between generic and specific data is clearly preserved through the choice of representation and the 'internal' cross references. Appendix E and F contain the data actually used during the experiments and it its obvious that the format does not correspond directly to that proposed in chapter 5.

6.5.3 Implementation Differences

These apparent differences are due to the fact that the investigations are subject to almost continual change. Although the basic ideas have been established at the beginning of the project, their form and final implementation have developed throughout the period and are - to an extent - still subject to change. At the time of submission of this thesis, the necessary data structures in Chapter 5 had been designed and evaluated, but the older format remained embedded in the experimentation software.

In fact, there is no conceptual difference in either data representation. After some experience with applying the 'evolved' structures, their inefficiencies and defects became apparent and new and more suitable formats needed to be established systematically. Both remain equally valid, except that the duplication of the information, for example, should be obvious when comparing Appendices E and F.

6.6 Frame Definition Tests

The implementation of the Local Accuracy concept (see section 5.1) is also heavily dependent on the choice of sensor system at the time of the cell design. One convenient way of realisation in the demonstration cell was the placement of beacons at points which define the frame. Figure 6.10 shows the location of three beacons - in the form of LEDs - for the pallet feeder. It is assumed that all LEDs are mounted at the same height, thus relaxing the requirements for height determination and allowing the use of a conventional camera in two dimensions.

6.6.1 Beacons in a Horizontal Plane

There must be a minimum of three points in order to define a calibration frame uniquely[73]. One must be placed at the origin of the frame, and the second on one of the coordinate axes. The choice of the x-axis is usual and used also in the ADEPT VALII system. The third and last is associated with the y-axis in such a way that it is positioned on a line which is parallel to the y-axis. This enables a practical implementation of the frame definition, as it would be impossible otherwise to maintain an exact point on the axis. The choice of direction of all axes is a matter of convention, but as soon as the x- and y-axes are fixed, the third one can be calculated from the cross product of the two existing ones.

Although the height is assumed to be equal for all beacons in this implementation, the conditions described above are valid for any three points in space. All product-specific equipment within the workcell is marked by beacons so that self-calibration can be carried out whenever a change of product takes place. Of course, one calibration frame can be used as a reference for many other locations, such as the jig for the motor serves to define the placement of the motor, the four locations for the screws, the motor mount placement plus a few more.

6.6.2 Calibration Tests

Commercial robots are not very well defined in numerical terms and despite the availability of repeatability data and sometimes even accuracy information, the way a robot calibrates when it is switched on for a session also has significant influence on the performance. This was one of the points raised by operators when observing the FLYMO cell (see chapter 2). Hence tests in the laboratory were carried out over several hundred runs, which were spread over over a few weeks. In this way, the results also contain robot specific calibrations.

When using a vision system to direct a robot towards a goal, the vision system and the robot must be calibrated with respect to each other. This usually represents a very time consuming exercise and a method was developed to perform this calibration automatically. The precise algorithm is described in Appendix H. As a consequence of such a self-calibration method, the height during the initial measurements when identifying the positions of the beacons was not critical.

The robot mounted camera needs to be guided over the LED so that it is somewhere within the field of view. The algorithms then takes over, performs a few test manoeuvers and uses that information to obtain the calibration between camera and robot. This *rough* positioning was simulated by generating random numbers during the tests and merely observing a maximum limit so that the beacon remained within the field of view.

Most of the tests were carried out on the motor jig (compare figure 6.1), which was placed approximately within the medium position of the robots reach. Both important angles on the shoulder and elbow joints were thus in their middle range. This has some influence upon the repeatability[65], as the radius of the links can change the numbers quoted by the manufacturers. The origin was found with a standard deviation of 0.82mm from the mean, and the maximum distance encountered was 1.72mm.

Potentially, the rotation of the frame relative to the robot does have more influence upon the repeatability, depending on the offset within the calibration frame. The standard deviation for the angle was measured to be 0.12°, and the maximum error was 0.45°. Taking a typical *short* distance of 100mm along each axis, the standard deviation in position increases to 0.92mm. Using the automatic calibration facility with the dynamic RAM vision system, the point at which mating begins, can be found within ±1mm.

Within an environment relying on blind repeatability, such values are not precise enough and the resulting error rate would certainly be too high. However, these large deviations are only significant whilst the cell is in a teaching mode which contains this self-calibration procedure. After the mating process has re-established the correct point for the end-configuration, only the repeatability of the robot is important. Corrections need only be executed in the first *trial* run, with minimal interference by the operator.

In fact, the rough positions and the offsets need to be taught only once so that the robot can use this information to start its own calibration procedure. As stated already, the operator merely has to confirm (or correct) the validity of the process through the cell's own observations. Any errors are then expected to be identified automatically.

6.7 Error Recovery Implementation

The advantages of detecting errors become apparent, only if the cell can take actions to recover from that situation. Since the monitored progress so far has concentrated only on the 'successful' parameters, it is not likely that the newly encountered errors can also be differentiated sufficiently using the same information. New sensor values have to be acquired in order to establish the causes as the best recovery action could not be chosen otherwise. The demonstration cell contains only a very simple - and thus limited - recovery mechanism.

6.7.1 Recovery Mechanisms

All error situations and their associated recovery strategies are recorded in a look-up table. If a potential error situation is encountered, a matching process is attempted trying to identify the current state with one of the known errors.

If the sensor signals provide the information for a match, then the action associated with that sub-space is executed. There are currently two different ways of trying to return the states to those associated with the successful progress:

- **Invoke a 'DO-UNTIL' Routine**

  This condition needs monitoring in order to check when the current error conditions are not true any more. It is equivalent to entering another Information Space in which the conditions are now defined to monitor the relevant axes for the error condition. An example can be found in the alignment of the key-features on bush and motor shaft described in section 6.7.2.

- **Execute a single action**

  This situation is generally associated with conditions which are clearly known and a single adjustment suffices to restore the desired assembly progress. An example of this can be found in the compliant adjustments, also described in section 6.7.2.

In either situation, renewed checks are done after leaving the error sub-space in order to establish the new situation. Such checks take account of errors which are not removed by single actions (or a number of repeats) or where the exit from this space may lead to another (and possibly new) error. Should that be the case or if there was no identifiable situation initially, the operator has to be consulted as to the identification and possible recovery action. It is then simply added to the list, and linked to the generic mating procedure currently used for this product.

It is at this point that the experience of operators is called for. It is up to them to identify the situation which has led to the new error. A human element is much more likely to come to a correct conclusion about the circumstances because their sensing can take a great number of additional factors into account. However, operators are not left to guess work, as the cell controller can provide a precise *path* which leads to the new conditions. Errors already *contemplated* but dismissed are also available during the first investigation and hence the accumulated experience of the cell eliminates unnecessary and lengthy diagnostic searches.

The last point becomes important in view of the unsupervised operation and also in view of changing personnel, not all of which are as familiar to the cell's operation as may be desirable. The operator is then asked to enter the desired *new* information identifying this error and how to acquire it with appropriate sensors, as well as any actions which certainly need to be carried out. Immediately afterwards, operators will still be available to check the validity of the error recovery entry so that the repair of the assembly sequence and its entry into the recovery strategy is verified.

Much more sophisticated algorithms do already exist[15,21,50] for even more autonomous operation, but the use of these techniques has been excluded deliberately in this demonstration cell in order to concentrate on the FTM system and the Information Spaces for error detection. Those errors which were encountered during trials are now described; their experimental details are presented in Appendix G.

6.7.2 Expected Error Conditions

It appeared from the observations of the cell in Darlington (compare chapter 2) that the relative rotation of bush and motor shaft was one of the most difficult problems to tackle. Since there are no facilities on either feeding mechanism to orientate the keys, a mismatch is very likely. A new Information Space was thus created which consisted of five dimensions:

- torque exerted during rotation,
- downward force during rotation,
- translation in z,
- rotation around the alignment axes and
- gripper opening.

Upon entering this error space, the robot was demanded to rotate around the axis of alignment. The moment at which the force along that axis reduces, indicates the alignment of the key features so that the insertion can be attempted again. The whole error condition is only true as long as the assembly take place in the 'chamfer region', hence the translation must be monitored. In any practical situation, the axes of the robot's rotation and that of the alignment are not the same, hence some lateral forces will appear and possibly hinder the rotation. At the same time, the key features themselves introduce lateral displacements and the overall effect can be observed through the gripper opening. If the jaws are forced open too far through a skew component, rotation must be abandoned and the other error situation attended.

The division of the Information Space leads to an automatic adjustment of forces, independent of any friction present at the time of contact. For example, as soon as rotation is attempted, the friction and other disturbances such as the robot's own vibrations will temporarily remove the downward force. The controller now assumes that it is possible that the keys are aligned and resumes the first level of mating space. After re-checking, it then pushes down further, increasing the level of contact force. Some adjustments can be made by the operator to reduce this iteration process, but essentially, the anticipated level of force was found automatically. Whether this is correct for a given product assembly cannot be established at assembly time.

Once the level of force has reduced in this context, the placement of the bush onto the motor shaft can continue. The context changes after the chamfer region has been left and an increase in forces now indicates a different error. It is possible now that there is a jamming or wedging situation. The controller makes its decisions with the help of additional information from the forces in the x-y plane and should this exceed a preset level, then a *compliant* move is executed.

Additional sensor readings are also necessary in the context of a misalignment of axes before the chamfer region is reached. The arguments presented in the sections concerning Local Accuracy also indicate that such an error is to be expected. When this happens, the hole and the peg make contact at two points resulting in a moment around an unknown axis on the horizontal plane. Again, this information can be measured and interpreted as the misalignment situation. In this case, error recovery is implemented by taking the moment as an indication of which direction to move in order to re-align. Using purely torque information does not allow quantative adjustments and consequently, several motions need to be made for larger errors.

The amount of movement eventually performed gives an indication about the error; additional information for operators if it happens *regularly* (as defined by the production engineer) or the cell itself when in teaching mode. In any case, this knowledge is available to the operator allowing informed decisions to be made be they for expected or unexpected situations.

6.7.3 Unexpected Error Conditions

A number of errors appeared in the demonstration cell which were not expected at the time of the design. These are partially due to the circumstances of this particular cell and although they are quite clear and obvious in hindsight, it would have required a very detailed analysis of all possible situations to include them from the start.

- **Overshoot:**

  The sampling time of the overall system during the application of the Information Spaces determines the overall speed of the robot. The desire for fast operations (see section 6.8 for discussion of the cell performance) then tempts the use of too high a speed for the robot. This results in an occasional overshoot during mating operations which have no defined endstop, such as picking up a component with open grippers. The system detects the overshoot; an examination of the progress and current state then make it a simple matter to demand a reversing of the motion.

- **Hardware Fault - Loose Gripper Jaws:**

  During the picking up of the bush, the gripper opening is measured as an indicator that the correct component is in the gripper. At some point, the jaws became loose and the resulting twist left the jaws in the wrong position. This error had to be traced and rectified by the operator as any situation with such a severe malfunction could not be tackled by the controller itself. Even a re-calibration of the gripper did not give any indication. This would be the only automatic way for the system to attempt the detection of such a fault.

- **Unforeseen Configurations:**

  At first glance, the possibility of unforeseen configurations does not correspond with a careful design of the process. However, one example highlights a situation which arises again from many set up specific parameters - including the learnt error recoveries. It is difficult to see how these new situations could have been taken into account before the assembly at run time, even with the most sophisticated software tools. This emphasises the unpredictability, not the probability of actual occurrence, since the example of the axis alignment can easily happen at regular intervals.

During the two-point contact encountered at the misalignment stage, a slight tilting of the bush takes place. It happens without opening the jaws beyond the preset limit. If it is now attempted to slide the bush across the top of the motor shaft, the angle causes a greater contact force than would be acceptable under these circumstances. After the automatic detection of this *getting stuck* situation, a simple compliant motion along the axis is sufficient as recovery. Furthermore, the situation is perfectly logical after the event, of course.

- **Automatic Retry:**

This automatic retry is probably the hardest to explain but is an excellent example for cell specific circumstances. The situation arose from safety measures which were included at the early stages of implementation. In the event of unforeseen errors, the gripper would open and the robot retract from the point of potential collision danger. As an oversight, this debugging procedure was not removed from the process which picks up the bush from the feeder.

A further coincidence caused a temporary failure in the communication between one of the slaves and the master. This has since been traced as an occasional software *bug*. Due to this error, the fail-safe routine was invoked and the bush which had already been gripped by the jaws, was dropped. During a number of occurrences, the bush sometimes fell back into its original place and sometimes tumbled onto its side. At the end of this recovery, the cell controller proceeded to take another assessment of the situation to determine where it was, and found (due to the information it gathered specifically for that purpose) that it was still in the right configuration to continue with mating. If it encountered the more favourable case in which the bush was standing upright as would be expected in a normal situation, the assembly proceeded with a retry, neither because of an explicitly pre-programmed case nor with any assistance from the operator. Alternatively, when the bush had fallen over, the Information Space rejected success and an error was flagged.

There is no suggestion of deliberate action in this error recovery or that the *intelligence* in the cell was actually able to interpret the situation in the correct way. The error (and its recovery) has emerged through an accumulated set of *coincidences*, which caused the retry because the system was able to judge whether it was in the correct situation or not. It does emphasise the existence of situations which are consequences of *peculiar* circumstances, such as coincidences and mistakes by operating personnel.

These circumstances leading to errors during the assembly process were observed and evaluated to achieve recovery in principle. The demonstration cell has been virtually under constant change during the development and not all parts of the implementation performed at a desirable speed.

6.8 Cell Performance

The overall performance was reduced noticeably by two principal aspects; one concerned with undesirable methods of writing the modular software and choice of communication protocols, the other with an unavoidable system drawback which could only have been overcome with closer co-operation of the robot manufacturer.

In the first instance, some choices towards the implementation of slave and master software were made without having adequate decision criteria to hand. Mostly, such mistakes were easily seen in hindsight, but not at the time of the design. They turned out to be incorrect when the whole system started to interact. At that point, some changes were deliberately not carried out in order to be able to concentrate on the aspects of error recovery and sensor integration. The latter made it necessary to have many individual sub-systems available (compare section 6.3), and rather than spending time on improving an individual slave's performance, sub-optimal performances were tolerated.

Secondly, the force control during the mating stage was executed by relating Cartesian force values to Cartesian robot coordinates. Although this is conceptually the easiest way, it demands the inclusion of the *longest* communication route. Robots are usually not designed to incorporate general sensor information on that level. Any fast communication (that is comparable to the update rate) remains special to a robot product, such as the inclusion of commercial vision into the ADEPT robot. Hence there is an unavoidable delay which could only be removed if the communications between the robot controller and external computers could be implemented directly incorporating these demands.

As a further part of *arbitrary* decisions during the demonstration cell design, a few processes have been chosen from some alternatives which would have been equally valid to be used in the cell. For example, the bolt and bush (see figures 1.1 and 6.3) are fed by linear vibratory feeders rather than by bowl or centre hopper feeders. Also, it was much easier in the circumstances of the demonstration cell to measure the relative rotation between the bush and the motor shaft using the robot itself. It would also have been possible to turn the motor shaft until the keys were aligned (compare section 2.1, sub-section 4), just as it is carried out in the FLYMO factory. The addition of another measurement system was not justified here, as no *extra* information could have been gained.

One other reason for using the robot was the inability of the practical sampling rate to allow the determination of the correct timing to react to the forces. The motor was obviously designed to run at several thousand revolutions per minute when used to cut the grass. In the assembly, it would have to run very slowly and thus operate at a speed for which the parameters are not designed and very large variations at this part of the parameter spectrum can make measurements difficult to be repeated reliably. In the end, the solution adopted consisted of the clamping of the rotor at the opposite end, the one which was placed inside the jig.

6.9 Summary

This chapter contains the description of the demonstration cell which is designed to show the practical application of the new approaches proposed in previous chapters. The choice of product and the reasons for its suitability within a generic assembly cell are explained. The underlying hardware and software structures are described and it is shown how the ideas were finally put into practice. The results are given in form of the cell behaviour, how the generic routines are defined and how it copes with errors arising during the trial runs.

The implementation itself is a comparatively large undertaking and much of the details have been achieved in a coordinated team effort. Many ideas have evolved during the period of the research and had to be modified (sometimes) to suit the requirements. Consequently, not all system components were optimised, but the implementation of the major aspects in the systematic assembly descriptions was achieved providing foundations for error recovery and resulting in the partial product assembly. Efforts were concentrated on the most difficult aspects, leaving out those which were assumed to be *straight forward*.

Some errors appeared as they were expected, but also unexpected conditions had to be catered for. All assembly errors were detected automatically and - after teaching the cell appropriate ways to recognise them and react to them - the newly acquired expertise could be applied the next time a similar situation arose without operator assistance. The following chapter will discuss the conclusions which can be drawn from the observations in the demonstration cell.

Chapter 7

Conclusions

The system developed in the thesis is viewed critically in this chapter and its implications for various aspects are discussed, such as system complexity and how to view errors and sensing. A combination of a variety of techniques must be drawn together in an automated assembly cell in order to provide an approach which genuinely contributes towards the improvement of factory productivity. It is concluded that the principle of knowledge-based assembly exhibits a more *intelligent* behaviour when compared to conventional installations. This can be attributed in the main to its learning abilitiy. Further tasks in the automation of assembly need to be completed so that the contributions from this aspect of the assembly description and error detection can be exploited fully in complete systems.

7.1 System Complexity

In general, the easier a production machine is to use, the higher are the chances that it will be employed successfully - not only by some 'advanced' companies with a more enlightened outlook, but also by small and medium sized companies which would benefit most from the application of this technology, since they are more likely to require flexibility.

In that context, there exists an apparent contradiction in the requirements for shop floor systems. On the one hand, a great deal of human attention must be devoted to ensure a smooth operation. This certainly remains as a critical factor in managerial decisions to employ further systems of this type of technology. On the other hand, the level of competence in the computerised supervision is not yet high enough, because the technology requires also a significant amount of skill and usually detailed knowledge from the operator to maintain the desired functional level.

It is necessary to make the system so complex that it can be run confidently and by personnel with little expertise in computer programming, electronic sensing or robot kinematics and dynamics. Complicated details must be hidden within the sophistication of automation, leaving only parameters which are essential to the process visible from the outside. As long as only a few functions remain to be chosen at run time, the likelihood of mistakes introduced by people also remains small. The conceptual level of interaction should also be high, for example, such statements as *"point the robot roughly in the right direction"* are easier and much quicker to carry out by inexperienced personnel than a demand for 0.1mm placement accuracy.

As a consequence, it is unlikely that unskilled operators are able to make the correct observations or even draw the correct conclusion from the sequence of events in the cell. This is particularly important, if the cell apparently changes its behaviour according to circumstances it encounters and does not follow a very strict, predictable pattern. These reactions are, of course, the result of the cell's own *electronic* experiences, which are not easily followed or even noticed by untrained operators. Therefore, any automated cell must also be able to explain its own actions in such a way that the behaviour is described clearly and logically, linking back to the essential process parameters. Any (legitimate) interference with the operation, such as setting up new batches, maintaining hard- and software and assessing the cell's performance, must take the data accumulated during the production run as the basis for any changes.

Naturally, the clearer and more straight forward the underlying control structure is, the better are the chances of achieving these goals. Much of the work presented in this thesis has been directed towards such aims. The behaviour of the demonstration cell has proven the validity of the decomposition of the assembly process into FTM. Each stage allows the presentation of performance data to be put into its correct context. The last phase of Mating, although potentially very complex, has found a suitable and conceptually very simple representation in the Information Spaces. This has been made possible by adjusting the underlying idea about error recovery from enumerating errors (or error situations) to describing the success.

7.2 About Errors

By regarding errors as the negation of success, the view of *what to regard as an error* changes as well. It is no longer necessary to identify those situations which are obvious (may be *obvious* only to some people) and include recovery situations for these at the design time. The strict definition of *successful progress* requires that even small deviations are already regarded as errors. All these deviations are catered for within the Information Spaces and not only in terms of positional accuracy; however, more likely than not, they are usually the same.

It is possible to assign errors to a scale of severity, say 0% - 100% where, at the lower end, recovery routines are obvious and easily included. A typical example for this 'category' of errors may be a compliant move, merely reacting to force experienced during the assembly. It then becomes a little more involved at about 50% to 60% say, when only the assistance of an operator may be able to restore the production flow because of some assembly error. Up to such a level of error, the system may have accumulated enough knowledge to recover, possibly needing more time or more sensing equipment with an increase in complexity of the situation.

Further up the scale, more serious mishaps may be experienced, such as the failure of some minor equipment which needs to be replaced. At the 100% end, a catastrophic failure could be envisaged which might be just short of a complete replacement of the cell. These examples serve only as illustrations of a potential error scale, and they are not the result of detailed studies. Nevertheless, they are a valid alternative as this view of errors is not universally supported, partially due to a lack of definition in the assembly area. The systematic approach to the description of successful assembly progress supports this or a similar graduation of errors.

In this thesis, only one part of the overall assembly process has featured in the FTM and Information Space methodology: error **detection**. No attempt has yet been made to link this first step to the two remaining areas of **identification** and finally the synthesis of automatic **recovery** strategies. The potential for research into an integrated system is considerable, and the choice of this particular aspect has proven to be worthwhile because it allows for a much clearer view of the essential conditions involved.

Furthermore, much effort has been invested worldwide to find solutions for truly autonomous control structures which can find new ways of recovering by themselves without the aid of an operator. This task has proven to be substantial and no single solution (or even a few contenders) has emerged which addresses industrial needs comprehensively and which could hence be attractive enough to be employed on the factory floor.

During the evaluation of this work, the necessity for completely independent cell controllers has not emerged, although the idea of *untouched by human hand* is usually promoted in this way. In fact, just a limited number of simple and well proven techniques from AI are already useful enough to go a very long way to make the system appear to exhibit an intelligent behaviour. The existence of a simple look-up table for each discovered error sub-space provides solutions for many problems, even if they are just remembered actions which had to be taught explicitly by an operator. Some more advanced techniques such as regarding the error sub-space as the root for a tree of recovery situations will enable to find *a* more sophisticated identification of the solution. Sometimes, it may even be possible to provide the *best* solution for the circumstances, but in the end, this thesis' method does not contain explicit algorithms to generate new and original strategies automatically.

Basically, economic constraints will eventually impose a limit on the amount of intelligence which can be provided by computer technology. This demands the availability of a *human back-up system* in practical implementations, even if one person can supervise many different cells and does not need to reserve his attention for a single one. The FTM division and the Information Spaces promise to provide good tools to enable the first step into the direction of *trouble-free* automatic assembly. Emerging results from other research can be used to extend the practical applicability until the overall system has reached a level of sophistication acceptable to many companies.

Furthermore, it is also possible to interpret the behaviour of the assembly cell within the Information Spaces in such a way that certain trends may be detected at a stage early enough to prevent serious damage or loss of production. *Abnormalities* in each trace in any one of the sub-spaces and the repetition of some error conditions may well be symptomatic of general wear or deterioration in quality of components. Although the interpretation itself is much more the responsibility of an error identification mechanism, without a suitable description of the problem domain such an attempt could not be undertaken. Thus this description of the assembly process also holds the key to an *early warning system*.

7.3 Sensor Need

Of course, no error conditions or trends could ever be expected to be detected if it was not for the existence of sensing systems and their correct choice. The availability of well defined criteria to decide on suitable of sensors plays an important role, and although a method for sensor choice in itself was not developed within the frame work of this thesis, the knowledge about the desired information forms a foundation for such a development. Whether the sensor has been made for dedicated automation and is commercially available, or whether it is the result of advanced research, its merit and contribution towards the intended assembly process can be decided upon during the fusion stage of the sensor information in the driver software.

Choosing sensors on the basis of detecting successful states has one disadvantage. It may well be possible to have multiple distinct error causes whose symptoms direct the Information Space trace into the same error sub-space. If the two causes cannot be distinguished with the attached set of sensors than there exists no method of deciding on the correct action to recover from it. However, if the hardware structure allows for an *easy* integration of additional equipment, then it is possible to enhance the detection and identification capability of the cell controller as required. This assumes, of course, that the operator himself is able to differentiate error causes in the first instant and can thus assign an appropriate sensor towards the detection. Re-arrangement of sensing equipment may not have to be done at run time, but can be the result of a regular maintenance plan which incorporates the data provided by the system itself.

The decisions of the cell controller are based on generic mating descriptions, and these in turn are defined in terms of the information content and context. It is thus possible to choose the correct sensors, as these provide the physical realisation of *gathering information*. In the end, the classification criteria are based on sensing, but generality is preserved in the Information Spaces so that the dependence on one sensor is avoided. New systems can be accommodated without changing the control structure in the implementation.

7.4 Intelligent (?!) Assembly

A closer inspection of the methodology suggested in this thesis reveals that there is scope for application not only for robotic but all assembly situations, whether they are dedicated or manual. The need for a system is greatest in very flexible automation because its advantages are also most apparent in that area, whereas other well established techniques and the wealth of experience in dedicated and manual assembly may not make the introduction of *another* new method appear necessary.

The way in which it is used in specific instances can be adapted to the needs at the time. By analysing the assembly process at a very basic level, no alteration of the underlying system is necessary but several implementations of the same FTM and Information Space may exist. Hence the actual demonstration cell (see chapter 6) is not intended to suggest to be the only way in which this methodology can be realised, but it serves as an example for a *real* cell, with its many different deficiencies. In that respect, set-up specific problems as well systematic problems have been highlighted during the trial assemblies and the validity of adopting this approach has been shown.

A basis for *intelligent assembly in flexible automation* has been provided in this thesis. The ability to make context dependent decisions and learn from its own mistakes give it the appearance of some degree of intelligence. The strict definition of the term *intelligence* is left to researchers in the AI field, but in production systems, the ability to carry out assembly operations with little interruption of the production flow, a minimum amount of supervision and as little specialised equipment as possible suffices to make the choice of this system with *limited intelligence* acceptable to industrial users.

The discussions in this chapter have highlighted the respective advantages and disadvantages of the proposed system. One of its major shortcomings could be the apparent lack of integration into many aspects of well accepted methods in assembly. Many of these lines of argument need further research and the *gaps* as well as the *hooks* into the overall system are the subject of the last chapter.

Chapter 8

Expansion of FTM Methodology

The principle of using the methods developed in this thesis has been shown to be valid in the implementation of the demonstration cell. Although this description of assembly processes has addressed a very important part within the overall assembly control, it does not claim to provide all answers in the complete automation process. Some research topics have already been addressed in other projects, such as the very important area of design for assembly, for which, for example, the FTM method leaves a number of *hooks* open. Such links can be used in further research and are, indeed, necessary to complete the system.

The major topics concerned with expansion of FTM and the Information Spaces are all directed towards a further increase in cell controller's autonomy. A large amount of data is accumulated during each assembly run and can be used for analysis at later stages, provided it can be presented in a suitable format. The importance of a suitable man-machine interface must not be underestimated, as an easy interaction with the cell significantly reduces demands on the operator. Information about scheduling at run time is contained implicitly in the FTM description so that independent processes can be carried out in parallel. A suitable method for choosing the correct sensor must also be found in order to avoid a re-emergence of the dependence on *good ideas*. After the classification criteria have been established in this work, a general definition library should be built up. This task is closely linked to the general interface to CAD, a necessary part in reducing down-time at batch change-overs.

8.1 Interpretation of Run Time Data

The systematic description of the assembly process needs almost constant updating from its sensors at a suitable rate to judge the correctness of the progress. This generates a large amount of data which can be used to interpret the behaviour and performance of the cell.

8.1.1 Man-Machine Interfaces (MMI)

The accumulated information is very valuable to the management which is then able to make decisions about the viability of the cell in this configuration. Assuming that no procedures change constantly in the cell, the evaluation is likely to happen at certain intervals, either at the end of batches or regular maintenance periods. Equally important is the ability of operators to identify the *meaning* of raw data in order to restore production flow in the case of an error. If operator interaction is enabled at run time, time wasted during the rectification period is an important factor. If the information is already put into an easily comprehensible form, supervision becomes less time consuming.

A certain amount of objectivity in interpretation is also preserved, at least as much as the designer of the MMI allows it to be. To a large degree, this prevents the unwitting distortion of the cell's true performance by operators, either through ignorance or malice. Operators frequently interfere with the process to keep up an appearance of smooth running, since frequent stoppages are often seen to reflect on the quality of their work. Additionally, operators are often unskilled and it could be difficult for them to interpret a given situation from the raw data in the absence of detailed knowledge. Pre-interpretation thus gives guides and aids to point them to the correct error diagnosis and may suggest possible recovery strategies.

8.1.2 Increase of Intelligence

In addition to the MMI, there is also room for a substantial increase in the apparent intelligence of the cell. Certain criteria (such as rules about judging the cell performance) can be applied which could incorporate the cell's own experience and allow the cell controller to make decisions by itself. For example, although the definition of success sub-spaces is likely to be obtained from a library, setting sub-space boundaries may well be left to an educated guess by cell designers, thus remaining essentially arbitrary. This does not necessarily diminish their skills, it may simply be that the information is not available at the time of the design. The most obvious example is found in friction coefficients which could have significant variations.

When including a teaching phase, these coefficients can be determined or verified on-line, but they can only be regarded as guidelines, because of the arbitrary and low sample of component with which the teaching is carried out. As the number of successful assembly runs increases, so does the *experience* of the cell and it is possible to judge the initial values on statistical grounds. For example, if the boundaries had been chosen too large, then the controller itself can reduce the sub-space and still remain in the area of success. On the other hand, a frequent encounter of specific recovery routines - such as *'push a bit harder'* - could be indicating that the sub-space has been chosen too small. Suitable and safe criteria for expanding sub-spaces are not obvious at this stage and need further investigation.

However, the way error diagnosis is handled can influence the increase in autonomy significantly. The ability to recall the cell's experience at times of error detection can take many other factors into account. For example, each recovery routine will have some cost (in terms of overall production costs) associated with it and when choosing alternatives, these must be consulted. It may well be the case that some more frequent error causes need a comparatively time consuming sensor procedure to be identified, whereas a *quick* check with binary sensing can take some less frequent errors into account. In appropriate *costing arrangements*, the controller could decide on the relative merits and find it cheaper to investigate less frequent errors first.

Using another side of the same argument, the actions needed for recovery can also be *cheap* or *expensive* and their cost can be incorporated into the recovery synthesis. The degree of autonomy can be adjusted for different implementations, in so far as the operator could be asked for permission to incorporate any change or, alternatively, the responsibility can be split according to confidence in the system's ability. In a complementary way, *improvements* suggested by operators can be checked against existing production rules of which he is either not aware of or which he might have forgotten.

8.2 Autonomous Action Planning

Cost factors are always high on the list of priorities when assessing the cell performance. Productivity can be improved quite often by speeding up the through-put and reducing the cycle time. The current description of the assembly process is only in a serial format, concerned only with the success of the process. Naturally, the independence of various devices in the cell lends itself to a truly parallel execution of the individual sub-processes with the aim of minimising the overall assembly time and cost factors. Much information about parallel processes is contained within the FTM and Information Space description and ready to be looked at in a systematic way.

This area is not intended plan the assembly sequence, but rather examining the serial FTM description and extract those processes which are based on independent hardware. For example, while the robot is busy in a mating phase, the feeding of the next component with a bowl feeder can already be accomplished. Much of this *planning ahead* is usually taken for granted, but it is important - again - to introduce a systematic approach to be able to optimise the cell productivity with adequate reproducibility.

8.3 Sensor Choice Methodology

As mentioned previously, a methodical approach is just as necessary for the choice of sensors in the cell. Those Information Spaces used for all mating processes define a potential set of sensors which is unlikely to be the best overall combination. At the same time, error recovery routines give further evidence towards the choice of sensing, since the additional sensing requirements for error identification may not have been covered already.

Although the majority of errors are expected to be set-up specific, a potential connection between generic assembly and *generic* recovery actions also exists. Accumulated experience from the cell must thus not be ignored during the design of the next cell (or even product). Recent work[92] has suggested an algorithmic alternative to rule-based systems for sensor choice. However, it is not necessary to exclude one method at the expense of another and a combination may be the best to provide automatic generation of the required sensor systems.

8.4 Bridging the Gap to Design for Assembly

In fact, as many aspects as possible should be generated automatically in order to maximise the advantages of CIM. This includes the FTM description and Information Spaces as well. Whereas the latter is a case of finding the most suitable entry in a library, FTM sequences have to be specified again for each new product.

Two lines of research are envisaged in this field. Firstly, a systematic expansion of the generic library, taking into account the work which already exists in related classification systems. A thorough testing on examples provided by industry has to follow so that the limits of the validity of this methodology can be established more clearly. This may include a look at materials which have different properties from conventional, solid engineering-type components. This must be closely linked to one or more of the other aspects of research (such as sensor choice) so that there is no loss of generality.

Secondly, the overall planning and execution of the assembly must be incorporated. Much work has been done in this field throughout the world, but the availability of a description for the control of assembly processes at run time provides a specification for the *output* of such programs. Generally, the input to CAD systems is defined quite clearly as the description of the product and its components, where in more sophisticated systems the information can be extracted even from engineering drawings. Criteria for the expected degree of difficulty or an estimate of costs could be derived from an early consideration of the generic mating procedures.

The final aim of the integration of this new methodology into the complete process is an increased confidence of production managers in the smooth running of the cell. It removes the necessity to specify every little detail as must be done with current systems. Automation of assembly, particularly of flexible batch assembly, is a large and complex field to which the FTM methodology and Information Space framework have made their contribution.

References

[1] Hill, J.J., Burgess, D.C., Pugh, A., *"The Vision-Guided Assembly of High-Power Semiconductor Diodes"*, Proceedings of 14th ISIR, Gothenburg, Sweden, October 2-4, 1984, pp 449-459

[2] Leete, M.W., *"Integration of Robot Operations at Flymo - A Case Study"*, Proc. of 6th BRA Annual Conference, May 16-19, 1983, Birmingham, UK,

[3] Selke, K. and Pugh, A., *"Sensor-Guided Generic Assembly"*, Proceedings of the 6th International Conerence on Robot Vision and Sensory Controls, Paris, France, 1986, pp. 11-20. pp 113-124

[4] Cronshaw, A.J., Heginbotham, W.B., Pugh,A., *"A Practical Vision System For Use With Bowlfeeders"*, Proc. of 1st International Conference on Assembly Automation, March 1980, pp 265-274

[5] Swift, K.G., Dewhirst, R.J., *"A Laser Electro-Optic Device For the Orientation of Mass Produced Components"*, Optics and Lasers in Engineering, 4, 1983, pp 203-215

[6] Browne, A., *"A Trainable Component Feeder"*, Proc. of 5th International Conference on Assembly Automation, May 1984, pp 85-93

[7] Redford, A.H., Lo, E.K., Kileen, P.J., *"Parts Feeder for a Multi-Arm Assembly Robot"*, Proceedings of 15th CIRP International Seminar on Manufacturing Systems Assembly Automation, June 1983, Massachusetts, pp 118-125

[8] Nevins, J.L., Whitney, D.E., *"Assembly Research"*, Automatica, Volume 16, No 6, 1980, pp 595-613

[9] Lozano-Pérez, T., *"Robot Programming"*, Proceedings of IEEE, Volume 71, No 7, July 1983, pp 821-841

[10] Williams, A.M., Walters, P.E., Ashton, M., Reay, D., "*A Flexible Assembly Cell*", 6th International Conference on Assembly Automation, May 14-16, 1985, Birmingham, UK, pp 57-66

[11] Groover, M.P., "*CAD/CAM: Computer-Aided Design and Manufacturing*", Chapter 12, Publ. Prentice/Hall International, 1984

[12] Selke, K., Swift, K.G., Taylor, G.E., Pugh, A., Davey, S.N., and Deacon, G.E., "*Knowledge-Based Robotic Assembly - A Step Further Towards Flexibility*", Computer Aided Engineering Journal, 1987, Vol. 4, No. 1, pp. 62-67.

[13] Owen,T., "*Assembly With Robots*", New Technology Modular Series, Kogan Page, 1985

[14] Lee, M.H., Hardy, N.W. and Barnes, D.P., "*Error Recovery in Robot Applications*", 6th British Robotics Association Annual Conference, Birmingham, England, 1983, pp. 217-222.

[15] Lee, M.H., Barnes, D.P., and Hardy, N.W., "*Knowledge Based Error Recovery in Industrial Robots*", Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany, 1983, Vol. 2, pp. 824-826.

[16] Hardy, N.W., *Interim Report on Generic Fault Analysis*, Technical Memo. 16/83, University College Wales, Aberystwyth, 1983.

[17] Barnes, D.P., Lee, M.H., and Hardy, N.W., "*A Control and Monitoring System for Multiple Sensor Industrial Robots*", Proceedings of the 3rd International Conference on Robot Vision and Sensory Controls, Boston, U.S.A., November 1983, pp. 471-479.

[18] Hardy, N.W., Lee, M.H., and Barnes, D.P., *"Knowledge Engineering in Robot Control"*, Proceedings of the 3rd Expert Systems Conference of the British Computer Society, Cambridge, England, December 1983, pp. 70-77.

[19] Lee, M.H., Hardy, N.W., and Barnes, D.P., *"Research into Automatic Error Recovery"*, Proceedings of the Conference on UK Robotics Research, London, England, December 1984, pp. 65-69.

[20] Lee, M.H., Barnes, D.P., and Hardy, N.W., *"Research into Error Recovery for Sensory Robotics"*, Sensor Review, October 1985, pp. 194- 197.

[21] Gini, G., Gini, M., and Salmavico, M., *"Emergency Recovery in Intelligent Robots"*, Proceedings of the 5th International Symposium on Industrial Robotics, Chicago, U.S.A., 1975, pp. 339-358.

[22] Gini, G., Gini, M., and Solmavico, M., *"Program Abstraction and Error Correction in Intelligent Robots"*, Proceedings of the 10th International Symposium on Industrial Robotics, Milan, Italy, 1980, pp. 101-108.

[23] Gini, M. and Gini, G., *"Recovering from Failures: A New Challenge for Industrial Robots"*, in "Robotics and Artificial Intelligence", Brady, M., Gerhardt, and L.A., Davidson, H.F. (Ed.s), NATO ASI Series, Series F: Computer and Systems Sciences Vol. 11, Spinger-Verlag, Berlin, Heidelberg, 1984.

[24] Gini, M., and Gini, G., *"Towards Automatic Error Recovery in Robot Programs"*, Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany, 1983, Vol. 2, pp. 821- 823.

[25] Smith, R.E. and Gini, M., *"Robot Tracking and Control Issues in an Intelligent Error Recovery System"*, Proceedings of the International Conference on Robotics and Automation, San Fransisco, U.S.A., 1986, Vol. 2, pp. 1070-1075.

[26] Paul, R.P. *"Robot Manipulators: Mathematics, Programming and Control"*, MIT Press, 1981

[27] Swift, K.G., *"A System for Classification of Automated Assembly"*, M.Sc. Thesis, University of Salford, Salford, England, 1980.

[28] Sanderson, A.C., *"Parts Entropy Methods for Robotic Assembly System Design"*, Proceedings of the International Conference on Robotics, Atlanta, U.S.A., 1984, IEEE, pp 600-608

[29] Rathmill, K. (Ed.), *"Robotic Assembly"*, IFS Publications Uk Ltd, Springer Verlag, 1985

[30] Taylor, P.M., Stubbings, C.A., *"Software and Hardware Aspects of a Flexible Workstation for Assembly Tasks Using Sensory Controls"*, 2nd IASTED International Symposium on Robotics and Automation, Lugano 1983

[31] Popplestone, R.J., Ambler, A.P., Bellos, I., *"RAPT: A Language For Describing Assemblies"*, The Industrial Robot, September 1978, p 13

[32] Johnson, D.G., Hill, J.J., *"A Calman Filter Approach to Sensor-Based Robot Control"*, IEEE Journal of Robotics and Automation, Vol RA-1, No 3, September 1985, pp 159-162

[33] Roussel, P., *"PROLOG: Manuel de Reference et d'Utilisation"*, Universite AixMarseilles, Luminy 1975

[34] Rueher, M., Thomas, M.-C., Gubert, A., Ladret, D., *"A PROLOG Based Graphical Appraoch For Task Level Specification"*, NATO ASI Series, Vol. F29, Languages for Sensor-Based Control in Robotics, Edited by U. Rembold and K. Hörmann, Springer Verlag, 1987

[35] Gini, M., *"Symbolic and Qualitative Reasoning for Error Recovery in Robot Programs"*, NATO ASI Series, Vol. F29, Languages for Sensor-Based Robotics, Edited by U. Rembold and K. Hoermann, Springer Verlag, 1987, pp 147-167

[36] Blume, C., Jakob, W., *"Programming Languages for Industrial Robots"*, Springer Series Symbolic Computation - Artificial Intelligence, Springer Verlag 1986

[37] Mcloy, D., Harris, M., *"Robotics: An Introduction"*, Edited by P. G. Davey, OU Press Robotic Series, 1986

[38] Caprioglio, L., *"The Deltix Spot Welding Robot in the Automotive Industry"*, Proceedings of the 4th ISIR, Tokyo, Japan, 19-21 November 1974, pp 507-518

[39] Laerdahl, S., *"Trallfa Robot - A New Concept of the Control System, Paint Spraying and Arc Welding Applications"*, Proceedings of the 3rd Industrial Robot Technology and 6th ISIR, University of Nottingham, UK, 24-26 March 1976, pp G3/17-G3/28

[40] Boothroyd, G., Redford, A., *"Mechanized Assembly"*, McGraw-Hill, 1968

[41] Boothroyd, G., Dewhurst, P., *"Design for Assembly Handbook"*, University of Massachusetts, 1983

[42] Boothroyd, G., *"Economics of General Purpose Assembly Robots"*, Proceedings of the 15th CIRP International Seminar Manufacturing Systems, Amherst, Massachusetts, 1983

[43] Boothroyd, G., *"Use of Robotics in Assembly Automation"*, CIRP General Assembly, Madison, Wisconsin, 1984

[44] Koutsou, Anastasia, *"A Survey of Model-Based Robot Programming Languages"*, Department of Artificial Intelligence, University of Edinburgh, Working Paper No. 108, December 1981, Draft 2, 79 pages

[45] Lozano-P§rez, T., *"Robot Programming"*, Proceeding of the IEEE, Volume 71, No. 7, July 1983, pp 821-841

[46] Redford, A., Lo, E., *"Robots in Assembly"*, Open University Press Robotics Series, 1986

[47] Finkel, R., Taylor, R., Bolles, R., Paul, R., Feldman, J., *"An Overview of AL, A Programming System for Automation"*, Proceedings of the 4th International Joint Conference on Artificial Intelligence, Georgia, USSR, 1975, pp 758-765

[48] Lieberman, L. I., Wesley, M. A., *"AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly"*, IBM Journal of Research and Development, Volume 21, No. 4, 1977, pp 321-333

[49] Latombe, J.-C., Mazer, E., *"LM: A High-Level Programming Language for Controlling Assembly Robots"*, Proceedings of 11th International Symposium on Industrial Robots, 7-9 October 1981, Tokyo, Japan, pp 683-690

[50] Winston, P. H., *"Artificial Intelligence"*, Second Edition, Addison-Wesley Publishing Company, 1984

[51] Heginbotham, W. B., Page, C. J., Pugh, A., *"Robot Research at the Unversity of Nottingham"*, Proceedings of the 4th ISIR, Tokyo, Japan, 19-21 November 1974, pp 53-64

[52] Taylor, R. H., Hollis, R. L., Lavin, M. A., *"Precise Manipulation with Endpoint Sensing"*, IBM Journal of Research and Development, Volume 29, No. 4, July 1985, pp 363-376

[53] Wilson, W. J., *"Vision Sensor Integration for Direct Manipulator End-Point Control"*, Proceedings of the Nato Workshop in Il Ciocco, Italy, October 1987, 19 pages

[54] Wirth, N., *"Programming in MODULA-2"*, Third Corrected Edition, Edited by David Gries, Springer Verlag, 1985

[55] Messer, P., Marshall., I., *"Modula-2: Constructive Program Development"*, Computer Science Texts, Blackwell Scientific Publications, 1986

[56] Laszcz, J. F., *"Product Design for Robotic and Automatic Assembly"*, Proceedings of ROBOTS8 Conference, SME Technical Paper MS88-362, Detroit, USA, 4-7 June 1984

[57] Schraft, R. D., Bässler, R., *"Considerations for Assembly Oriented Product Design"*, 5th International Conference on Assembly Automation, Paris, 22-24 May 1984

[58] Swift, K. G., *"A Computer-Based Design Consultation System"*, Assembly Automation, 3(3), 1983, pp 151-154

[59] Craig, J. J., *"Introduction to Robotics, Mechanics and Control"*, Addison Wesley Publishing Company, 1986

[60] Hollerbach, *"A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity"*, in *Robot Motion*, M. Brady (Ed.), MIT Press 1983

[61] McCallion, H., *"A Compliance Device for Inserting a peg Into a Hole"*, The Industrial Robot, Volume 6, No 2, 1979

[62] Drake, S. H., *"Using Compliance in Assembly"*, Proceedings of Autofact West, Michigan, USA, Vol 2, 1980

[63] Raibert, M., Craig, J. J., *"Hybrid Force/Position Control of Manipulators"*, ASME Journal of Dynamic Systems, Measurement and Control, June 1981

[64] Rueb, K. D., Wong, A. C., *"Visual Identification and control in a Robot Workcell"*, Systems Design Engineering, University of Waterloo, Canada, September 15, 1987, 24 pages

[65] Taylor, P. M., Selke, K. K. W., Pugh, A., *"Visual Feedback Applied to the Assembly of Thyristors"*, Proceedings of the Congrès AFCET Automatique 1983, Productique et Robotique Intelligente, Besançon, November 1983

[66] Whitehead, D. G., Mitchel, I., Mellor, P. V., *"A Dynamic RAM Camera"*, IEE Colloqium on Developments in Robot Sensors, Digest No 1984/18, February 1984, 5 pages

[67] Whitehead, D. G., Mitchel, I., Mellor, P. V., *"A Low Resolution Vision Sensor"*, Phys. E. Sci. Instrum., Vol 17, August 1984, pp 653-656

[68] Kemp, D. R., Taylor, G. E., Taylor, P. M., *"An Adaptive Sensory Gripper for Fabric Handling"*, Proceedings of IASTED, Fourth International Conference on Robotics and Automation, Amsterdam, September 1984, 4 pages

[69] Johnson, D. G., Hill, J. J., *"A Modular Linear Camera for Robot Vision"*, Journal of Digital Systems for Industrial Automation, Vol. 2, No. 3, 1984

[70] Barry Wright Corporation, *"FS6-120A 6-Axis Force Sensor"*, Software Manual, Watertown, Massachusettes, 1982

[71] Barry Wright Corporation, *"FS6-120A 6-Axis Force Sensor"*, Hardware Manual, Watertown, Massachusettes, 1982

[72] Stubbings, C. A., *"ROBUS - A Cheap Multi-Processor Robot and Sensor Control Bus"*, Technical Report, Department of Electronic Engineering, University of Hull, May 1983

[73] Adept Technology Inc., *"VAL II Reference Guide"*, Version 5.0, Revision C, June 1986, Adept Technology Inc., 1212 Bordeaux Drive, Sunnyvale, CA 94089

[74] Fairey Automation, *"Flexible Assembly at Perkins"*, Flexible Automation News, April 1985, National Exhibition Centre Brimingham, England, 14-17 May 1985

[75] Hine, D., *"Automatic Engine Assembly and Test"*, Proceedings of the 9th International Conference on Assembly Automation, 15-17 March 1988, London, pp 29-40

[76] Mointe, M., *"Practical Example of a Computer Integrated Assembly System Built Step By Step"*, Proceedings of the 9th International Conference on Assembly Automation, 15-17 March 1988, London, pp 109-116

[77] Morgan, C. G., Bromley, J. S. E., Davey, P. G., Vidler, A. R., *"Visual Guidance Techniques for Robot Arc Welding"*, in *Robot Sensors*, Volume II, Tactile and Non-Vision, Alan Pugh (Ed.), IFS (Publications), 1986

[78] Digital Equipment Corporation, *"Device Drivers for VAX/VMS"*, Version 4.4, Volume 10C, System Documentation, Appendix H, *Mapping I/O Space and Connecting to an Interrupt Vector*

[79] Davey, S. N., Selke, K., *"Bus Structured DCN: A Modular System for Robotic Control and Multi-Sensor Fusion"*, Proceedings of International Conference on Factory 2000: Intergrating Information and Material Flow, Churchill College Cambridge, 31st August - 2nd September 1988, pp 289-294

[80] LOGITECH, *"M2 VMS Logitech Modula-2"*, Compiler and Library for VAX/VMS, User's Manual, Reference LS-UD-003-02, Initial Issue: April 1983, Current Issue: March 1987

[81] Taylor, P. M., Selke, K. K. W., Taylor, G. E., *"Closed Loop Control of an Industrial Robot Using Visual Feedback from a Sensory Gripper"*, Proceedings of 12th ISIR and 6th International Conference on Industrial Robot Technology, June 9-11, 1982, Paris, France, pp 79-86

[82] Kuntze, H. B., *"Methods for Collision Avoidance in Computer Controlled Industrial Robots"*, Proceedings of 12th ISIR and 6th International Conference on Industrial Robot Technology, June 9-11, 1982, Paris, France, pp 519-530

[83] Taylor, P.M., Taylor, G. E., Kemp, D. R., Seim, J., *"Software and Harware Aspects of Sensory Gripping Systems"*, Proceedings of 11th ISIR, October 7-9, 1981, Tokyo, Japan

[84] British Standard Institution, *"Token Bus Local Area Networks"*, Draft for Development, Part I, Technical Specification (ISO DP 8802/4; IEEE 802.4, Draft F)

[85] British Standard Institution, *"Guidance for Implementors"*, Draft for Development, Part II, Technical Specification (ISO DP 8802/4; IEEE 802.4, Draft F)

[86] IEEE Standards for Local Area Networks, *"Logical Link Control"*, ANSI/IEEE Standard 802.2-1985, ISO Draft, International Standard 8802/2

[87] IEEE Standards for Local Area Networks, *"Carrier Sense Multiple Access with Collision Detection"*, ANSI/IEEE Standard 802.3-1985, ISO Draft, International Standard 8802/3

[88] IEEE Standards for Local Area Networks, *"Token-Passing Bus Access Method"*, ANSI/IEEE Standard 802.4-1985, ISO Draft, International Standard 8802/4

[89] IEEE Standards for Local Area Networks, *"Token Ring Access Methods"*, ANSI/IEEE Standard 802.5-1985, ISO Draft, International Standard 8802/5

[90] Boothroyd, G., Dewhurst, P., *"Product Design for Assembly"*, Published by Boothroyd and Dewhurst Inc, 2 Holly Road, Wakefiel, R1 02879

[91] Boothroyd, G., Dewhurst, P., *"Design for Assembly Software Toolkit"*, Published by Boothroyd and Dewhurst Inc, 2 Holly Road, Wakefiel, RI 02879

[92] Taylor, P. M., Pugh, A., *"Sensor Guided Assembly"*, Proceedings of IFAC 1987, 10th Congress on Automatic Control, Munich, July 1987

Appendix A

ROBUS Specifications

Version 1.1

The specifications for ROBUS have first been developed within the Robotics Research Unit at the Unversity of Hull during 1983. Much of the initial detailed work was carried out by C. A. Stubbings as part of his research into LASCAR[30], a language and software support for assembly cells. The documentation in this appendix is based on the provisional Issue 1 and some ideas have been confirmed in practical use and omissions rectified. The specifications are helpful in more detailed understanding of the *ROBUS philosophy.*

A.1 Introduction

This appendix contains the reasoning behind ROBUS, the specification of ROBUS in Version 1.1. During the research into sensor-guided robotic systems at the Unversity of Hull a significant amount of time was spent in interfacing robots and peripherals. Individual members in the group required the same type of interface for either the same or very similar units and hence tended to duplicate each others work.

For these reasons, it was decided to adopt an in-house *standard* robot interface. This *robot interface* was required to be capable of supporting any peripherals or robots which were likely to be encountered during future research.

A.2 Physical Construction

The interface is based on a 64-way indirect edge connector (RS 467-469/467-475) and double height eurocards in a 6U rack. In this rack it is also possible to add an additional connector to provide a high speed vision bus. This second, optional bus is a 96-way indirect connector (RS 467-914/467-920). Version 1.1 does not define the *vision* bus.

The main (64-way) bus is the lower of the two connectors in the rack. Since the upper connector is optional, ROBUS can be implemented on single height eurocards in a 3U rack. See section A.6 for the connector pinouts.

A.3 Bus Signals

ROBUS has the following signals on the lower connector:

- 8 primary address lines,
- 8 secondary (or register) address lines,
- 8 bi-directional data lines,
- 7 control lines,
- 9 interrupt lines.

A.3.1 Primary/Secondary Address

Each device on the bus has a unique primary address. Two addresses - 0 and 255 - are defined as invalid. Address 0 is excluded for ease of handling purposes in software and address 255 actually represents a slave not present on the bus. Alternatively, a slave with local intelligence can make itself invisible to the master. The reason for this lies in the possibility to perform transparent communication monitoring on the bus.

In addition, there are up to 256 secondary addresses. These enable the slave to have more than one byte direct communication, such as may be the case for very fast data transfer or even for slaves without any intelligence at all.

A.3.2 Data Lines

The eight data lines are bi-directional and used to exchange data between the master and the slaves. The direction is controlled by the master.

A.3.3 Control Lines

The remaining control lines are also activated by the master and obeyed by the slaves.

A.3.3.1 Read and Write (RD, WR)

The master generates active low signal on these lines to indicate the direction of the data flow.

A.3.3.2 Interrupt Acknowledge (IACK)

Active low signal generated by the master when acknowledging a slave interrupt. This should be used as a read qualifier. If the slave is not interrupting and IACK is active it should not respond to a bus read.

A.3.3.3 Slave Data Valid (SLDV)

Active low signal generated by the slave during a master read to indicate that the data being read is valid and has not been read before. This signal is required to qualify all master read operations.

If the slave does not generate this signal, the pin should be pulled to logic zero by a high-impedance buffer when controlled by the ROBUS RD line. This line should be pulled high by the master.

A.3.3.4 Overrun (OVRN)

Active low signal generated by the slave during a master write operation to indicate that it has not accepted the previous byte yet and is ignoring the one being written. Note that the previous byte must not be corrupted if an OVRN condition exists. This line should be implemented on all slaves that accept multi-byte commands if the maximum time between accepting two consecutive byte of data is greater than 10 microseconds. This line should be pulled high by the ROBUS master.

A.3.3.5 Reset (RESET)

Active low signal generated by the master to initialise all slaves. In the absence of a defined slave protocol, responses to a RESET command have to be programmed individually.

A.3.3.6 Selective Reset (SELRES)

Active low signal generated by the master to initialise an individual slave. A slave may respond to the SELRES signal only in conjunction with a valid slave address. The slave response is defined in the same manner as for a general RESET.

A.3.4 Interrupts

In order to facilitate asynchronous communication across ROBUS, two types of interrupts have been implemented so that tasks can be executed in parallel and independent of each other as much as possible.

A.3.4.1 Communication Cord (CC)

This is an emergency interrupt generated and monitored by any slave. It should be used in all cases of severe malfunction or operational error to indicate that all devices on the bus should stop their current activities immediately and wait for further instructions from the master. It is strongly recommended that this line is monitored by all slaves controlling active devices such as robots, active fixtures, x-y tables or similar. This line should be pulled high by the master.

A.3.4.2 General Interrupts (INT0 - INT7)

These are the ordinary slave-to-master interrupts. A slave should use INT0 unless the master has asked it to use one of the other lines. INT1 to INT7 are assigned to slaves by the master. The support of dynamic interrupt assignments is optional. The priority levels of interrupts are intended to be application dependent and programmed within the structure of the control program and the master's capabilities.

All interrupt lines should be pulled high by the master.

A.4 Slave Interfaces

Slave interfaces can be split into two main categories: *smart* and *dumb* slaves. The difference is found in the control electronics, that is whether a slave is controlled by an *intelligent* micro processor or by hardwired logic. In most most cases, the slave is expected to be *smart*, however, it is considered that *dumb* slaves retain their usefulness in certain applications. The use of register addressing is recommended particularly in these applications.

A.5 Protocol

The definition of the protocols presented here is not fully matured. It is suggested to regard them as proposals which have emerged as useful and important during initial trial runs with ROBUS.

A.5.1 Initialisation

The master will hold the RESET low for at least 100ms, then read each bus primary address in turn after a delay to allow slaves time to initialise themselves. The delay is variable and defined by the master to suit applications. All smart slaves should respond with an acknowledge message so that identification codes can be implemented. Dumb slaves are expected to respond with a valid SLDV signal at any time.

The initialisation message is variable in length and depends on the slave and its program. Therefore, the master will read the message and then wait after the receipt of the last byte for another application dependent time to ensure the reception of the complete message. If the SLDV signal is not asserted during the first read, it is assumed that there is no slave with that address on the bus.

A proposed device identification code consists of four bytes. The first is the device type (robot, camera, etc), the second is the interface code (which kind of robot, for example), the third is a *smartness factor* (how much information the slave has for the master about itself, 0=none) and the fourth is the software revision number. Individual codes remain to be defined.

A.5.2 Terminators

In the interest of easy programming it is recommended that slaves should terminate all data transfer to the master with a single character terminator. The character is dependent on the data formats and, for example, may be a carriage return (ASCII code 13) in the case of human readable character strings.

A.5.3 Use of Registers

Where a single piece of information is longer than eight bits, it may either be sent byte by byte to the same register or to a sequence of registers. In the latter case, it is recommended to write to consecutive registers.

A.5.4 Data Formats

It is suggested that the order and format of data sent to the slave is consistent for all commands, for example:

LSB first coded in HEX ASCII

or MSB first coded in decimal ASCII.

There are a number of common data formats and it is suggested that one of the following group is chosen and adhered to for one particular slave:

- Hexadecimal ASCII
- Octal ASCII
- Decimal ASCII
- Binary

A.6 ROBUS connector Pinout

	B-SIDE		A-SIDE	
	GND	1	GND	
	GND	2	GND	
	+12V	3	+12V	
	-12V	4	-12V	
Secondary Address Lines for Multi-Register Devices	(A1)	5	(A0)	Secondary Address Lines for Multi-Register Devices
	(A3)	6	(A2)	
	(A5)	7	(A4)	
	(A7)	8	(A6)	
Primary Address Lines	A9	9	A8	Primary Address Lines
	A11	10	A10	
	A13	11	A12	
	A15	12	A14	
		13		
		14		
		15		
		16	EXP4	To be pulled high by the controller, for future expansion
		17	EXP3	
Communication Cord	$\overline{\text{CC}}$	18	EXP2	
Device Write	$\overline{\text{WR}}$	19	EXP1	
Slave Data Valid	$\overline{\text{SLDV}}$	20	$\overline{\text{RESET}}$	General Reset
Device Read	$\overline{\text{RD}}$	21	$\overline{\text{IACK}}$	Interrupt Acknowledge
Selective Reset	$\overline{\text{SELRES}}$	22	$\overline{\text{OVRN}}$	Data Write Overrun
Bidirectional Data Lines	D1	23	D0	Bidirectional Data Lines
	D3	24	D2	
	D5	25	D4	
	D7	26	D6	
General Purpose Interrupt Request	$\overline{\text{IRQ0}}$	27	$\overline{\text{IRQ4}}$	General Purpose Interrupt Request
	$\overline{\text{IRQ1}}$	28	$\overline{\text{IRQ5}}$	
	$\overline{\text{IRQ2}}$	29	$\overline{\text{IRQ6}}$	
	$\overline{\text{IRQ3}}$	30	$\overline{\text{IRQ7}}$	
	-5V	31	-5V	
	+5V	32	+5V	

Appendix B

ROBUS To DRV11J Interface

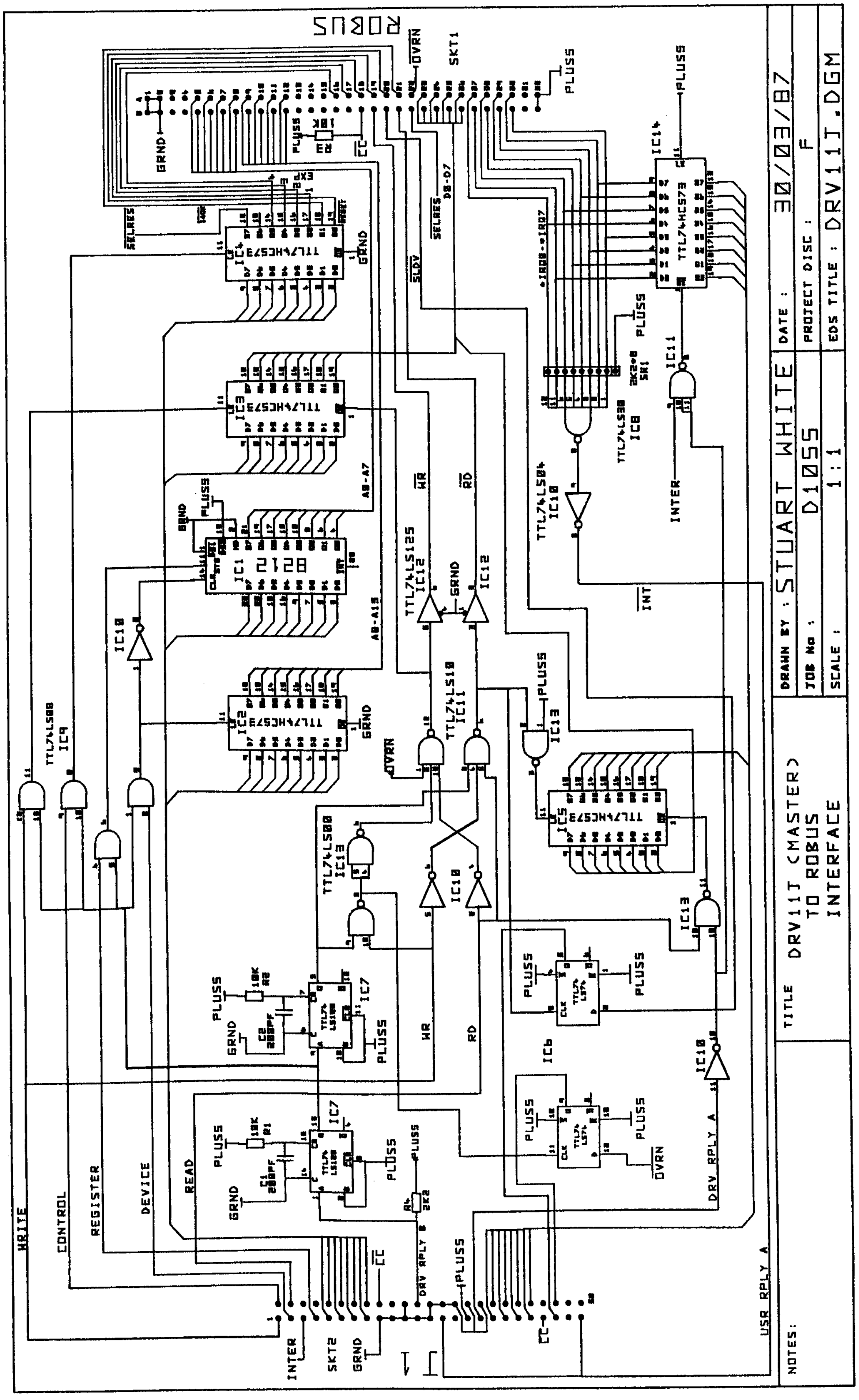
Circuit diagram

The interface diagram has been provided as a matter of completeness for the documentation and also as a cross-reference for the low level ROBUS software.

This circuit provides for asynchronous read and write operations by both master and slave. The connection into the MicroVaxII through the parallel ports simplifies the hardware design, but also necessitates the generation of bus signals via software. The data exchange has been restricted on a single byte at a time.

Interrupt facilities are provided in two ways. In version 1 of the ROBUSIO module (see Appendix C), interrupts by slaves can only be detected by polling the slave interrupt lines through a latch. The priority scheme is determined in software. Future versions of the sofware can utilise the hardware interrupt capability of the MicroVax itself.

There are no known bugs in this interface.


TITLE DRV11J (MASTER) TO ROBUS INTERFACE
DRAWN BY : STUART WHITE
DATE : 30/03/87
JOB No : D1055
PROJECT DISC : F
SCALE : 1:1
EDS TITLE : DRV11J.DGM
NOTES:
ROBUS
SKT1
SKT2
IC1 2128
IC2 TTL74HC573
IC3 TTL74HC573
IC4 TTL74HC573
IC5 TTL74HC573
IC14 TTL74HC573
IC6 TTL74 LS74
IC7 TTL74 LS123
IC8 TTL74LS30
IC9 TTL74LS08
IC10 TTL74LS04
IC11 TTL74LS10
IC12 TTL74LS125
IC13 TTL74LS00
A0-A7
A8-A15
D0-D7
WR
RD
INT
INTER
WRITE
CONTROL
REGISTER
DEVICE
READ
DRV RPLY A
DRV RPLY B
USR RPLY A
PLUS5
GRND
SELRES
SLDV
OVRN
CC
C1 200PF
C2 200PF
R1 10K
R2 10K
R3 10K
R4 2K2
2K2*8 SR1

Appendix C

ROBUSIO Definition Module

Low level Software Interface Listing

Only the DEFINITION MODULE **has been included in this appendix, since this is the most important interface for a software engineer. The** IMPLEMENTATION MODULE **also exists, but does not add to the information in the context of this thesis. The** ROBUSIO **modules have been written as general support routines for the Robotics Research Unit, and contain all routines to provide functional control of a ROBUS card cage.**

```
DEFINITION MODULE RobusIO;

(* The procedures and data structures necessary for ROBUS communications
   are defined in here

        ***********************************

        K. Selke
        Robotics Research Unit,
        University of Hull,
        Department of Electronic Engineering,
        Hull, HU6 7RX

        ***********************************

        Version 0:      7 November 1986

        Version 1:      16 June 1987
        No interrupt facility available yet, but the basic ROBUS
        servicing routines are working and tested to a large degree.
        If there are any bugs, please let KS know as soon as possible
        for corrective actions.
*)

FROM SYSTEM IMPORT
        BYTE, SHORTWORD;
```

```
EXPORT  QUALIFIED

        (x CONSTANTS x)
        ReadBit, WriteBit, RegisterBit, DeviceBit, ControlBit,

        CommCordBit, InterruptBit, OverrunBit, SlaveDataVldBit,

        ResetBit, IackBit, Exp1, Exp2, Exp3, Exp4, SelResBit,

        NullSlave, HighestSlaveAddress, LowestSlaveAddress,
        Register0, Register255,

        (x TYPES x)
        BYTESET, SHORTSET, SlaveType, SlaveRegister,

        SlavePriorityStatusSet, SlavePriorityStatus,
        SlavePriorityRecord, SlavePriorityListType,

        SlaveStatus, SlaveStatusSet, SlaveBufferType, SlaveInputBuffer,

        RobusIOResult, CommunicationMode,

        (x Common VARIABLES x)

        SlavePriorityList,    (x set up during InitialiseRobus x)
        RetryCount,           (x Changing this variables gives applications
                                 the freedom to decide for themselves.
                                 Initial value: 100 x)

        (x PROCEDURES x)

        GetCurrentRobusMode, TransparentTalk,
        SetRobusDeviceAddress, SetSlaveRegister, SetRobusControlLines,
        ClearRobusControlLines, ResetSelectedSlave,
        WriteRobusRawData, ReadRobusRawData,

        RobusRead, RobusWrite, RobusDirectRead, RobusDirectWrite,
        InitialiseROBUS;

CONST

        ReadBit              = 8;        (x Output control bits x)
        WriteBit             = 9;
        RegisterBit          = 10;
        DeviceBit            = 11;
        ControlBit           = 12;
        OverrunBit            = 14;      (* Input status bits     *)
        SlaveDataVldBit       = 15;
        CommCordBit           = 13;
        InterruptBit          = 12;

        ResetBit              = 0;       (* Control status bits *)
        IackBit               = 1;
        Expl                  = 2;
        Exp2                  = 3;
        Exp3                  = 4;
        Exp4                  = 5;
        SelResBit             = 7;

        NullSlave             = 0;
        LowestSlaveAddress    = 1;
        HighestSlaveAddress   = 254;

        Register0             = 0;
        Register255           = 255;

TYPE
        BYTESET               = SET OF [0..7];
        SHORTSET              = SET OF [0..15];

        SlaveType             = [NullSlave..HighestSlaveAddress];
        (* Addresses 0 or 255 are not valid though 0 is included in the
         * slave addresses because it allows a NULL terminator.
         * 255 actually represents slave address not present on the bus. *)

        SlaveRegister         = [Register0..Register255];

        SlavePriorityStatus   = ( InputWanted, ConditionCheck,
                                  EmergencyCall, Idle);
        SlavePriorityStatusSet = SET OF SlavePriorityStatus;
        (*
         * All these status identities are concerned with access to the slaves
         * during the interrupt processing. Hence not valid in
         * version 0 and 1.
        InputWanted        : solicitated input from this slave.
        EmergencyCall      : interrupt received, but not solicitated.
        ConditionCheck     : Slave has been asked to check a condition
                             continuously, and has found the condition to be
                             out of spec, hence the interrupt.
        Idle               : Slave is sitting there, doing nothing. *)
```

```
SlavePriorityRecord  = RECORD
                           Status  : SlavePriorityStatusSet;
                           Slave   : SlaveType;
                         END;

SlavePriorityListType = ARRAY[LowestSlaveAddress..HighestSlaveAddress]
                           OF SlavePriorityRecord;
(*
 * This list contains the list of slaves in order of priority to be
 * responded to when an interrupt occurs.
 *
 * The list is terminated by a NULL assignmnet.
 *
 * By default, the slaves with lower numbers have higher priorities
 * and the array will be initialised during the first inquiry period
 * after switch on. Thus this list is an automatic way of finding out
 * which slaves are actually connected to ROBUS after a reset.
 *
 * Thereafter, the user has the freedom to change any of the priorities.
 * If a slave is assigned a higher priority than it had previously,
 * all the ones preceeding it are reduced, and conversely, if a slave
 * is assigned a lower priority, all the slave below it will have a
 * higher priority after the reassignment. The 'reshuffling' terminates
 * at the newly assigned slave priority.
 *
 * Note: Version 1 does not contain a procedure for reshuffling
 *
 * These are the intensions on 10 November 1986.
 * Implementation needs to be scrutenised to see if it matches.
 *)

RobusMode      = (reading, writing, controlling, deviceselecting,
                   registerselecting, idle);

SlaveStatus    = (SlaveIdle, SlaveBusy, SlaveIOactive,
                  SlaveNotAvailable, NewSlaveData,
                  OldDataNotAccepted);
SlaveStatusSet = SET OF SlaveStatus;

 (* These slave status indicators are concerned with the run time
  * control of the slaves.
```

```
        SlaveBusy           : the slave is currently executing a task,
        IOactive            : There is IO going on to the slave,
        SlaveNotAvailable   : Double Check: This one is just not on the bus,
        NewSlaveData        : New data from the slave now in buffer,
                              this is the status after initialisation,
        OldDataNotAccepted  : The slave input buffer has overflown,
        There may be many more as time passes by *)

        SlaveSourceAddress  = RECORD
                                reg     : SlaveRegister; (* to identify the register
                                          from which the byte was received *)
                                data    : BYTE; (* The data sent by the slave *)
                              END;

        SlaveBufferType     = RECORD
                                InPtr   : BYTE; (* current input position *)
                                OutPtr  : BYTE; (* current output position *)
                                Status  : SlaveStatusSet;
                                DataBuffer : ARRAY [0..255] OF
                                          SlaveSourceAddress; (* RING *)
                              END;

        SlaveInputBuffer = ARRAY [LowestSlaveAddress..HighestSlaveAddress]
                                  OF SlaveBufferType;

(*
 * Version 1:
 * Since the output is not interrupt driven, there is no real
 * point to have output buffers.
        SlaveOutputBuffers = ARRAY [1..254] OF SlaveBufferType;
*)

        CommunicationMode   = (directly, interruptdriven);
                              (* just as the names suggest *)

        RobusIOResult       = (ok, BufferEmpty, TimeOut, RobusActive,
                               SlaveDataNotValid, SlaveNotThere, Overrun,
                               NoPower, LockedOut, InterruptNotYetAvailable,
                               VMSerror);
        (*
         * The condition when a ROBUS access is attempted.
         *
        ok                  : access was successful,
        BufferEmpty         : there is currently no byte in the slave
                              interrupt buffer
        TimeOut                  : the maximum time for waiting for a
                                   response was exceeded.
        RobusActive              : Somebody(thing) is accessing ROBUS
                                   currently. SlaveDataNotValid
                                 : No byte available for reading.
        SlaveNotThere            : another check for interrupt handling. This
                                   is returned upon access to a the buffer of
                                   a slave, if slave has not been found before.
        Overrun                  : Data has not been accepted by the Slave.
        NoPower                  : ROBUS power has not been switched on.
        VMSerror                 : Something went wrong during the virtual
                                   memory assignment.
        InterruptNotYetAvailable : will be so for a long time (sigh)
         *)

VAR
        SlavePriorityList      : SlavePriorityListType;
        RetryCount             : CARDINAL;

(*----*)

PROCEDURE GetCurrentRobusMode() :RobusMode;
(*
 *      Returns the state of the Signal lines controlling ROBUS
 *)

PROCEDURE ClearRobusControlLines;
(*
 *      Just set all the control bits in the interface inactive, as well as the
 *      control line for the DRV interface.
 *)

PROCEDURE SetRobusDeviceAddress ( slave : SlaveType);
(*
 *      Set up the Robus Slave Address. After that access will be to that
 *      slave only.
 *)

PROCEDURE SetSlaveRegister ( register : SlaveRegister);
(*      Set up the appropriate register address for a slave.
        The hardware will ensure that any slave address will set the
        register address as zero again, if a new slave address has been
        loaded. So this routine is only useful, if it is called AFTER the
        address has been set up.
 *)
```

```
PROCEDURE SetRobusControlLines( RobusControlLines: BYTESET);
(*
 *       Set up the robus interface to access the control lines
 *)

PROCEDURE WriteRobusRawData (value : BYTE): RobusIOResult;
(*
         Attempts to write out a data value to the data lines on ROBUS.
         If successful, ok is returned. Use in conjunction with previously
         set up slave address and register.
*)

PROCEDURE ReadRobusRawData (VAR value : BYTE): RobusIOResult;
(*
         Attempts to read a data value from the data lines on ROBUS.
         If successful, ok is returned. Use in conjunction with previously
         set up slave address and register.
*)

PROCEDURE GetSlaveDataValid(): BOOLEAN;
(*
         Returns the condition of the slave data valid line
 *)

PROCEDURE GetOverrun(): BOOLEAN;
(*
         Returns the condition of the overrun line
 *)

PROCEDURE RobusRead ( slave: SlaveType; VAR register: SlaveRegister;
                      VAR Slaves: SlaveInputBuffer;
                      VAR value: BYTE) : RobusIOResult;

(*
 *       Read from the interrupt buffer for the corresponding slave.
 *
 * 1)    Check whether slave is actually present in the slave priority
 *       list.
 * 2)    Looks at the input buffer and returns the values in the buffer.
 *       --- NOTE --- (version 1)
 *       While the interrupt routines are not implemented, there will only
 *       be data in the buffer after all or one slave have been reset. It is
 *       therefore recommended that RobusDirectRead is used for 'normal'
 *       conversations with slaves in preference to RobusRead.
 × Input:
 ×       slave:  slave's address,
 ×       value:  variable for the byte read from the slave,
 ×
 × Returns:
 ×       ok:               if there was a byte present,
 ×       BufferEmpty:      no bytes available in the buffer,
 ×       SlaveNotPresent:  The slave is not in the priority list,
 ×                         thus assumed not to be present physically,
 ×       register:         The register from which the byte was read
 ×)

PROCEDURE RobusWrite ( slave: SlaveType, register: SlaveRegister,
                       value: BYTE) : RobusIOResult;

(×
 ×       Write the value out to the desired slave.
 ×       This procedure does the following tests before attempting to
 ×       write the value to the slave
 ×
 × Checks whether the slave is actually present in the slave priority list.
 × --- NOTE ---
 × While the interrupt routines are not implemented, this is a
 × straight forward hand-down to the RobusDirectWrite routine. Thus it is
 × recommended that RobusDirectWrite is used in preference to RobusWrite.
 ×
 × Input:
 ×       slave:            slave's address,
 ×       register:         register in the slave,
 ×       value:            byte to write to the slave,
 × Returns:
 ×       ok:               the slave has accepted the byte
 ×       SlaveNotPresent:  The slave is not in the priority list, thus assumed
 ×                         not to be present physically,
 ×       Overrun:          The slave has not accepted the previous byte yet
 ×)

PROCEDURE RobusDirectRead ( slave: SlaveType, register: SlaveRegister,
                            VAR value: BYTE): RobusIOResult;

(×
 × Read from the slave on the robus directly. If there is no new data
 × available, the corresponding error will be returned.
 ×
 * Input:
 *       slave:            slave address,
 *       value:            variable for the byte read from the slave,
 * Returns:
 *       ok:               if the slave has given new data.
 *       SlaveDataNotValid: if no data present
 *)

PROCEDURE RobusDirectWrite ( slave: SlaveType, register: SlaveRegister,
                             value: BYTE) : RobusIOResult,
(*
 * Input:
 *       slave:            slave address,
 *       value:            byte to write to the slave,
 * Returns:
 *       ok:               if the slave has accepted the data,
 *       Overrun:          if the slave has not yet read the previous data
 *)

PROCEDURE InitialiseROBUS ( CommCordHandler: PROC,
                            VAR input-buffer: SlaveInputBuffer,
                            comtype: CommunicationMode) : RobusIOResult,
(*
 * A procedure which sets up the correct conditions for subsequent IO to the
 * slaves, either for direct communication or for interrupt driven
 * Slave-to-Master talking. Master-to-Slave has to be direct all the time.
 *
 * MicroVax/VMS specific setup things:
 * 0) The DRV11J registers are mapped into the virtual memory of the uVax.
 * 1) The DRV11J hardware is set up correctly for the ROBUS interface.
 *
 * Modula-2 Software specific setup things:
 * a) The SlavePriorityList is set up such that it is filled with
 *    the slave addresses actually present on the bus.
 * b) The input buffers are initialised, ready for normal action.
 * c) Do whatever you have to in order to responds to a communication
 *    Cord interrupt.
 *
 *    --- NOTE --- (version 1):
 *    No interrupts available yet.
 *
 ×    This procedure is set up during a RobusInitialisation to do
 ×    nothing. It is the programmer's responsibility to write a correct
 ×    (and system dependent) response routine.
 ×
 × RETURNS:
 ×      NoPower:                if the power to the ROBUS has not been
 ×                              switched on
 ×      InterruptNotYetAvailable: If an attempt is made to initialise the
 ×                              interrupt rouitnes.
 ×      VMSerror:               This is a serious error in the basic setup
 ×                              routine. Please consult your friendly
 ×                              computer manager.
 ×)

PROCEDURE TransparentTalk ( WhichSlave: SlaveType;
                             InitCommand,
                             ExitCommand: ARRAY OF CHAR) : RobusIOResult;
(×
 × Initialise the condition for direct and trasparent talking to the individual
 × slaves. It does assume, a suitable routine for terminal IO exists in the
 × slave. The main reason for this is something like the teaching and program
 × writing for the robots.
 × Input:
 ×      WhichSlave:     The slave's address
 ×      InitCommand:    A string containing the command initiating
 ×                      direct communication in the slave.
 ×                      Usually for the Z80 slave card, which in turn is
 ×                      connected to an external device via the serial line.
 ×      ExitCommand:    A string representing the command which returns
 ×                      the slave to normal Master control.
 × Returns:
 ×      Overrun:        if any of the characters sent to it are
 ×                      not accepted.
 ×      ok:             if the routine has executed as intended.
 ×
 ×      This procedure will terminate upon the sequence of ControlA followed
 ×      by X or x. If ^A is not followed by X or x the ^A character
 ×      will be transmitted to the slave as well as the character which
 ×      has followed it.
 ×)
```

```
PROCEDURE ResetSelectedSlave (slave: SlaveType;
                              VAR Slaves: SlaveInputBuffer;
                              ResetTime: SHORTWORD): RobusIOResult;
(*
 *      Resets a selected slave. Keeps the reset line low for the specified time
 *      (in milliseconds).
 *)

END RobusIO.
```

Appendix D

FTMDataStructures Definition Module

```
DEFINITION MODULE FTMDataStructures;
(* Definition of common data Structures for the FTM Pair description
   Assembly Cell Control *)

FROM Strings IMPORT String;
FROM SYSTEM IMPORT ADDRESS, BYTE;

TYPE
(* Internal representation of information about the robot positions *)
        (* Flag definitions:
           useable:     ok
           entered:     entered from the terminal,
           taught:      by showing the robot,
           loaded:      off-line from CAD?
           compound:    result from (VAL) compound transformation,
           calculated:  such as the frame from three points,
           ByRefOnly:   accessible by reference only, ie values not known
        *)
        TransfStatus    = SET OF (useable, entered, taught, loaded,
                                  compound, calculated, ByRefOnly);
        TransformPtr    = POINTER TO Transformation;
        Transformation  = RECORD (* Equivalent to a VAL location *)
                             ident             : String;
                             status            : TransfStatus;
                             x, y, z,
                             roll, pitch, yaw  : REAL;
                          END;
```

```
(* The points for origins, x and y are deferred to enable exact
   correspondance should they be the same physical points.
   The resultant frame will be unique to this one - hence stored uniquely.
*)
      Frame = RECORD (* Datum Frame *)
                  ident      : String;
                  Resultant  : Transformation;  (* Final Transformation *)
                  origin,    (* Point coinciding with origin *)
                  XPoint,    (* Point on X, & Point on axis || to Y *)
                  YPoint     : TransformPtr;
              END;
      (* Frame list is necessary to find out which have been defined
         and which need still to be taught *)
      FramePtr       = POINTER TO Frame;
      FrameListPtr   = POINTER TO FrameList;
      FrameList      = RECORD
                           FrameEntry   : FramePtr;
                           next         : FrameListPtr;
                       END;
(* --------------------------------------------------------------- *)
(* Different types of parameters must be accomodated for any type of sensor
   and robot *)
      ParameterType  = (Byte, Cardinal, Integer,
                        Real, Name, Adres, Transform, Fram );
      ParameterListPtr = POINTER TO ParameterList;
      ParameterList  = RECORD
                         next            : ParameterListPtr;
                         CASE ParType:ParameterType OF
                           Byte:       byte:       BYTE;           |
                           Cardinal:   cardinal:   CARDINAL;       |
                           Integer:    integer:    INTEGER;        |
                           Real:       real:       REAL;           |
                           Name:       ident:      String;         |
                           Adres:      adres:      ADDRESS;        |
                           Transform:  transform:  TransformPtr;   |
                           Fram:       frame:      FramePtr;       |
                         END;
                       END;
```

```
—(x -------------------------------------------------------------- x)
      DriverResult    = (Ok, ParameterOutOfRange, NotEnoughParameters,
                         HardwareFailure, CommsError, TimeOut);
      DriverRoutine   = (x Handling physical interfaces from information x)
                    PROCEDURE(VAR ParameterList): DriverResult;
      Driver = RECORD (x for individual situations to be handled x)
                    ident       : String;
                    Routine     : DriverRoutine;
                  END;
(x -------------------------------------------------------------- x)
      PassiveRegion = RECORD
                          ident            : String;
                          DatumFrame       : Frame;
                          DatumOffset      : Transformation;
                          FeederDriver     : Driver;
                          PassivePars      : ParameterList;
                        END;
      PassiveRegionPtr = POINTER TO PassiveRegion;
(x -------------------------------------------------------------- x)
      PathRoutine   = PROCEDURE(VAR ParameterList): DriverResult;
      (x Parameter list must be the position at which the transport ceases x)
      PathGenerator = RECORD
                          ident        : String;
                          PathType     : PathRoutine;
                          PathPars     : ParameterList;
                        END;
(x -------------------------------------------------------------- x)
      BoundaryPtr       = POINTER TO BoundaryList;
      BoundaryList      = RECORD
                            lower,
                            upper      : REAL;
                            InfoAxis   : String;
                            next       : BoundaryPtr;
                          END;
(x xxxxxxxxxx NOTE xxxxxxxxxx
```

In the implementation, steps must be provided to ensure that boundaries correspond to the information axes. Suggestion:
At initialisation the identification of both information axes and boundaries are compared and boundaries are ordered to be in the same

```
    sequence. Any comparison can then just 'run through', without having
    to find the axes the current values correspond to them. That way, it
    will also be ensured that the number of boundaries corresponds to the
    number of information axes.
*)
        InfoAxisListPtr = POINTER TO InfoAxisList;
        InfoAxisList = RECORD
                            ident          : String;
                            Information    : Driver;
                            InfoParams     : ParameterList;
                            next           : InfoAxisListPtr;
                          END;
        ActionListPtr = POINTER TO ActionList;
        ActionList = RECORD
                          ident        : String;
                          action       : Driver;
                          actionPars   : ParameterList;
                          next         : ActionListPtr;
                        END;
        SubSpacePtr  =  POINTER TO SubSpace;
        SubSpace     =  RECORD (* for generic and actual data combined *)
                          ident     : String;
                          Actions   : ActionList;
                          Limits    : BoundaryList;
                        END;
        SubSpaceListPtr = POINTER TO SubSpaceList;
        SubSpaceList    = RECORD
                            subspace   : SubSpacePtr;
                            next       : SubSpaceListPtr;
                          END;
(* ----------------------------------------------------------------- *)
        ActualMatePtr = POINTER TO ActualMate;
        ActualMate = RECORD (* for generic and actual data combined *)
                           ident        : String;
                           InfoAxes     : InfoAxisList;
                           SubSpaces    : SubSpaceList;
                           Success      : SubSpacePtr;
                           Approach     : Transformation;
                         END;
```

```
(* --------------------------------------------------------------- *)
        ActiveRegionPtr = POINTER TO ActiveRegion;
        ActiveRegion = RECORD
                          ident          : String;
                          PRegion        : PassiveRegionPtr;
                          ActualMating   : ActualMatePtr;
                          Path           : PathGenerator;
                        END;
(* --------------------------------------------------------------- *)
        FTMInfo = RECORD
                     FeedInfo       : PassiveRegionPtr;
                     TransportInfo  : ActiveRegionPtr;
                     MateInfo       : ActualMatePtr;
                   END;
(* --------------------------------------------------------------- *)
        FTMPairPtr = POINTER TO FTMPair;
        FTMPair = RECORD
                     Head, Tail     : FTMInfo;
                     Nested, Next   : FTMPairPtr;
                   END;
END FTMDataStructures.
```

Appendix E

Generic Assembly Process Definitions

```
;;; Generic descriptions of assembly processes in terms of the data required
;;; to specify them.
;;; (Note: a prefixing underscore denotes negative infinity, and a postfixing
;;; underscore positive infinity.)

generic_mate  (grasp,
 [ [open_jaws, [[tz, Tz_, Tz0], [fz, Fz0, Fz1], [gp, G1, _G1]], open_gripper],
   [clearance, [[tz, Tz_, Tz0], [fz, Fz0, Fz1], [gp, G2, G1]], go_down(v_fast)],
   [approach, [[tz, Tz0, Tz1], [fz, Fz0, Fz1], [gp, G2, G1]], go_down(v_fast)],
   [pick_up_point, [[tz, Tz1, Tz2], [fz, Fz0, Fz1], [gp, G2, G1]], close_gripper],
   [depart, [[tz, Tz_1, Tz2], [fz, Fz2, Fz1], [gp, G3, G4]], go_up(ex_fast)],
   [success, [[tz, Tz_, Tz_1], [fz, Fz2, Fz3], [gp, G3, G4]], null_action]
 ] ).

generic_mate  (keyed_hole_onto_peg,
 [ [pre_contact, [[tz, Tz_, Tz0], [fz, Fz0, Fz1], [gp, G1, G2]], go_down(moderate)],
   [initial_contact, [[tz, Tz0, Tz1], [fz, Fz0, Fz2], [gp, G1, G2]], go_down(slow)],
   [sliding_contact, [[tz, Tz1, Tz2], [fz, Fz0, Fz4], [gp, G1, G2]], go_down(fast)],
   [end_stop, [[tz, Tz2, Tz3], [fz, Fz0, Fz3], [gp, G1, G2]], go_down(slow)],
   [objects_configured, [[tz, Tz2, Tz3], [fz, Fz3, _Fz], [gp, G1, G2]], open_gripper],
   [depart, [[tz, Tz_1, Tz3], [fz, Fz5, Fz6], [gp, G3, G4]], go_up(ex_fast)],
   [success, [[tz, Tz_, Tz_1], [fz, Fz5, Fz6], [gp, G3, G4]], null_action]
 ] ).

generic_mate(hooking_pickup,
 [ [approach, [[tz, Tz_, Tz1],  [fz, Fz0, Fz1], [drz, Drz0, Drz3]], go_down(ex_fast)],
   [hook_twisted, [[tz, Tz1, Tz2], [fz, Fz0, Fz1], [drz, Drz0, Drz2]], neg_rotate_in_z],
   [hook_clear, [[tz, Tz1, Tz4], [fz, Fz0, Fz1], [drz, Drz2, Drz3]], go_down(v_fast)],
   [hook_inserted, [[tz, Tz4, Tz5], [fz, Fz0, Fz1], [drz, Drz1, Drz3]], rotate_in_z],
   [hook_positioned, [[tz, Tz3, Tz5], [fz, Fz2, Fz1], [drz, Drz0, Drz1]], go_up(slow)],
   [lifting, [[tz, Tz0, Tz5], [fz, Fz3, Fz2], [drz, Drz0, Drz1]], go_up(ex_fast)],
   [success, [[tz, Tz_, Tz0], [fz, Fz3, Fz2], [drz, Drz0, Drz1]], null_action]
 ] ).
```

```
generic—mate(hooked—insertion,
 [ [approach, [[tz, Tz—, Tz1], [fz, Fz3, Fz2], [drz, Drz0, Drz1]], go—down(fast)],
   [insert, [[tz, Tz1, Tz2], [fz, Fz3, Fz2], [drz, Drz0, Drz1]], go—down(fast)],
   [release—weight, [[tz, Tz2, Tz3], [fz, Fz3, Fz1], [drz, Drz0, Drz1]], go—down(fast)],
   [clear—hook, [[tz, Tz2, Tz3], [fz, Fz1, Fz0], [drz, Drz0, Drz1]], go—down(fast)],
   [rotate—hook, [[tz, Tz3, Tz4], [fz, Fz1, Fz0], [drz, Drz0, Drz2]], neg—rotate—in—z],
   [withdrawing, [[tz, Tz0, Tz4], [fz, Fz1, Fz0], [drz, Drz2, Drz3]], go—up(ex—fast)],
   [success, [[tz, Tz—, Tz0], [fz, Fz1, Fz0], [drz, Drz2, Drz3]], null—action]
 ] ) .

generic—mate  (hole—onto—peg,
 [ [approach, [[tz, Tz—, Tz1], [fz, Fz0, Fz1], [gp, G0, G1]], go—down(moderate)],
   [insertion, [[tz, Tz1, Tz2], [fz, Fz0, Fz1], [gp, G0, G1]], go—down(ex—fast)],
   [make—contact, [[tz, Tz2, Tz3], [fz, Fz0, Fz4], [gp, G0, G1]], go—down(moderate)],
   [configured, [[tz, Tz2, Tz3], [fz, Fz4, Fz5], [gp, G0, G1]], open—gripper],
   [withdrawing, [[tz, Tz0, Tz3], [fz, Fz2, Fz3], [gp, G2, G3]], go—up(ex—fast)],
   [success, [[tz, Tz—, Tz0], [fz, Fz2, Fz3], [gp, G2, G3]], null—action]
 ] ).

generic—mate  (null, null). ;;; Used to terminate building of the sensory space
```

Appendix F

Specific Assembly Process Definitions

```
%%% Product-specific data for the Flymo assembly...

%%% Order of assembly...

sequence (flymo,
          [ ftm(gripper, motor—tool, grasp),
            ftm([gripper, motor—tool], motor, hooking—pickup),
            ftm([gripper, motor—tool, motor], jig, hooked—insertion),
            ftm([gripper, motor—tool], tool—holder, hole—onto—peg),
            ftm(gripper, bush, grasp),
            ftm([bush, gripper], [motor, jig], keyed—hole—onto—peg)
          ]).

%%% Mating descriptions...

mate(gripper, motor—tool, grasp,
  [ [open—jaws,
      [[tz, positive—infinity, -2.0], [fz, 0.35 , -3.0], [gp, 4650, negative—infinity]],
        O—]—Action],
    [clearance,
      [[tz, positive—infinity, -2.0], [fz, 0.35, -3.0], [gp, positive—infinity, 4650]],
        C—Action],
    [approach,
      [[tz, -2.0, -41.75], [fz, 0.35, -3.0], [gp, positive—infinity, 4650]],
        A—Action],
    [pick—up—point,
      [[tz, -41.75, -43.0], [fz, 0.35, -3.0], [gp, positive—infinity, 4650]],
        P—U—P—Action],
    [depart,
      [[tz, 20.0, -43.0], [fz, 3.0, -3.0], [gp, 3830, 3780]],
        D—Action],
    [success,
      [[tz, positive—infinity, 20.0], [fz, 3.0, 0.5], [gp, 3830, 3780]],
        S—Action]
  ] ).
```

```
mate([bush, gripper], [motor, jig], keyed-hole-onto-peg,
[ [pre-contact,
     [[tz, positive-infinity, -3.0], [fz, 0.25, -4.0], [gp, 3920, 3870]],
        P-C-Action],
  [initial-contact,
     [[tz, -3.0, -6.0], [fz, 0.25, -10.0], [gp, 3920, 3870]],
        I-C-Action],
  [sliding-contact,
     [[tz, -6.0, -16.0], [fz, 0.25, -10.0], [gp, 3920, 3870]],
        S-C-Action],
  [end-stop,
     [[tz, -16.0, -17.5], [fz, 0.25, -10.0], [gp, 3920, 3870]],
        E-S-Action],
  [objects-configured,
     [[tz, -16.0, -17.5], [fz, -10.0, negative-infinity], [gp, 3920, 3870]],
        O-C-Action],
  [depart,
     [[tz, 30.0, -17.5], [fz, 0.25, -1.5], [gp, positive-infinity, 4650]],
        D-Action],
  [success,
     [[tz, positive-infinity, 30.0], [fz, 0.25, -1.5], [gp, positive-infinity, 4650]],
        S-Action]
] ).


mate(gripper, bush, grasp,
[ [open-jaws,
     [[tz, positive-infinity, -2.0], [fz, 0.25 , -1.5], [gp, 4650, negative-infinity]],
        O-J-Action],
  [clearance,
     [[tz, positive-infinity, -2.0], [fz, 0.25, -1.5], [gp, positive-infinity, 4650]],
        P-C-Action],
  [approach,
     [[tz, -2.0, -21.2], [fz, 0.25, -1.5], [gp, positive-infinity, 4650]],
        A-Action],
  [pick-up-point,
     [[tz, -21.2, -22.05], [fz, 0.25, -1.5], [gp, positive-infinity, 4650]],
        P-U-P-Action],
  [depart,
     [[tz, 10.0, -22.05], [fz, 1.5, -1.5], [gp, 3920, 3870]],
        D-Action],
  [success,
     [[tz, positive-infinity, 10.0], [fz, 1.5, 0.15], [gp, 3920, 3870]],
        S-Action] ] ).
```

```
mate( [gripper, motor—tool], motor, hooking—pickup,
[ [approach,
     [[tz, positive—infinity, 0.1], [fz, 0.4, -1.0], [drz, 18.5, -0.5]],
        A—Action],
   [hook—twisted,
     [[tz, 0.1, -4.0], [fz, 0.4, -1.0], [drz, 18.5, 0.5]],
        H—T—Action],
   [hook—clear,
     [[tz, 0.1, -18.0], [fz, 0.4, -1.0], [drz, 0.5, -0.5]],
        H—C—Action],
   [hook—inserted,
     [[tz, -18.0, -21.0], [fz, 0.4, -1.0], [drz, 16.5, -0.5]],
        H—I—Action],
   [hook—positioned,
     [[tz, -13.0, -21.0], [fz, 18.0, -1.0], [drz, 18.5, 16.5]],
        H—P—Action],
   [lifting,
     [[tz, 40.0, -21.0], [fz, 24.0, 18.0], [drz, 18.5, 16.5]],
        L—Action],
   [success,
     [[tz, positive—infinity, 40.0], [fz, 24.0, 18.0], [drz, 18.5, 16.5]],
        S—Action]
] ).
```

```
mate([gripper, motor—tool, motor], jig, hooked—insertion,
[ [approach, [[tz, positive—infinity, 0.0], [fz, 1.0, -5.0], [drz, 0.5, -0.5]],
A—Action],
[insert, [[tz, 0.0, -17.0], [fz, 1.0, -5.0], [drz, 0.5, -0.5]],
I—Action],
[release—weight, [[tz, -17.0, -26.0], [fz, 1.0, -18.0], [drz, 0.5, -0.5]],
R—W—Action],
[clear—hook, [[tz, -17.0, -26.0], [fz, -18.0, -24.0], [drz, 0.5, -0.5]],
C—H—Action],
[rotate—hook, [[tz, -26.0, -28.0], [fz, -18.0, -24.0], [drz, 0.5, -16.5]],
R—H—Action],
[withdrawing, [[tz, 20.0, -28.0], [fz, -18.0, -24.0], [drz, -16.5, -18.5]],
W—Action],
[success, [[tz, positive—infinity, 20.0], [fz, -18.0, -24.0], [drz, -16.5, -18.5]],
S—Action]
] ) .
```

```
mate([gripper, motor—tool], tool—holder, hole—onto—peg,
[ [approach, [[tz, positive—infinity, -28.0], [fz, 1.0, -2.1], [gp, 3830, 3780]],
go—down(moderate)],
[insertion, [[tz, -28.0, -42.0], [fz, 1.0, -2.1], [gp, 3830, 3780]],
  go—down(ex—fast)],
  [make—contact, [[tz, -42.0, -44.0], [fz, 1.0, -5.0], [gp, 3830, 3780]],
  go—down(moderate)],
[configured, [[tz, -42.0, -44.0], [fz, -5.0, -10.0], [gp, 3830, 3780]],
open—gripper],
[withdrawing, [[tz, 10.0, -44.0], [fz, -1.5, -3.0], [gp, positive—infinity, 4650]],
go—up(ex—fast)],
[success, [[tz, positive—infinity, 10.0], [fz, -1.5, -3.0], [gp, positive—infinity, 4650]],
null—action]
] ).

mate([gripper, motor—tool], tool—holder, hole—onto—peg,
[ [pre—contact, [[tz, positive—infinity, 0.0], [fz, 0.25, -0.25], [gp, 3830, 3780]], P—C—Action],
[sliding—contact, [[tz, 0.0, -41.75], [fz, 0.25, -5.0], [gp, 3830, 3780]], S—C—Action],
[end—stop, [[tz, -41.75, -43.0], [fz, -5.0, -6.0], [gp, 3830, 3780]], E—S—Action],
[objects—configured, [[tz, -41.75, -43.0], [fz, -6.0, -7.0], [gp, 3830, 3780]], O—C—Action],
[depart, [[tz, 10.0, -43.0], [fz, -2.0, -2.5], [gp, positive—infinity, 4650]], D—Action],
[success, [[tz, positive—infinity, 10.0], [fz, -2.0, -2.5], [gp, positive—infinity, 4650]], S—Action]
] ).

mate(null, null, null, null).   ;;; Used to terminate building of sensory space
```

Appendix G

Accumulated Error Definitions

```
error_region([gripper, motor_tool], motor, hooking_pickup,
   hook_not_clear,
   [[tz, -5.0, -10.0], [fz, -0.25, -5.0], [mz, 100.0, -100.0], [drz, 0.5, -0.5],
           [gp, 3830, 3780]],
   rz_comply).

error_region([gripper, motor_tool], motor, hooking_pickup,
   xy_compliance,
   [[tz, -11.0, -21.0], [fz, -0.25, -3.0], [mz, 100.0, -100.0], [drz, 0.5, -0.5],
           [gp, 3830, 3780]],
   xy_comply).

error_region([gripper, motor_tool], tool_holder, hole_onto_peg,
   not_aligned,
   [[tz, -27.0, -28.0], [fz, -2.0, -5.0], [mz, 100.0, -100.0], [drz, 0.5, -0.5],
           [gp, 3830, 3780]],
   align).

error_region([bush, gripper], [motor, jig], keyed_hole_onto_peg,
   flats_misaligned,
   [[tz, -3.0, -6.0], [fz, -10.0, -31.0], [mz, 200.0, -200.0], [drz, 361.0, -0.5],
           [gp, 3920, 3870]],
   rotate_in_z).

error_region([bush, gripper], [motor, jig], keyed_hole_onto_peg,
   xy_compliance,
   [[fxy_sqd, positive_infinity, 16.0], [fz, 0.0, -40.0], [tz, -3.0, -17.5],
           [mz, 200.0, -200.0], [drz, 361.0, -0.5], [gp, 3920, 3870]], x
   y_comply).

error_region([bush, gripper], [motor, jig], keyed_hole_onto_peg,
   tz_compliance,
   [[tz, -3.0, -17.5], [fz, 0.0, -50.0], [mz, 20.0, -20.0], [drz, 361.0, -0.5],
           [gp, 3920, 3870]],
   tz_comply).
```

```
error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   stuck–rotation,
   [[tz, -3.0, -17.5], [fz, 0.0, -40.0], [mz, 300.0, -300.0], [drz, 361.0, -0.5],
          [gp, 3920, 3870]],
   tz–comply).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   back–off,
   [[tz, -3.0, -17.5], [fz, 0.0, -50.0], [mz, 400.0, -400.0], [drz, 361.0, -0.5],
          [gp, 3920, 3870]],
   tz–comply).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   skew,
   [[tz, -3.0, -17.5], [fz, 0.0, -50.0], [mz, 400.0, -400.0], [drz, 361.0, -0.5],
          [gp, 4100, 3870]],
   xy–comply).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   alignment,
   [[tz, 0.0, -3.0], [fz, -4.0, -25.0], [mxy–sqd, 9000.0, 25.0], [gp, 3920, 3870]],
   align).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   alignment–backoff,
   [[tz, 0.0, -3.0], [fz, -24.0, -38.0], [mxy–sqd, 9000.0, 25.0], [gp, 3920, 3870]],
   tz–comply).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   alignment–severe–backoff,
   [[tz, 0.0, -3.0], [fz, -37.0, -55.0], [mxy–sqd, 9000.0, 25.0], [gp, 3920, 3870]],
   tz–comply).

error–region([bush, gripper], [motor, jig], keyed–hole–onto–peg,
   alignment–torque–backoff,
   [[tz, 0.0, -3.0], [fz, -4.0, -55.0], [mxy–sqd, 98000.0, 25.0], [gp, 3920, 3870]],
   tz–comply).
```

```
error_region(gripper, bush, grasp,
    too_much_push,
    [[tz, -21.2, -22.05], [gp, 3925, 3875], [fz, -1.5, -3.0]],
    tz_comply).

error_region(gripper, bush, grasp,
    missed_pick_up,
    [[tz, -22.05, negative_infinity], [gp, positive_infinity, 4660]],
    go_up(slow)).

error_region(gripper, bush, grasp,
    overshoot,
    [[tz, -22.05, negative_infinity], [gp, 4660, negative_infinity]],
    open_gripper).

error_region([bush, gripper], [motor, jig], keyed_hole_onto_peg,
    gripper_stuck,
    [[tz, 30.0, -17.5], [fz, -0.1, -2.0], [gp, positive_infinity, 4000]],
    open_gripper).

error_region([bush, gripper], [motor, jig], keyed_hole_onto_peg,
    gripper_caught,
    [[tz, 30.0, -17.5], [fz, -10.0, negative_infinity], [gp, positive_infinity, 4000]],
    tz_comply).
```

Appendix H

Automated Calibration of Reference Beacons

One of the major problems in a rapidly reconfigurable environment is the repeated changing of conditions for sensory measurements. More often than not, the acquired data is used to influence the motion of the robot and thus make up the shortfall in the manipulator's and the environment's degree of positional repeatability. The *translation* of sensory data into a format which is suitable and relevant for the robot, is carried out during a calibration procedure. It establishes the geometric relationship between the robot's and the sensor's frames of reference because they are unlikely to be the same. In general, such procedures are relatively time consuming and thus reduce the attraction of using sensors.

H.1 *Eye-In-Hand* Vision

A physically small camera can be mounted on the end-effector, acquiring itself the same degrees of freedom as the robot. In such a way, even a relatively low resolution can be exploited over the whole range of the robot's working envelope as the field of view can be made to be small. According to need, the camera can be placed at the point where there is a feature of interest.

The provision of beacons to enable Local Accuracy (see section 5.1.2) makes full use of this characteristic. In the course of the product assembly, beacons can be placed anywhere within the working envelope. It would not be feasible to provide a stationary vision system (or even a set of cameras) to achieve the desired resolution. Indeed, it is not necessary to know about those areas which do not contain beacons for the purpose of this calibration. A wrist- or gripper-mounted vision system overcomes this problem. It is frequently referred to as *Eye-In-Hand* vision.

Although the relationship between the two coordinate frames is fixed once the camera is mounted, a relatively detailed knowledge is still required by the operator to carry out the initial calibration. Conditions under which calibration needs to be performed include a change of cameras because individual systems may differ significantly in the important parameters. In fact, any changes in the hardware such as a repair or maintenance should also be followed by a re-calibration. Because camera mobility, the additional problem of an unknown distance from the object arises and also has an influence upon the calibration. Ideally, such details should be known from the design stages and thus could be taken into account, but possible variations encountered in batch manufacturing systems may not be kept tight enough with economical means.

H.2 Initial State Required

One of the aims in providing an automated calibration procedure is a minimum demand on operators. In the case described here, the beacon (an LED) needs to be placed somewhere in the field of view of the camera. Since this is generally much larger than the area which represents the image of the beacon in the sensor, no special precision is required and hence a minimum amount of time is necessary. Alternatively, if the positional information has been acquired through a CAD interface, then the mismatch between the real world and the model in the computer software representing that world can be relatively large. In either case, these variations can be tolerated.

Before the calibration procedure is started, the parameters of the robot are known and it is possible to use these to find the relation between camera and robot through observations. In other words, if the robot is moved by a known amount, then the resulting displacement of the object within the field of view the camera allows a calibration of how many pixels correspond to a millimetre. The use of constant features on the observed objects (beacons for all reference frames, compare figure 6.10) lends itself ideally to this kind of argument.

There are some practical problems to overcome in order to take account of the imperfections in the system.

- **General Sensor Non-Linearities**

  Even once the system is calibrated, the relationship may not appear to be constant. For example, some distortions in the lens system or variations of pixel sensitivity across the light sensitive area may distort readings. This makes is relatively difficult to apply a *visual servoing* algorithm. Much of these imperfections are not evident in more expensive vision systems, but the use of cheap (and thus more likely) equipment may well have to contend with such deviations.

- **Non-square aspect ratio**

Some vision sensors based on electronic tubes scanning (such as Vidicons) do not have a rectangular aspect ratio. This is also the case for the cheaper dynamic RAM cameras, where a square aspect ratio of element placing was not a design consideration. Consequently, the constants of movements vary in the important directions and have to be determined after the cameras have been installed.

- **2D Measurements**

The two dimensions in which measurements can be carried out in a vision system cannot provide any depth information directly. Thus variations in distances from the camera caused by re-organisation of the workspace is difficult to detect.

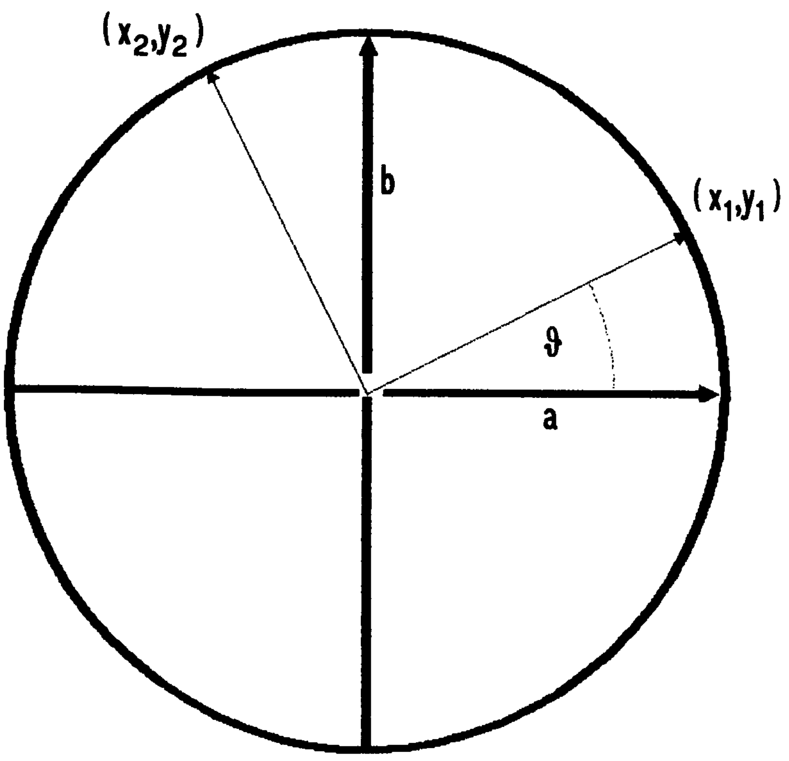


Figure H.1: Frame Calibration Map

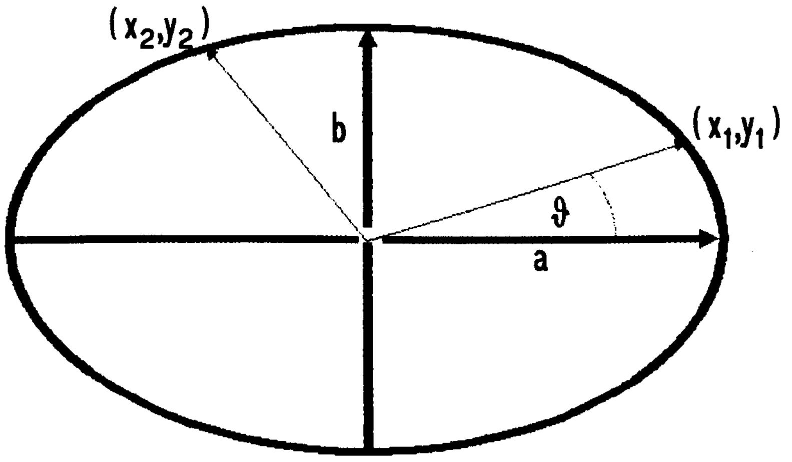


Figure H.2: Distortion of Calibration Circle

H.3 The Calibration Method

If the relationship between robot and camera is unknown, then it is possible to move the robot a known distance d along each of its axes thought to correspond to those of the camera and observe the apparent change in the centre of the object. Figure H.1 shows that a move from an arbitrary point can then result in a measurement taken anywhere along the circumference of the circle. For a motion along the robot's x-axis, the new coordinate pair (x_1,y_1) results and for the move along the y-axis the pair (x_2,y_2) can be obtained. The relative orientation around the normal to the plane of view is given by the simple geometric relation of

$$\vartheta = \text{ATAN2}\ (y_1,\ x_1) \qquad \text{(H-1)}$$

where ϑ is the orientation angle and ATAN2 is the arctangent function returning the value corrected according to the quadrant. However, if the distortion of the circle due to a non-square aspect ratio, R, is also taken into account, then the angle is given by

$$\vartheta = \mathrm{ATAN2}\,(R\ y_1,\ x_1) \qquad \text{(H-2)}$$

and the number of pixels per mm, p, is obtained from

$$p = \frac{\sqrt{(R\ y_1)^2 + x_1^2}}{d} \qquad \text{(H-3)}$$

Both equations (H-2) and (H-3) involve the aspect ration which can be calculated from the two coordinate pairs and their relationship to the general ellipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad \text{(H-4)}$$

where a and b are the major and minor axes respectively. They are related by the unknown ratio

$$R = \frac{a}{b} \qquad \text{(H-5)}$$

Since both sets of coordinates coincide with the circumference of the ellipse, it must hold that

$$\frac{x_1^2}{a^2} + \frac{y_1^2}{b^2} = 1 \qquad \text{(H-6)}$$

and

$$\frac{x_2^2}{a^2} + \frac{y_2^2}{b^2} = 1 \qquad \text{(H-7)}$$

These are two equations with two unknown variables and can thus be solved to result in an expression for a^2

$$a^2 = \frac{x_1^2}{1 - \frac{y_1^2}{b^2}} \qquad \text{(H-8)}$$

Substituting this into equation (H-7) results in an expression for b^2

$$b^2 = \frac{x_2^2\, y_1^2 - x_1^2\, y_2^2}{x_2^2 - x_1^2} \qquad \text{(H-9)}$$

and thus a and b can be determined from the two sets of coordinates.

Two special cases have to be considered which arise from the possibility of the denominators in equation (H-8) and (H-9) resulting in a zero value

i) when $x_2^2 = x_1^2$ and

ii) when $y_1^2 = b^2$.

Both cases have precise physical meaning and can be determined as well from the considerations in figures H.1 and H.2. If the squares of both x-values are the same, then the angle ϑ is determined from one of the four possibilities

$$\vartheta = \begin{cases} 45^o & \text{for } x_1 > 0 \text{ and } x_2 = < 0 \\ 135^o & \text{for } (x_1 = x_2) < 0 \\ -135^o & \text{for } x_1 < 0 \text{ and } x_2 = > 0 \\ -45^o & \text{for } (x_1 = x_2) > 0 \end{cases} \qquad \text{(H-10)}$$

In the second case, ii), an equality of the squared value of y_1 and b indicates a value for ϑ as

$$\vartheta = \begin{cases} 90^o \text{ for } y_1 > 0 \\ -90^o \text{ for } y_1 < 0 \end{cases} \qquad \text{(H-11)}$$

The aspect ratio is also easily determined for a right angle relationship as

$$R = \frac{x_1}{x_2} \qquad \text{(H-12)}$$

Knowing the calibration constants, it is possible to calculate the distance and relative coordinates from the current position to the centre point. The centre point is chosen arbitrarily as the fixed point of reference. All measurements are thus repeatable in the same way, regardless of any variations in camera geometry and sensitivity.

The determination of signs is also important, as it represents a further possible rotation between the robots plane of movement and the plane of the imaging device. The axis of rotation can now be pictured as coinciding with the picture plane, that is perpendicular to the axis around which ϑ was measured, and an opposite sign in the x- and y-calibration constants is equivalent to 180^{o} rotation around that axis.

H.4 Further Improvements

By the same argument, the distance of the beacons from the camera could also be estimated by using a similar technique. The robot could be moved away by a known amount and then repeat a single movement in the x-y plane. Using the principle of triangulation, the variation in the pixel to movement ratio gives an indication about the distance away from the camera.

However, the assumption that two dimensions are sufficient is valid within the majority of robotic assembly situations, as the dominant direction of insertion or placement is directly from above. Thus the third coordinate necessary for each of the positions can be assumed to be constant for all three measurements. This is not so if the plane on which parts are presented is inclined with respect to the table. This may well occur in gravity feeding situations.

In general, it cannot be assumed that the movement carried out by the robot always contains the object wholly within the field of view. Checks have to be made when calibrating that the area of the object remains as constant as possible and some trials like shifting the origin of the calibration circle should be incorporated for the ease of the operator.

H.5 Summary

In this appendix, the automated calibration method was presented which requires a minimum of operator attention or knowledge. The use of a single calibration method is recommended throughout the cell, as this encourages a higher degree of familiarity with only one system. The results have been presented in chapter 6, showing that this is a viable method to employ on the factory floor.