ASTON UNIVERSITY IHD SCHEME

THE APPLICATION OF ROBOTICS TO THE TURBINE BLADE ENCAPSULATION PROCESS

ANTHONY RICHARD WAITE

Doctor Of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM
April 1987

Collaborating Organisation : Cirrus Equipment Limited

The University of Aston in Birmingham

The Application of Robotics to the Turbine Encapsulation Process

Anthony Richard Waite

Doctor of Philosophy 1987

SUMMARY

A survey of the existing state-of-the-art of turbine blade manufacture highlights two operations that have not been automated namely that of loading of a turbine blade into an encapsulation die, and that of removing a machined blade from the encapsulation block. The automation of blade decapsulation has not been pursued.

In order to develop a system to automate the loading of an encapsulation die a prototype mechanical handling robot has been designed together with a computer controlled encapsulation die. The robot has been designed as a mechanical handling robot of cylindrical geometry, suitable for use in a circular work cell. It is the prototype for a production model to be called "The Cybermate".

The prototype robot is mechanically complete but due to unforeseen circumstances the robot control system is not available (the development of the control system did not form a part of this project), hence it has not been possible to fully test and assess the robot mechanical design. Robot loading of the encapsulation die has thus been simulated.

The research work with regard to the encapsulation die has focused on the development of computer controlled, hydraulically actuated, location pins. Such pins compensate for the inherent positional inaccuracy of the loading robot and reproduce the dexterity of the human operator. Each pin comprises a miniature hydraulic cylinder, controlled by a standard bi-directional flow control valve. The precision positional control is obtained through pulsing of the valves under software control, with positional feedback from an 8-bit transducer. A test-rig comprising one hydraulic location pin together with an opposing spring loaded pin has demonstrated that such a pin arrangement can be controlled with a repeatability of +/-.00045". In addition this test-rig has demonstrated that such a pin arrangement can be used to gauge and compensate for the dimensional error of the component held between the pins, by offsetting the pin datum positions to allow for the component error. A gauging repeatability of +/- 0.00015" was demonstrated. This work has lead to the design and manufacture of an encapsulation die comprising ten such pins and the associated computer software. All aspects of the control software except blade gauging and positional data storage have been demonstrated. Work is now required to achieve the accuracy of control demonstrated by the single pin test-rig, with each of the ten pins in the encapsulation die. This would allow trials of the complete loading cycle to take place.

Keyword : ROBOTICS, AUTOMATION, TURBINE BLADES, ENCAPSULATION, GAUGING

ACKNOWLEDGEMENTS

I would like to express my thanks to my Academic Supervisors Dr.T.J. Vickerstaff and Professor R.H. Thornley, to my Industrial Supervisor Dr. H.T. Hayward and to my Tutor Dr. A.J. Cochran for their guidance and encouragement throughout this project.

I would also like to thank the Directors of Cirrus Equipment Limited for sponsoring this project and for making available the facilities that were required for the completion of the project.

Finally I would like to thank Mrs. M. Parker for her help in preparing this thesis.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL	DESCRIPTION	UNIT
v	linear velocity	ms^{-1}
a	linear acceleration	ms^{-2}
t	time	s
w	angular velocity	rad s^{-1}
α	angular acceleration	rad s^{-2}
T	torque	Nm
I	moment of inertia	kgm^2
J	polar moment of inertia	m^4
P	power	W
R	radius	m
x	length	m
l	length	m
L	length	m
e	efficiency	%
pi	3.14159	
M	mass	kg
d	diameter	m
p	lead-screw pitch	m
r	gearbox reduction ratio	
g	gravitational acceleration	(9.81 ms^{-2})

1.0 INTRODUCTION

1.1 Profile of Cirrus Equipment Limited

Cirrus Equipment Limited is a privately owned engineering company which was started 20 years ago and is based in Redditch, Worcestershire. The Company employs eighty people and has a turnover of approximately £ 2.5 million.

The Company designs and manufactures specialist automated equipment. The range of equipment that has been produced in the past is diverse, but has been concentrated in two areas, the Motor Industry and the Aircraft Engine Industry. Most of the equipment supplied to the Motor Industry is for mechanical handling applications and increasingly involves the use of sophisticated computer control systems. It was thus felt towards the end of 1981 that it would be a logical extension of the Company's capabilities to enter the field of robotic manufacture and supply.

The second major area of work for the Company is the supply of turbine blade encapsulation equipment to the Aircraft Industry. At the time that the Company was considering entry into the robot market it was also aware of the increasing automation of the total turbine blade machining process. Since it was necessary to identify possible application areas should the Company decide to produce it's own robot, this was seen as one such market. This project was thus designed in order to identify and produce automation proposals for those areas of the blade machining process that had not been automated, and to design, build and apply a suitable mechanical handling robot which at the same time would be the prototype for a production version.

1.2 Aircraft Engine Turbine Blades

A modern gas turbine aero-engine contains several hundred turbine and compressor blades. Figure 1 shows a three stage aero-engine that has three sets of compressor blades and three sets of turbine blades. The blades are short in comparison with the radius of the rotor to which they are attached and are located radially to the axis of the engine.

The compressor blades compress the air that enters the intake of the engine prior to the air being heated. After heating, the air expands through the turbine stages of the engine, driving the turbines and generating the power to drive the compressors, and depending on the type of engine, a propeller or by-pass fan as well. For a detailed description of the operation of a gas turbine engine see [1].

The manufacture of turbine blades is one of large scale batch production (for simplicity the term "turbine blade" will be used to indicate both compressor blades and turbine blades unless the distinction is important). The large number of turbine blades in an aero-engine forms a significant proportion of the total engine cost, which when coupled with the fact that large numbers of identical blades need to be manufactured, indicates that automation of the turbine blade manufacturing process is economically desirable.

The manufacture of turbine blades involves complex machining operations to very tight dimensional tolerances. The financial investment in a blade as it passes through the manufacturing cycle is thus high (typically £ 1500 per blade).

Since turbine blades account for such a significant proportion of the overall cost of an aircraft engine, it is logical to attempt to minimise the cost of blade manufacture through the use of automation.

In addition, it is feasible that the methods of automation would be such as to bring about improvements in the quality of the manufactured blades (for example, the tightening of existing dimensional tolerances) thus as well as a reduction in the cost of blade manufacture, an improvement in the performance of the aero-engine could be possible through the introduction of manufacturing automation.

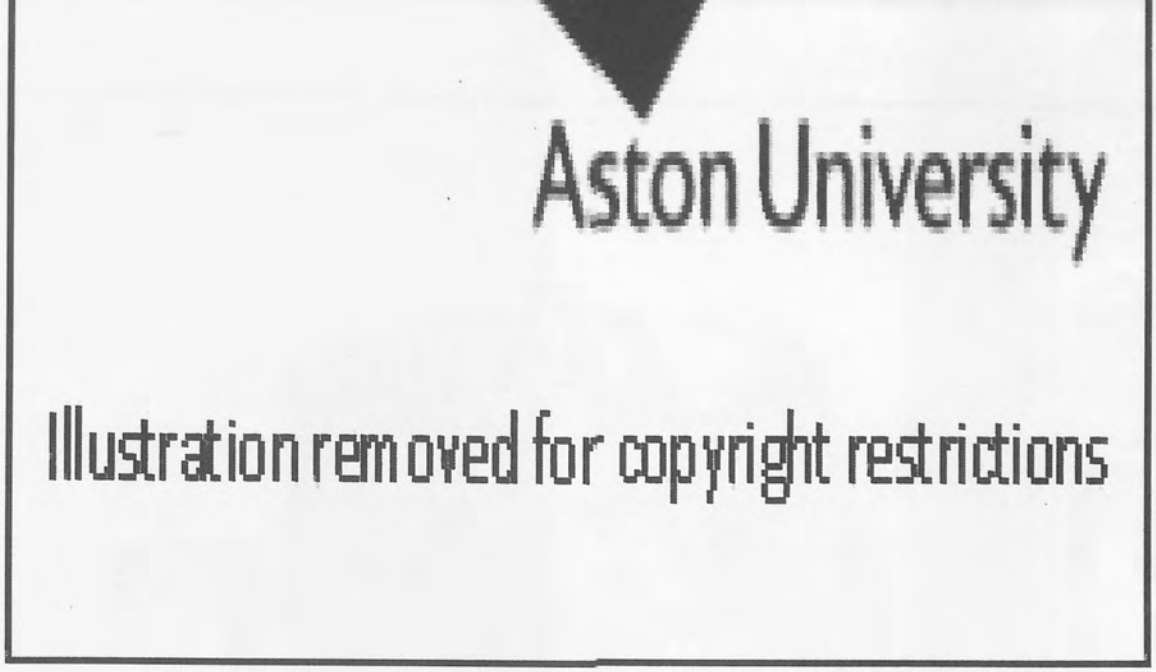


Figure 1 : Cutaway Drawing of an Aircraft Engine

A typical turbine blade is shown in Figure 2. The blade consists of the root (which anchors the blade to the engine rotor), the aerofoil (which directs the airstream) and the shroud (which prevents the air by-passing the aerofoil). A compressor blade would have a simple root fixture rather than the fir-tree of the turbine blade and would normally be shroudless.

A turbine blade is first formed as a casting or a forging. In the cast or forged state the aerofoil of the blade requires only polishing in order to be finished, and forms the datum for the machining of the root and shroud ends of the blade. This requires the extremely complex shape of the aerofoil (comprising compound curves, thin sections and a lack of flat location surfaces) to be located and clamped during the very precise blade machining operations. Clamping of the aerofoil is undesirable because of the difficulties involved and because of possible damage to the already finished aerofoil.

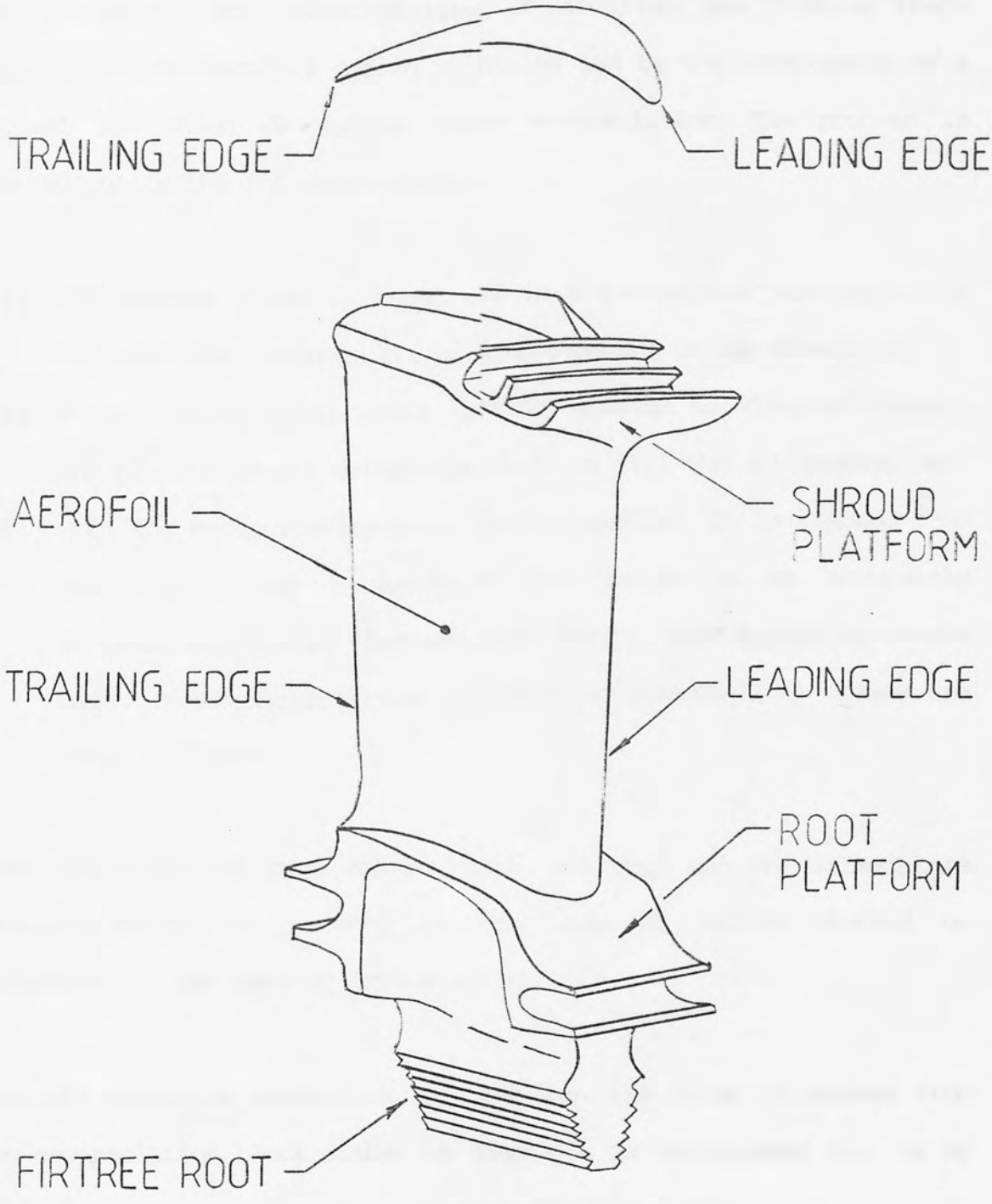


Figure 2 : A Typical Turbine Blade

1.3 Turbine Blade Encapsulation

The difficulty and undesirability of locating the turbine blade directly on the aerofoil during machining led to the development of a process now known as turbine blade encapsulation. The process is carried out in the following manner:

i) the turbine blade is supported in a rectangular die cavity by die-pins that locate the aerofoil relative to the die-faces;

ii) a low melting point metal (either zinc or an alloy of bismuth and tin) is poured around the blade to fill the die cavity; and

iii) when the encapsulating metal has solidified, it is removed from the die having encapsulated the blade in an accurately dimensioned, easily handled metal block, thus providing simple datum faces for location. A number of encapsulated blades are shown in Figure 3.

Once the blade has been encapsulated, the root and shroud ends are machined either by grinding (in the case of turbine blades) or broaching (in the case of compressor blades).

Once all machining operations are complete, the blade is removed from the encapsulation block either by immersion in hot mineral oil, or by splitting the encapsulation block away from the blade.

It is the process of encapsulation that has facilitated automation of the turbine blade manufacturing process.

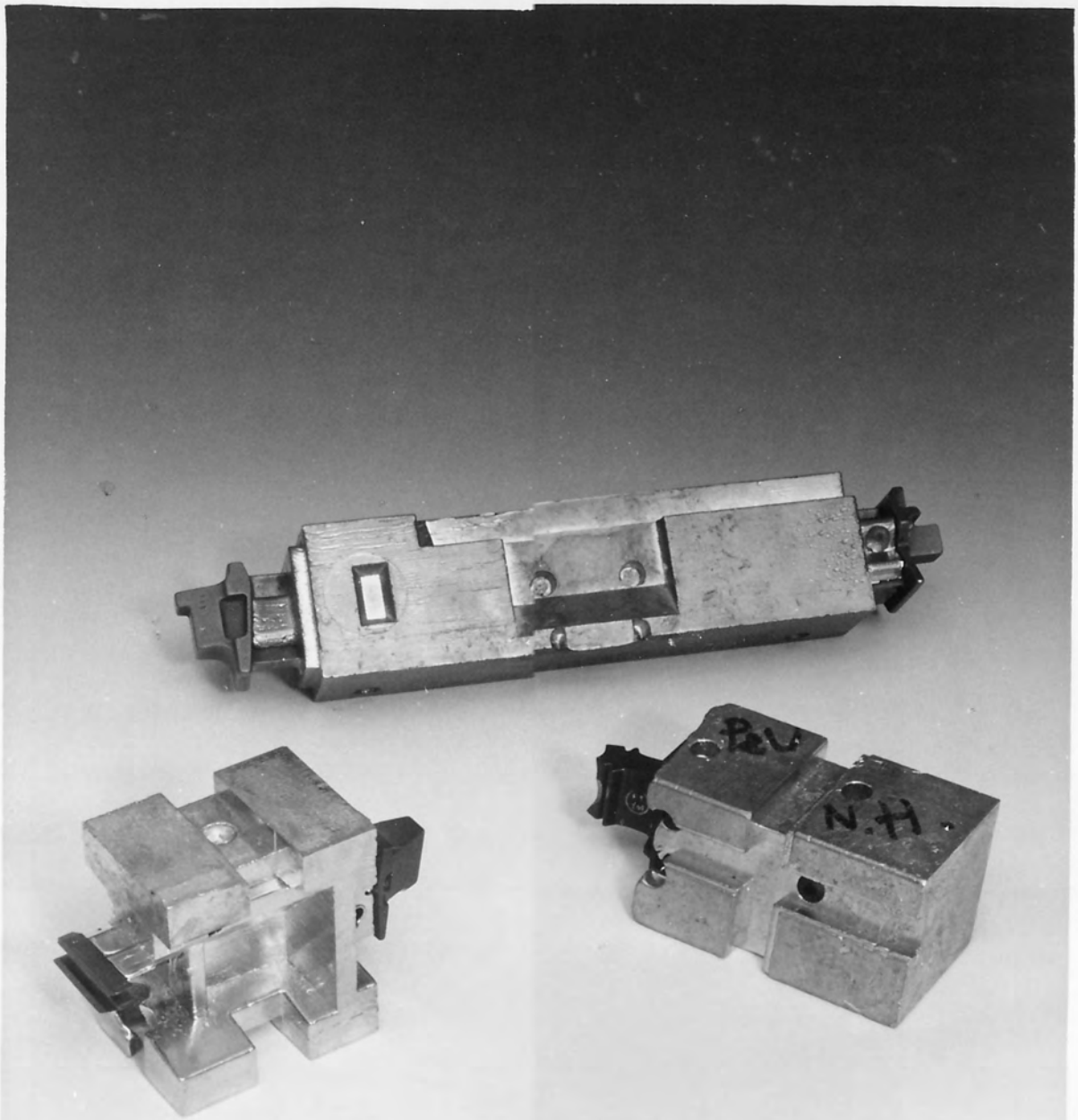


Figure 3 Typical Turbine Blade Encapsulations

1.4 Project Objectives

At the start of the project it was known that since turbine blade encapsulation has been in use as an industrial process for a number of years, the automation of the blade machining process following encapsulation is well advanced. It was also known that no successful attempts had been made to automate the process of turbine blade encapsulation itself. The two broad objectives of the project were thus formulated to be :

i) to survey the aero-engine manufacturing industry in order to establish the current levels of automation used in the turbine blade manufacturing process and to establish those areas of the manufacturing process that have not yet been automated and that would benefit from automation; in addition to establish the requirements of and the constraints placed upon the manufacturing methods used in those areas to be considered for automation; and

ii) to carry out a programme of experimental work in order to produce proposals for the automation of both the encapsulation process itself and any other areas of the blade manufacturing process highlighted as a result of the above survey.

It was anticipated that the automation of the blade encapsulation process would require the use of a mechanical handling robot. It was decided that the mechanical design and development of an industrial mechanical handling robot, suitable for the automation of the blade encapsulation process, would constitute one part of the work of the project. The robot thus designed would form the prototype for a commercial production robot.

1.5 Literature Review

In accordance with the principles of interdisciplinary research this project covers a number of related but distinct fields. Consequently a number of literature reviews have been conducted and have been placed at appropriate points within the thesis, rather than being grouped together. Figure 4 shows the structure of the literature survey.

Figure 4: Structure of the Literature Survey

2.0 SURVEY OF EXISTING TURBINE BLADE MANUFACTURING TECHNIQUES

2.1 Literature Review

Due to the highly competitive nature of the Aircraft Engine Industry little detailed information has been published concerning the manufacturing techniques that are employed. However, a certain amount of information was found and may be divided into two areas :

i) turbine blade encapsulation techniques; and

ii) the state-of-the-art of automated blade machining.

2.1.1. Turbine blade encapsulation

Weller [2] describes the equipment developed by Fisher Gauge Limited of Canada that uses die-cast low melting point metal alloy for encapsulating the turbine blade. The principle of supporting the blade on aerofoil datum pins in a die cavity is explained.

Ediger [3] describes a process developed by Rolls-Royce Limited for encapsulating compressor blades prior to broaching. Two alloys are mentioned, the first is a bismuth-tin alloy called Cerrotru, the second a lead-antimony-tin alloy. Encapsulation is achieved by using Fisher Gauge equipment.

Graham [4] having described two manual methods of blade encapsulation again describes the Fisher Gauge equipment which is referred to as the "Injected Metal Fixturing" system.

Hayward [5] describes the alternative to the pressure injection encapsulation technique used by Fisher Gauge that utilises alloys of bismuth and tin.

2.1.2. The state of the art of automated machining

References [6], [7] and [8] describe the seven cell automated turbine blade grinding line at Rolls-Royce Derby. Reference is made to the new techniques of blade casting, and to the reductions in blade machining cycle time and increases in machining flexibility. A zinc based alloy is used for encapsulation by Fisher Gauge equipment.

2.2 Contact with Turbine Blade Manufacturers

Personal visits have been made to three Rolls-Royce sites to enable the current blade manufacturing process to be viewed and the extent of and need for automation to be assessed. The three sites were:

i) Glasgow where compressor blades are manufactured from forgings and where the only automation takes the form of dedicated transfer lines to pass the encapsulated blades through a series of broaching machines;

ii) Coventry where compressor and turbine blades are machined and where new automation is planned; and

iii) Derby where turbine blades are machined, and where two new automated creep-feed grinding lines have been installed.

In addition discussions have been held with visiting engineers from the Pratt and Whitney Aircraft Corporation of the United States of America and Kongsberg Vapen Fabrik of Norway (sub-contractors to Pratt and Whitney), concerning the present state of and future plans for blade machining automation.

2.3 Survey Conclusions

The turbine blade machining process has been successfully automated from post-encapsulation to pre-decapsulation. Whilst the use of this type of automation has not yet reached all parts of the blade machining industry, practical systems are currently in use as for example at Rolls-Royce Derby [see Section 2.1.2].

Thus the two areas that have not yet benefitted from the introduction of automation are the loading of a turbine blade into the encapsulation die prior to encapsulation, and the removal of a machined blade from the encapsulation block.

Of these two areas, the latter has not been pursued. Once a blade has been machined, accurate datum features exist on the parts of the blade that protrude from the encapsulation block (notably at the blade root). These features can be used to hold the blade whilst it is decapsulated, to handle it for further processing following decapsulation and to act as datum locations for further machining. It was thus felt that the automation of this area of the machining process would be straight-forward and did not require research work.

Thus, the area of the manufacturing process selected to be investigated in detail was the automatic loading of a turbine blade into an encapsulation die.

3.0 TURBINE BLADE HANDLING

3.1 Current Encapsulation Die Design and Operation

Referring to Figure 5, the two basic elements of a typical encapsulation die are the datum die-half (housing the fixed datum pins) and the moving die-half (housing the spring-loaded support pins). The blade to be encapsulated is manually located within the cavity on fixed datum pins and is held in place against these datum pins by the spring-loaded support pins. The encapsulating alloy is then poured into the die cavity and once the material has solidified, the die is opened and the resulting encapsulation block is removed.

A die is designed either for vertical or for horizontal blade loading. If the blade is to be loaded and clamped in a vertical orientation, the die is closed prior to blade loading. The support pins are held in a retracted position whilst the blade is manually loaded into the die and is located and manually held on the datum pins. The support pins are then released thus clamping the blade and the operator releases the blade. If the blade is to be loaded in a horizontal orientation, the die is held open during blade loading. Once the blade has been manually located on the datum pins it is released by the operator (the datum pin positions are designed so that the blade will remain in position once released). The die is then closed (with the support pins retracted) and the support pins are released so as to clamp the blade.

In order for the operator to know that a blade has been correctly orientated onto the datum pin, a contact feed-back system is provided. Each datum pin has a low voltage applied to it. When the blade is clamped by the support pins, contact of the blade with a datum pin

shorts this voltage to ground (the body of the die) and illuminates an indicator lamp on a mimic diagram. The operator can thus be sure that the blade is contacting all the datum pins. This system obviously requires that each of the the datum pins be electrically insulated from the body of the die. This electrical insulation is achieved by applying a non-conductive ceramic coating to each daatum pin and by the use of insulating spacers (Figure 6). Electrical connection to the pin is facilitated by a screw thread at the end of the pin.

The ceramic coating of datum pins is a costly and time consuming process. Each pin must first be ground with the relief for the ceramic coating, the coating is then applied and is finish ground. It is not uncommon for the ceramic coating to fail during the finish grinding process (necessitating a new coating), but not however during service.

Each datum pin is made from hardened steel and has a carbide tip brazed to its end. The carbide tip is ground relative to the pin shoulder (Figure 6) to form a hemispherical feature that forms the contact with the blade aerofoil.

Once a die is complete, the datum pins must be adjusted to achieve the correct blade location relative to the datum faces of the die cavity. This adjustment is made by a repeated process of encapsulating a blade in the die, removing the encapsulation from the die and gauging the blade position relative to the encapsulation block datum faces (using very accurate gauging jigs), and then removing and grinding the required datum pin spacers (Figure 6 item 3). This is obviously a costly and time consuming process, which must be repeated periodically

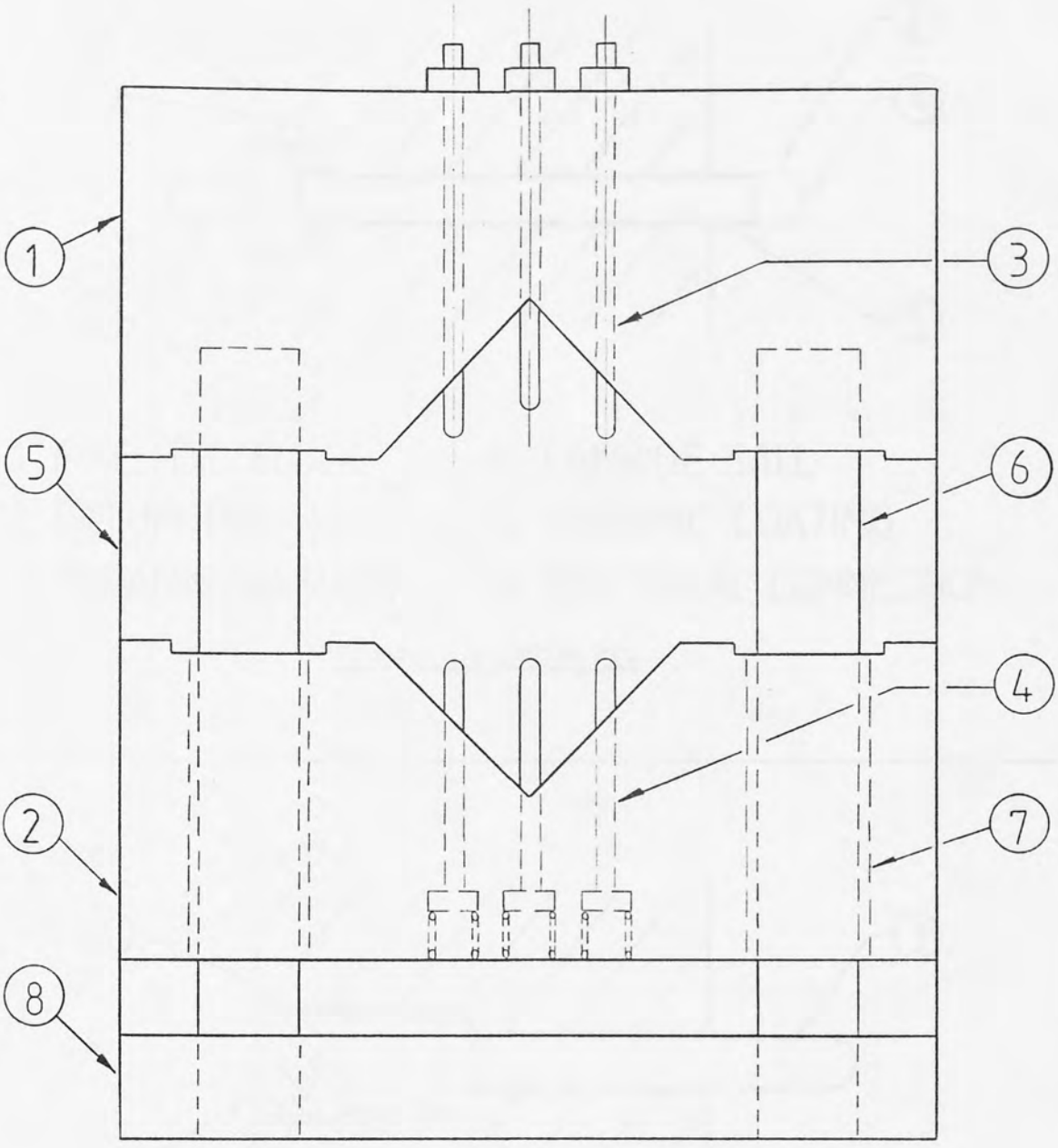
during the life of the die to take account of datum pin wear.

It is known that as the encapsulating material cools in the encapsulation die, due to metallurgical volume changes, a blade will often move away from one or more of the datum pins. Insufficient is known about the metallurgical changes that occur for accurate predictions to be made about how a particular blade will move. The adjustment of the datum pins is thus done on the basis of setting the pins in a position that, following the movement of the blade during the cooling of the encapsulation, will leave the blade in the desired position relative to the encapsulation block datum faces. The datum pins are thus not necessarily set in the theoretical positions for correct blade location.

Once a blade has been encapsulated the encapsulation block must be removed from the die. As can be seen from Figure 5, the encapsulation block faces are arranged so as to facilitate block extraction. In order to facilitate datum and support pin extraction, the pin axes are all arranged to be parallel to the direction of block extraction. This places a major contraint on the die design. The location points on a blade for datum pin contact are specified by the blade manufacturer (Figure 8). Where two datum points are close together, the need to extract the encapsulation block off the datum pins requires that either very small diameter datum pins be used, which is undesirable because of lack of pin stiffness (pin diameters are normally no less then 6mm) or one or more retractable pins be used with axes that are not parallel to the direction of block extraction (the pins are then individully retracted prior to removal of the encapsulation block), which is undesirable because of the lack of

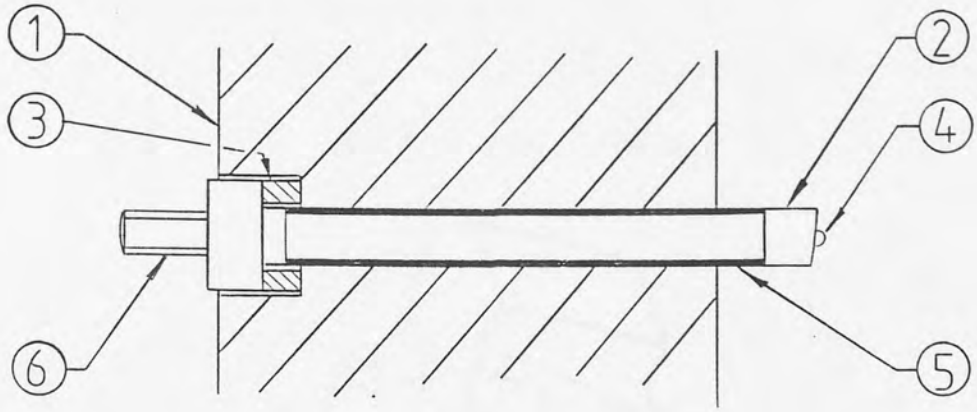
datum pin positional accuracy.

It can thus be seen that the major ccontraints that are placed upon the design of an encapsulation die (apaart from those imposed by the blade manufacturer) are the need to eleectrically insulate the datum pins from the die body, the need to proovide a means of accurately setting the datum pins in a known positiion and the need to extract the encapsulation block off the datum and support pins.



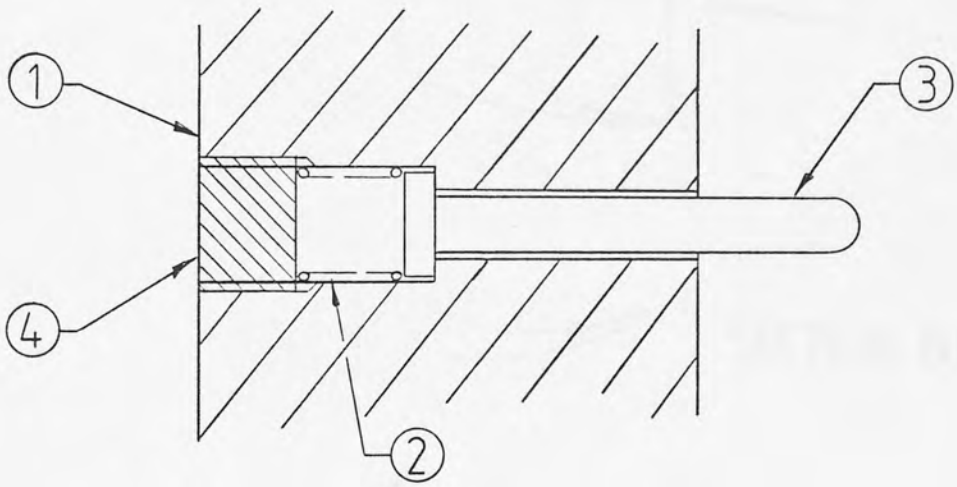
1 FIXED DIE BLOCK
2 MOVING DIE BLOCK
3 DATUM PIN
4 SPRING CLAMP PIN
5 BASE PLATE
6 GUIDE RAIL
7 LINEAR BEARING
8 END PLATE

Figure 5 : Encapsulation Die Schematic Drawing



1 FIXED DIE BLOCK
2 DATUM PIN
3 CERAMIC WASHER
4 CARBIDE BALL
5 CERAMIC COATING
6 ELECTRICAL CONNECTION

Figure 6 : Datum Pin



1 MOVING DIE BLOCK
2 COIL SPRING
3 CLAMP PIN
4 SCREW PLUG

Figure 7 : Support Pin

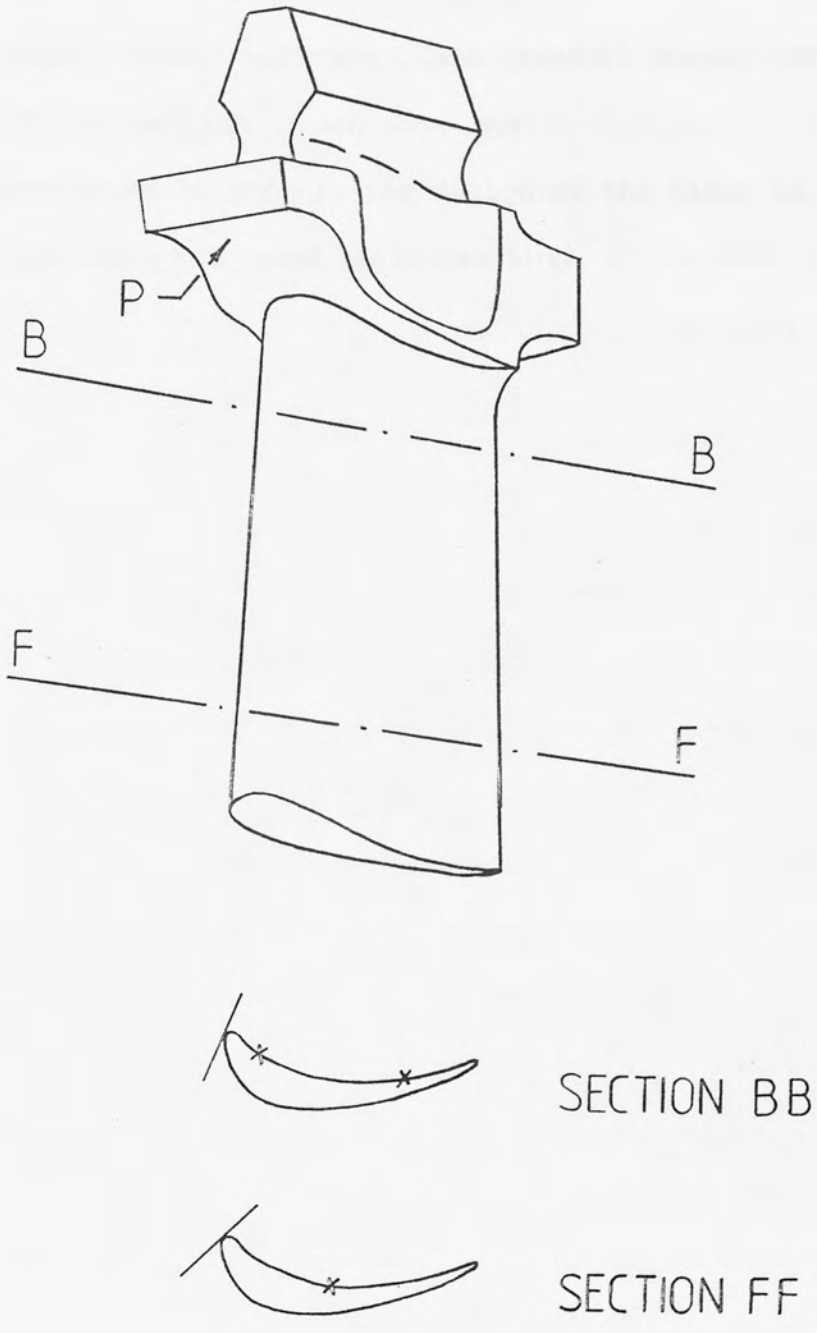


Figure 8 : Typical Location Points on a Turbine Blade

3.2 Turbine Blade Handling Constraints

Following discussions with manufacturers of turbine blades the following blade handling constraints were established;

i) that any physical contact with the blade aerofoil should not mark the surface of the aerofoil in any way; and

ii) that no changes could be made to the design of the blade in order to ease or facilitate automated blade handling.

3.3 Discussion of Die-Loading Techniques

Having established how manually loaded encapsulation dies are currently designed and operated, together with the contraints that apply both to die design and use and the handling of turbine blades, it is now possible to consider how a fully automated die loading system would operate. There are two inter-linked elements to such a system, namely the blade delivery and loading device and the encapsulation die. It has not been assumed that the current design of manually loaded encapsulation die is necessarily suitable for inclusion in an automated system.

The selection of the type of automated delivery device was between a dedicated pick-and-place unit and a handling robot. From the outset it was considered that the inherent inflexibility of pick-and-place units would be an undesirable contraint on the development of the die-loading system. Eventhough it was accepted that the final proposed system might make use of a pick-and-place unit for blade delivery and loading it was decided to use a handling robot during the project so as to give maximum flexibility during the period of research.

3.3.1 Die-loading options

Four options are considered. In each case it is assumed that the blade is initially positioned in the die using a robot which has a positional accuracy that is lower than that required for the final positioning of the blade in the die. The four options are:

a) A fully automatic die and a standard robot

The sequence of die loading would be:

i) the robot places the blade into the open die;

ii) the die closes with both the datum and support pins being automatically adjusted so as to clamp the blade without stressing;

iii) the robot releases the blade and withdraws; and

iv) the die pins are automatically adjusted so as to move the blade into its final accurate position, again without stressing the blade.

This option uses a standard robot with no additions apart from a suitable gripper. The mechanism for the final positioning of the blade is included in the die, which is undesirable for two reasons: first, the accurate pin positioning mechanism must operate in a hostile environment (including close proximity to molten metal) and secondly, the positioning mechanism must be included in every die, even though the robot may be serving up to four dies. It should however be possible for a single positioning mechanism controller to control all the dies on an encapsulation machine since only one die is being loaded at any given time.

The orientation of the blade into its final position would be a complex operation in order to prevent stressing of the blade caused by unco-ordinated movements of multiple clamping points. It might be necessary to provide additional pins to provide the necessary degrees of freedom for positioning.

Movement of a die pin along its axis would be a relatively straight forward operation, however, any lateral movement of the pin would be difficult to achieve (this applies particularly to the P-pin see Figure 8).

The pin positioning mechanism must have high repeatability and accuracy, as the die datum pins are no longer static. Provided that the position of the axes of the aerofoil datum and support pins were carefully selected, it should be possible to gauge the thickness of the aerofoil section and to adjust the datum position of the die pins to allow for the thickness of the aerofoil. This in-process gauging of blades and datum position adjustment could lead to improvements in blade manufacture and engine performance. In addition it might be possible to improve on the present method of datum pin setting (Section 3.1).

b) A standard die with a robot mounted micro-positioning head

The sequence of die loading would be:

i) the robot places the blade into the open die;

ii) a micro-positioning head which is attached to the robot wrist and which holds the blade, is used to accurately position the blade against the die datum pins;

iii) the die closes and clamps the blade;

iv) the micro-positioning mechanism releases the blade and the robot withdraws.

This option uses a standard robot with the addition of a suitable micro-positioning head and has the advantage that the current design of encapsulation die could be used. Since the main robot axes are not

available for blade orientation within the die (as already stated the robot positional accuracy is insufficient for this work), the micro-positioning head would have to be an extremely complex and accurate five or six degree of freedom device. In addition provision would have to be made for the micro-positioning head to pick-up a three dimensional datum on the encapsulation die, in relation to which the blade would be orientated.

c) A standard die, a robot mounted micro-positioning head and a "dummy" die

The sequence of die loading would be:

i) the robot places the blade into the "dummy" die;

ii) a micro-positioning head attached to the wrist of the robot and which actually holds the blade, is used to position the blade against the "dummy" die datum pins, using the sensors of the pins for feed-back;

iii) the robot transfers the correctly oriented blade into the actual die;

iv) the die closes and clamps the blade;

v) the micro-positioning mechanism releases the blade and the robot withdraws.

As with option (b), a standard robot is used with a suitable micro-positioning head together with the existing design of die.

With this option there is scope for additional sensing of the blade, other than electrical contact with the die pins (for example force feed-back) and for a pin layout in the "dummy" die to be used that is more suited to orientating the blade other than that used for clamping

in the encapsulation die. A drawback to this system is that the robot must have a high positional accuracy in order to transfer the accurately orientated blade from the "dummy" into the encapsulation die. A typical positional repeatability and accuracy of +/- 0.1 mm would not be sufficient for this transfer operation.

Since the blade orientating is done externally to the encapsulation die the encapsulation cycle time would not be extended.

d) A standard die and a standard robot using compliance

The problem of robot inaccuracy in automated assembly has been investigated by a number of researchers (Section 3.4). One solution to the problem of using a robot that is not sufficiently accurate to locate one part relative to another, is to introduce a compliant element between the robot and the part that the robot is being used to locate. This compliant element then allows the errors in robot position to be either actively or passively accommodated and facilitates parts mating to a degree of accuracy that is greater than the positional accuracy of the robot.

In this instance a compliant element could be introduced between the robot and the blade so that the positional errors of the robot could be accommodated during the die loading. A literature search (Section 3.4) was carried out in order to examine how this could be achieved.

3.4 Literature Review

As shown in Figure 4, this area of the Literature Review was designed to cover relevant areas of the fields of automatic assembly and automatic gauging. No relevant published work was found for the latter area. Within the field of automatic assembly the literature searches were concentrated on the area of system error measurement and compensation.

3.4.1 Overview

In their summary of the state-of-the-art in programmable assembly systems Abraham, Stewart and Shun [9] review the work carried out on force-sensing in assembly operations up to 1977. Four main areas of work are cited :

i) remote sensing of reaction forces at the manipulator joint drives;

ii) sensing of reaction forces on a work-table or pedestal;

iii) wrist force sensing; and

iv) passive compliance.

It is reported that the first method of force sensing is suitable only for very coarse assembly force measurement due to unknown frictional forces at the manipulator joints and variable compliance of the manipulator and its load.

Wrist force sensing involves measurement of the reaction forces at the wrist produced by contact of a tool, part or gripper with a part or work surface. These measurements are used in a force feedback control strategy based on active accomodation.

One problem with the active accommodation type of force sensing device is the need for highly accurate and fast response position servo devices that act on the information that is taken from the force sensors. This is what lead to the development of simpler passive compliance devices.

Passive compliance is an economical and effective method for accommodating position and alignment errors in parts assembly. An example of a passive compliance device is the Remote Centre Compliance device (RCC). This kind of open loop control is particularly suitable for peg-in-a-hole type assembly operations.

Van Brussel & Simons [18] define passive and active accommodation of assembly forces in the following way:

> accommodation is defined as the process in which the contact forces between parts held by the manipulator and the environment modify their relative position or motion;
>
> passive accommodation is said to occur when the positioning command signal remains unchanged as a result of resistive forces, with only the response of the manipulator being altered by the contact forces that occur between the part being held and its environment;
>
> active accommodation is said to occur when the positioning command signal is itself modified by feedback from the contact forces.

Active accommodation is said to be achievable using classical servo algorithms or by using adaptive control algorithms based on decision making and learning. A high degree of flexibility is provided since the original path of the manipulator may be changed on-line. However, the fine correction motions required by an active accommodation strategy are said to be outside the dexterity of currently available robots.

Passive accommodation provides a simple means of overcoming resistive forces by utilising the inherent compliance of the manipulator wrist or devices attached to the wrist (such as the RCC), however, the compliant structure must be designed to suit a given application thus limiting flexibility and applicability.

Simunovic [22] considers both the required accuracy of robots for assembly tasks and how the necessary accuracy may be obtained.

The assembly process is regarded as having two district phases:

i) part positioning or transition which comprises large, coarse movements of the parts in free space; and
ii) parts mating which comprises small movements of the parts within the clearance between the parts.

It is suggested that for assembly work a robot must have a positional accuracy that is at least four times smaller than its total range of movement.

Part positioning is presented as being a function of positional

control only, requiring a robot with a high positional resolution.

Once positioned, parts may be mated either using active force-feedback (again requiring a high resolution robot capable of fine motions) or passive compliance. Both force feed-back and compliance replace the need for a robot of high positional accuracy although fine motion is still required in the former.

3.4.2 Force measurement and active accommodation

Nevins and Whitney [11] describe a compliant six degrees of freedom wrist force sensor device together with the transformation matrix necessary to resolve the transducer readings into the required force values. A force nulling strategy is proposed as an active accommodation strategy for inserting a peg into a hole.

A pedestal force sensor and a wrist force sensor (both compliant six degrees of freedom devices) are described by Watson and Drake [10]. The transfer function matrix necessary to transform the transducer signals to the desired force readings is described.

Simunovic [12] presents a mathematical analysis of the forces and moments generated during the insertion of a round peg into a round hole using the pedestal force sensor described in [10].

Goto, Inoyama and Takeyasu [14] describe a two robot assembly system that is suitable for peg-in-a-hole type insertion assembly, which utilises a simple flexible wrist mechanism together with wrist force sensing to facilitate active accommodation during the insertion operation. As in the work of Nevis and Whitney [11] a search and

force nulling strategy is used.

Van Brussel and Simons [13] summarise the two main approaches currently used in automated assembly. Referring to [11] they state that the force nulling strategy for the insertion operation is not applicable in many assembly tasks. With reference to [12] it is noted that any nulling strategy should be referred to the top of the peg that is being inserted. Active accommodation systems using for example the diagonal matrix of the nulling strategy are described as having "software compliance". Two passive accommodation devices are referred to namely the Hitachi Hi-T-Hand [14] and the Remote Centre Compliance device [15]. The RCC is described as a hardware realisation of the nulling strategy referred to the top of the peg.

A flexible assembly test system is described which includes a five degree of freedom wrist and a six degree of freedom force sensor. A two dimensional peg-into-a-hole assembly experiment is described and the idea of a two phase assembly operation is introduced : the approach phase and the insertion phase.

It is stated that simple force feedback as used in the nulling strategy cannot always be applied since in some circumstances ambiguity can occurr. Thus, the active accommodation system must include strategies (e.g. search strategies) for resolving ambiguous situations such as jamming of a peg in a hole.

Hirzuger and Brunet [21] describe the application of the Deutsche Forschungs-und Versuchansalt fur Luft-Und Raumfahrt (DFVLR) three axis force-torque sensor to assembly operations. The use of this sensor for

both force-torque feedback during assembly and for teaching robot initial values of forces and torques to be applied during assembly is reported. In addition the self-learning capability of the assembly system is described. It is reported that the use of the parallel interfaces and sixteen bit computers is sufficient to make such a system practical in real-time control.

3.4.3 Compliant devices

Drake, Watson and Simunovic [15] describe a Remote Centre Compliance device that was used without force sensing or feedback control in an assembly operation.

Van Brussel and Simons [16] discuss the limitations of both active accommodation and passive compliance devices. It is stated that the processing time required by active accommodation devices such as described in [10] is too long for practical use in real time control since complicated control algorithms are needed in order to interpret the force sensor feedback. This lead to the development of passive compliant devices such as described in [15] but these are generally non-universal as they must be designed for a particular type of assembly operation and are often linked to the dimensions of the parts concerned.

Thus a five degree of freedom active adaptable compliant wrist (AACW) was developed. The programmable compliance resolves the problem of ambiguous force feed back situations such as jamming of a peg in a hole, which normally require complex algorithms based on search techniques to resolve them. However, programmable compliance is still insufficient in order to deal with all situations, thus active

accommodation is also provided in the wrist.

A three dimensional assembly operation of a peg into a hole is presented as an example.

The concept of self-learning for the wrist is introduced, where the wrist is considered as a stochastic learning automaton in order to optimise the controlled compliance of the wrist.

Van Brussel and Simons [17] having described the AACW report the continuation of their work of [16] by giving a detailed description of the self-learning techniques used to optimise the insertion operation of a peg in a hole.

Whitney and Nevins [19] describe the Remote Centre Compliance (RCC) device and list typical applications. The RCC is an entirely mechanical device that passively accommodates angular and positional misalignments between parts during assembly. It may be defined as an engineered compliance whose behaviour is known and sufficiently repeatible. The object being assembled is compliantly suspended from its tip (the point where engagement with the mating part occurs) in such a way that angular and lateral error are absorbed independently.

Stoyanov and Ivanov [20] present a geometrical analysis of the RCC device and contrary to previous researchers state that angular and lateral errors are not absorbed independently by the device. It is stated that up to 50% parasitic reaction can occur when either a pure lateral or a pure angular force is applied. Of the two cases it is stated that parasitic angular reaction to a pure lateral force is the

least desirable as this leads to two point contact between the mating parts rather than the line contact caused by the other case. It is proposed that this worst case could be reduced in RCC design by incorporating rotational members of greater rigidity and translational members of increased elasticity.

3.5 Die Design and Die Loading Proposals

The conclusions drawn from the preceding literature survey were:

i) the positional errors of the blade relative to the datum pins in the die could not be corrected using active accommodation with the type of robot that would be employed; and

ii) the types of passive compliance devices described could not reproduce the dexterity of the human operator and would be particular to one design of blade. This would necessitate the design and manufacture of such a device to suit each encapsulation die design.

It was furthur concluded that if any positioning mechanism were to be included in the die-loading system (in addition to the robot), that this mechanism should form part of the encapsulation die itself. By including a fine positioning mechanism within the die, rather than in the form of an external micro-positioning head or a compliant device for holding the blade, it would be possible both to ease some of the existing constraints on encapsulation die design and to introduce the possibility of automatic blade gauging and datum correction. The two present constraints on encapsulation die design that might be eased in this way are the method of setting the datum position of the pins and the need to have all pin axes in parallel to facilitate encapsulation block removal(see Section 3.1).

Since the final positioning mechanism is to be included in every die, and one robot might serve up to four dies on one encapsulation machine, it will be necessary for the positioning system to be produced as economically as possible so that the encapsulation system

cost is not greatly increased.

It is thus proposed that a standard robot be used for blade loading and that the die be equipped with datum pins that are capable of movement under direct computer control. In addition the spring-loaded support pins of the die are to be fitted with displacement transducers so that the blade thickness can be gauged and the normal datum pin positions adjusted to allow for the blade tolerance. Since the die-pins will move in to clamp the blade whilst it is still held by the robot, it may prove necessary to include some sort of simple compliant element between the robot wrist and the blade to prevent stressing of the blade prior to the release of the blade by the robot.

4.0 ROBOT MECHANICAL DESIGN

4.1 Literature Review

The literature searches that were carried out revealed no published information relating to the physical construction of robot manipulators. Three areas of published work were revealed : the theoretical dynamic performance of robot manipulator structures, the choice of drive systems for robots, and methods for the measurement of robot positional error.

4.1.1 Manipulator dynamic performance

Burckhardt and Helms [23] propose that any manipulator be designed so that the natural frequency of vibration of the structure has a minimum value, such that the maximum amplitude of vibration following constant deceleration to rest, is equal to the desired positional accuracy of the robot. The idea of a Figure of Merit (Q) is introduced for the comparison of manipulator performance. Q is defined as the product of the largest excursion of a degree of freedom of the manipulator and the natural frequency of vibration of that degree of freedom. Since the use of constant deceleration to rest makes high speed movement and high positional accuracy mutually exclusive, a Gaussian velocity distribution is proposed as the ideal, with a simplified distribution using constant acceleration but smoothed deceleration being put forward as a practical approximation to the ideal.

Demaurex and Gerelle [24] consider the theoretical structural design of a revolute, precision assembly robot. The stiffness density (a) and the angular natural frequency (w) are defined (a manipulator is said to be coherent if these two parameters are the same for each

link of the manipulator structure) and are combined with the Figure of Merit (Q) to give an expression for the dimensionless Balance Factor (b). Once the lengths of each of the links has been fixed, b and Q may be used to calculate both a and w and hence the mass and stiffness of each link. The required stiffness is then used to select the required drive for each link. An example is given.

Burckhardt and Gerelle [25] summarise the work that has been done in this area and define a second Figure of Merit which relates to the frequency of oscillation of the system during gross motion (given by the time to perform one radian of movement of the load vector). The paper then presents a design analysis of the Trallfa TR3000 W robot.

4.1.2. Choice of drive system

The three drive systems that are available for robot actuation are pneumatic, hydraulic and electric. Much has been written about the comparison of pneumatic and hydraulic systems and both Boulder [26] and Metzer [27] provide good summaries of this debate.

In general, hydraulic systems are superior when the requirement is for large forces over long distances and precise control is required. Pneumatic systems are better for low force, fast response actuation and for shock free systems. Hydraulic systems are characterised by operating at high pressures (typically 2000 psi) and are therefore more compact for a given force output than pneumatic systems that operate typically at 125 psi. One significant feature of pneumatic systems is the compressibility of the air. This is an advantage when a shock-free or energy storing system is required, but is a disadvantage when precise motion

control is required.

Archer and Blenkinsop [28] compare electric and hydraulic drives for the actuation of robots. The merits and demerits of both types of drive are listed and discussed together with the specific requirements of robot actuation. The conclusion is drawn that there are no strong technical reasons for selecting one system or the other (unless the operating environment of the robot precludes certain types of drive, as in the case of paint spraying robots precluding the use of electric drives). However, system cost is an important factor to be considered. A graph of system cost against robot payload for the two types of drive is presented. Whilst the weight (and hence cost) of electric drive systems increases with robot payload in an approximately linear manner from the origin, the cost of a small hydraulic system starts at a much higher level than for an electric system, but the curve is much flatter as robot payload increases. This is because the servo-valves constitute a significant proportion of the overall cost of a hydraulic system and the cost of these is relatively independent of the volume of flow. The two curves cross at a point approximating to a system cost of £40,000 and a robot payload of 35 kg. Below 35 kg payload capacity the electric drive system is cheaper, above this capacity the hydraulic system is cheaper.

4.1.3. Measurement of robot error

Inagaki [30] and Ho [31] both discuss the sources of robot positional error and attempt to define measurements of such error.

Inagaki reports on the draft standard prepared by the Japanese

Industrial Robot Association that is an attempt to standardise the measures of robot accuracy.

Considering a point-to-point (PTP) teach and play-back type robot, it is stated that there are three categories of point : the desired working point, the taught point and the played-back point. Two measures of accuracy are defined : the playback accuracy is given by the difference between the taught point and the mean of the played-back points, and the positional accuracy is given by the difference between the desired point and the mean of the played-back points. The repeatability of the robot is given as the distribution of the played-back points.

Ho defines the positional accuracy of a robot as the mean of the errors measured between the desired point and the played-back points. The robot precision is said to be equal to the variance of the positional errors. These two measures of robot error are independent. It is stated that high positional accuracy can be achieved by using a proper robot calibration procedure. A theoretical analysis of the system errors generated in an insertion operation is presented, as an example of how the accuracy of a proposed system may be examined prior to system construction.

Morgan [32] and McEntire [33] both present practical methods for measuring robot positional accuracy.

Morgan states that any measure of robot accuracy must be related to the robots working volume. A method of Volumetric Accuracy Mapping (VAM) is proposed whereby the robot working volume is divided into a

number of cube-like elements and the accuracy of the robot measured at each corner node. The mean accuracy measured from three different approach directions was calculated from readings taken from a robot mounted clock-gauge acting on a flat plate located at each node. The results may be presented in tabular form for each major plane of the robot working volume, or plotted as an accuracy distribution curve showing where the robot approaches or exceeds its quoted tolerances.

McEntire presents a method for measuring robot positional accuracy and repeatability that requires the robot to position a part-cube (three mutually perpendicular faces) to a jig of clock gauges positioned at the taught point. A mathematical analysis is presented that relates the gauge readings to the positional and orientation vectors of the cube (and hence the robot). An analysis to derive the error vectors is also provided.

4.2 Introduction

The robot that is to be designed and developed as part of this project will be required to fulfill two requirements:

i) to enable a method of automation of the loading of an encapsulation die to be developed and demonstrated; and

ii) to serve as a prototype machine from which a production robot could be developed and manufactured.

As stated in Section 3.5, the proposed system for encapsulation die loading will comprise a standard robot and an "intelligent" die. The robot that will be designed will thus be a so-called "first generation" robot, utilising existing technology. This will constitute the prototype of a production robot, to be called the "Cybermate".

4.3 The Design Philosophy

The proposed major application area of the Cybermate robot is machine loading. With the increasing use of machining cells, as stand alone units or as a part of larger automated manufacturing systems, and in particular as interest in Flexible Manufacturing System (FMS) grows, the market for robots capable of operating within such machining cells is seen to be a potential growth area. This is the justification for the development of the prototype robot.

The normal configuration for such a cell is a circle of machine tools, gauging stations, cleaning stations and input/output stations (Figure 9). The robot operates from the centre of the circular cell, working with a horizontal radial action. The diameter of the cell must be large enough to allow a sufficient number of machine tools to be included, but not so large as to give a high cell cycle time or to require a robot with a very long horizontal reach. The two main characteristics required of the robot are thus a long horizontal reach and a base rotation of 360 degrees.

Of the four main robot structural configurations (Figure 10) cylindrical geometry is the most suitable for this application area and has been used in the robot design. It is this feature that is the main justification for marketing a new industrial robot. Many robots available today are designed for flexibility of application and consequently employ the anthropomorphic geometry. These robots are complex and thus expensive. As stated by Salmon [29] the control of a revolute robot is complex due to the non-linear positional errors in the rotating joints (the errors being dependent on the position of the robot in the work space) and the differing inertia of the robot arm

(again dependent on the position of the arm). By limiting the proposed application areas the required flexibility of the Cybermate robot is reduced, resulting in a simple and thus less expensive machine. Whilst in some cases itt is arguable whether the robot purchase price is a significant facctor in customer selection (since the robot may constitute only a small proportion of the associated system cost), robot reliability and eease of maintenance is significant in all cases. Simplicity of the robot design must therefore be attractive to potential users and is a key feature of the design.

The robot has also been designed on a modular basis so that the minimum number of axes required ffor a given application may be supplied, thus minimising the installlation cost.

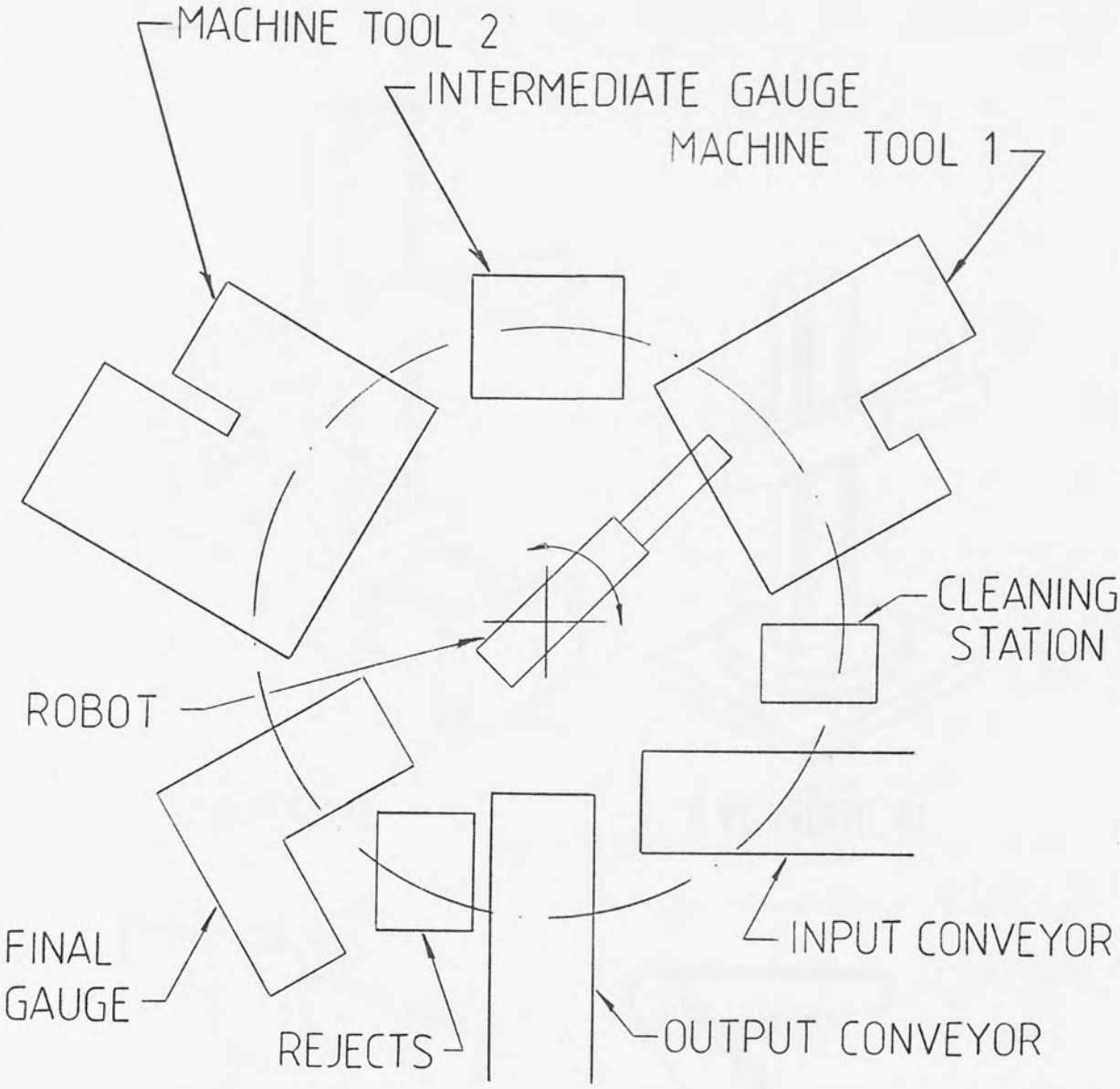
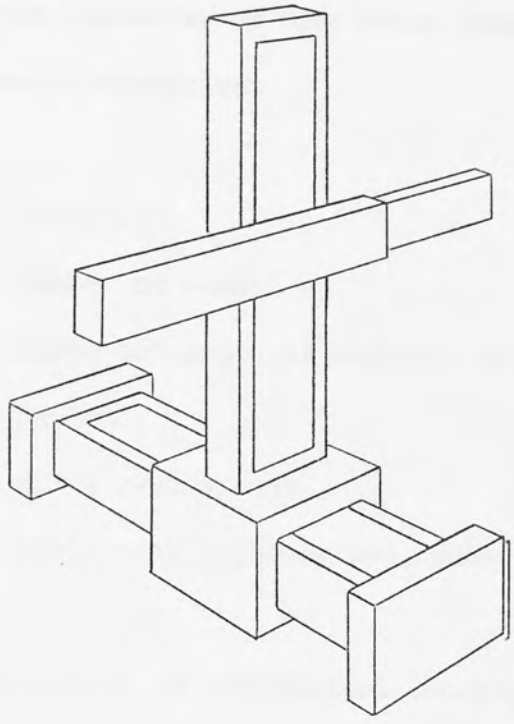
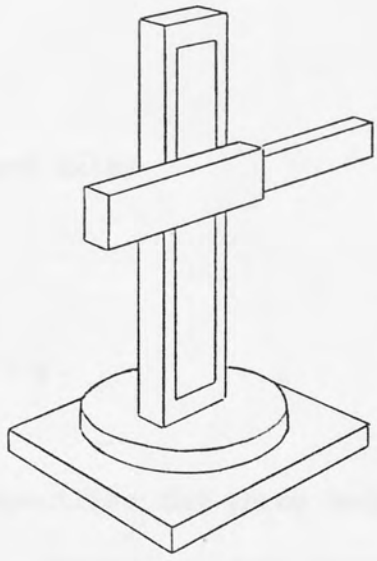


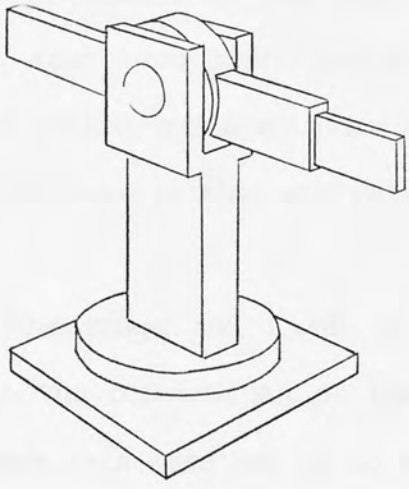
Figure 9 : A Typical Robot Machining Cell



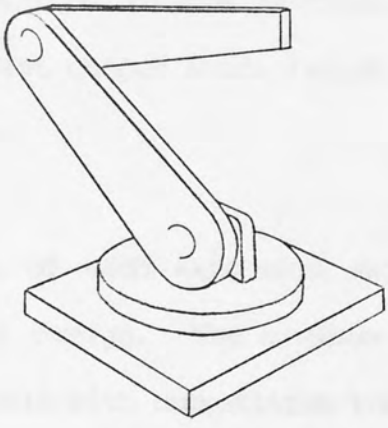
CARTESIAN



CYLINDRICAL



POLAR



ANTHROPOMORPHIC

Figure 10 : Robot Arm Configurations

4.4 Design Specification

The specification of the robot design may be broken down into the following categories:

i) geometry;
ii) number of axes;
iii) range and speed of movement for each axis;
iv) payload;
v) drive system; and
vi) positional accuracy and repeatability.

The selection of cylindrical geometry specifies the three main axes which are vertical lift, horizontal extension and horizontal rotation. For the applications envisaged it is necessary to add wrist movements at the end of the extending horizontal arm. Two wrist axes have been included: a rotation in the vertical plane (wrist pitch) and a rotation of the wrist output shaft (wrist roll). The Cybermate is thus a five axis robot.

Both the range and speed of movement of each axis were maximised within the constraints of the physical design. The minimum values for each axis were set as to be comparable with competitive robots. Payload, and positional repeatability were chosen to meet the envisaged applications and to be comparable with competitive robots.

An electrical drive system was selected in preference to a pneumatic or hydraulic system. Pneumatic actuation was rejected because of the lack of controllability associated with pneumatic drives. Hydraulic drives were rejected on the basis of cost and cleanliness and since

the relatively low payload of the robot did not justify a hydraulic system (see Section 4.1.2).

4.5 The Detail Design

For the purposes of design the robot was divided into four parts: the horizontal arm, the vertical column, the wrist and the rotary base. The design work was carried out in the above order.

Two constraints were imposed on the detail design:

i) that it must meet the design specification; and

ii) that (i) should be achieved at minimum cost.

Whilst these constraints appear to be self-evident it is important to state them here. In particular (ii) was a major influence in the detail design work, both in the selection of bought-out proprietary items and in designing components for in-house manufacture. The following design method was used for each of the four parts of the robot design:

i) theoretical analysis of structural requirements;

ii) determination of drive system;

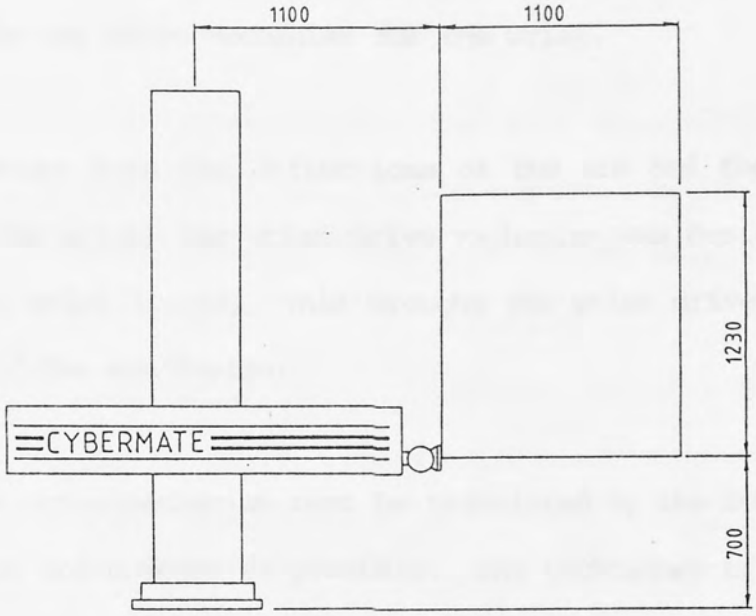
iii) preparation of alternative design schemes;

iv) selection and optimisation of most suitable design scheme; and

v) detail design of components, and selection of bought-out items.

Figure 11 shows the basic dimensions of the robot.

TECHNICAL SPECIFICATION



TECHNICAL SPECIFICATION

Cylindrical geometry - five degrees of freedom.

Main Axes

Horizontal Arm Extension:	2.2m max. 1.1m min.
Horizontal Arm Rotation:	+/- 180deg.
Vertical Column Movement:	1930 max. 700 min.

Wrist Axes

Wrist Pitch:	+/- 90deg.
Wrist Roll:	+/- 180deg.

Payload Capacity:	15Kg
Repeatability:	+/- 0.1mm

All electric dc - servo drive.

Point-to-Point and Continuous Path programming modes available.

Figure 11: Specification of the Cybermate Robot

5.0 ROBOT ARM

5.1 Design Schemes

The robot arm is required to fulfil two functions:

i) to support and translate the robot wrist horizontally; and

ii) to provide the drive mechanism for the wrist.

In order to reduce both the deflections of the arm and the physical dimensions of the wrist, the wrist drive mechanism was designed to be remote from the wrist itself. This brought the wrist drive mechanism into the area of the arm design.

Since the wrist drive mechanism must be translated by the robot arm it must be as light and compact as possible. The techniques of providing remote independent drives for a two axis wrist are well known (Figure 12 (a) and (b)). Both of these methods of providing a wrist drive mechanism would require the robot arm to have an unacceptably large area of cross-section in order to accommodate the two wrist drive motors. It was thus decided that both of the wrist axes would be driven by a common motor and drive-shaft (Figure 12 (c)). The method of operation of the wrist drive is described in Section 8.2.

Figure 13 shows the basic requirements for the structure of the robot arm. The three basic elements of the arm are the static structure, the translating structure and the bearing support. The static structure (which is attached to the robot column) houses the motor and ballscrew assembly that provides the means to drive the translating structure. The translating structure houses the motor and drive shaft assembly that provides the drive for the robot wrist. The bearing support provides the method of attachment of the translating structure

to the static structure. The figure shows that the wrist drive axis and the translation drive axis are offset and parallel. In order to keep the overall cross-section of the robot arm as small as possible this offset of the drive axes was constrained to be a minimum.

Figure 14 shows the two design options that were considered for the translating structure and associated bearing support. Option (a) uses two circular section tie-bars to form the rigid structure that supports the wrist drive shaft. This structure could be mounted vertically as shown, or horizontally. Option (b) uses a tubular structure to support the wrist drive shaft. A square section tube is shown, a circular section tube was also considered as an alternative. Both options show two point bearing support (points A and B) in order to support the translating arm structure as a built-in cantilever.

The maximum cross-section dimension (w) is shown for both design options in Figure 14. For a given value of w the tubular section of option (b) provides a stiffer structure than that of option (a), thus design option (a) was rejected. A circular cross-section was selected for option (b) since linear bearings and tubular shafting of this section are readily available as standard items whereas the square section shaft would have had to be specially manufactured and the bearings built up from a number of flat linear bearings.

Figure 14 (b) shows the two bearing support points A and B. In order to maximise the rigidity of the arm structure and in order to minimise the possibility of the two bearings "fighting" one another, it is desirable for the dimension L to be as large as possible. In order to achieve this and in order to minimise the dimension P (non-productive

arm stroke) the rear bearing point A was moved as shown in Figure 15.

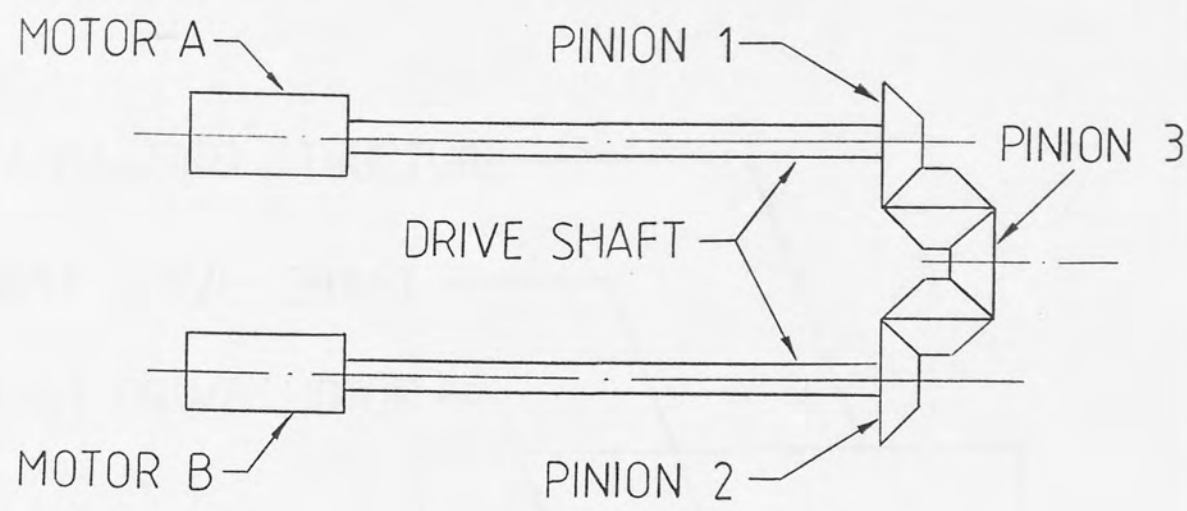
Referring to Figure 13, having determined the translating structure and the method of bearing support, the nature of the static structure could be considered.

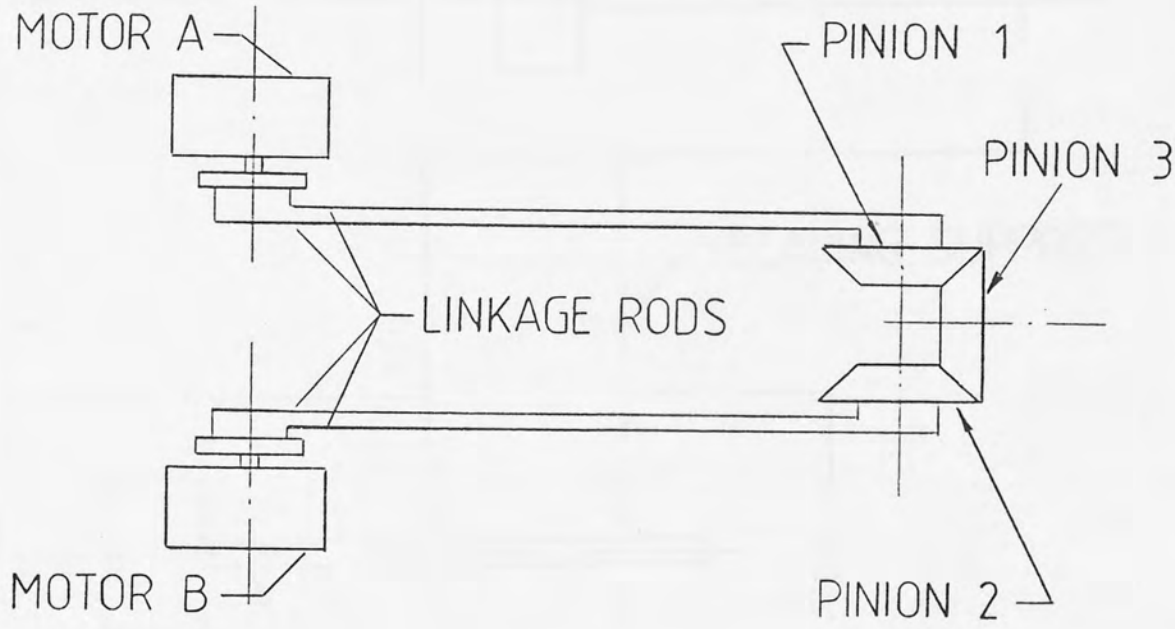
Four options were considered for the design of the static structure of the robot arm (Figure 16). As with the translating arm structure, for a given value of w (the maximum height of the structure cross-section), a tubular or box section gives greater stiffness than the tie-bar arrangement.

Two types of linear recirculating ball bearing were considered to provide the support at bearing point A (Figure 15), and are shown in Figure 17. The circular type of linear bearing was rejected since the accuracy of the position of the bearing as it translates along the bearing shaft is dependent only on the straightness of the shaft itself. At this stage in the design of the robot it was intended that the robot should be capable of continuous path (CP) motion as well as point-to-point (PTP). When considering CP motion the lateral error of the bearing position due to lack of shaft straightness becomes important. Since a tie-bar structure for the robot arm would require the use of this type of bearing, and because of the comparative lack of stiffness, the tie-bar structure option was rejected (Figure 16c). The two flat track bearing types both rely on the straightness of the machined track location for positional accuracy. It is thus necessary to machine a track bed along the length of the static arm structure. The two tubular structure options (box and circular section) shown in Figure (16 (a) and (b)) were thus rejected since such a machining

operation was deemed impractical.

The static structure of the robot arm was therefore designed on the basis of an H-channel section, (Figure 16 (d)) a section that has good stiffness properties and that allows access for all forseeable machining operations.





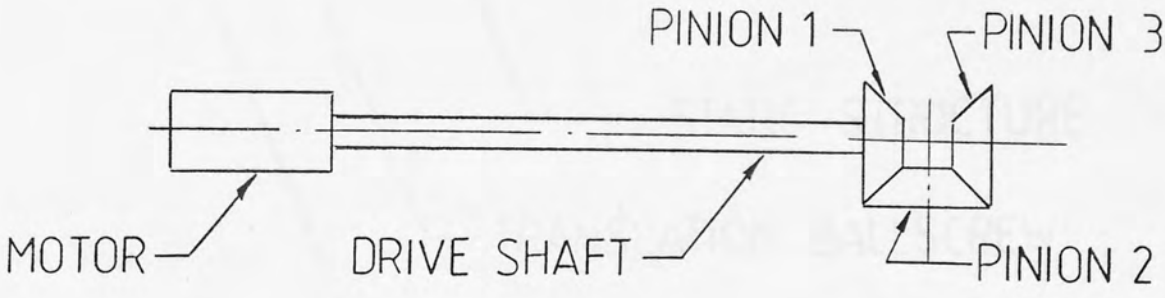


Figure 12 : Remote Drive Systems

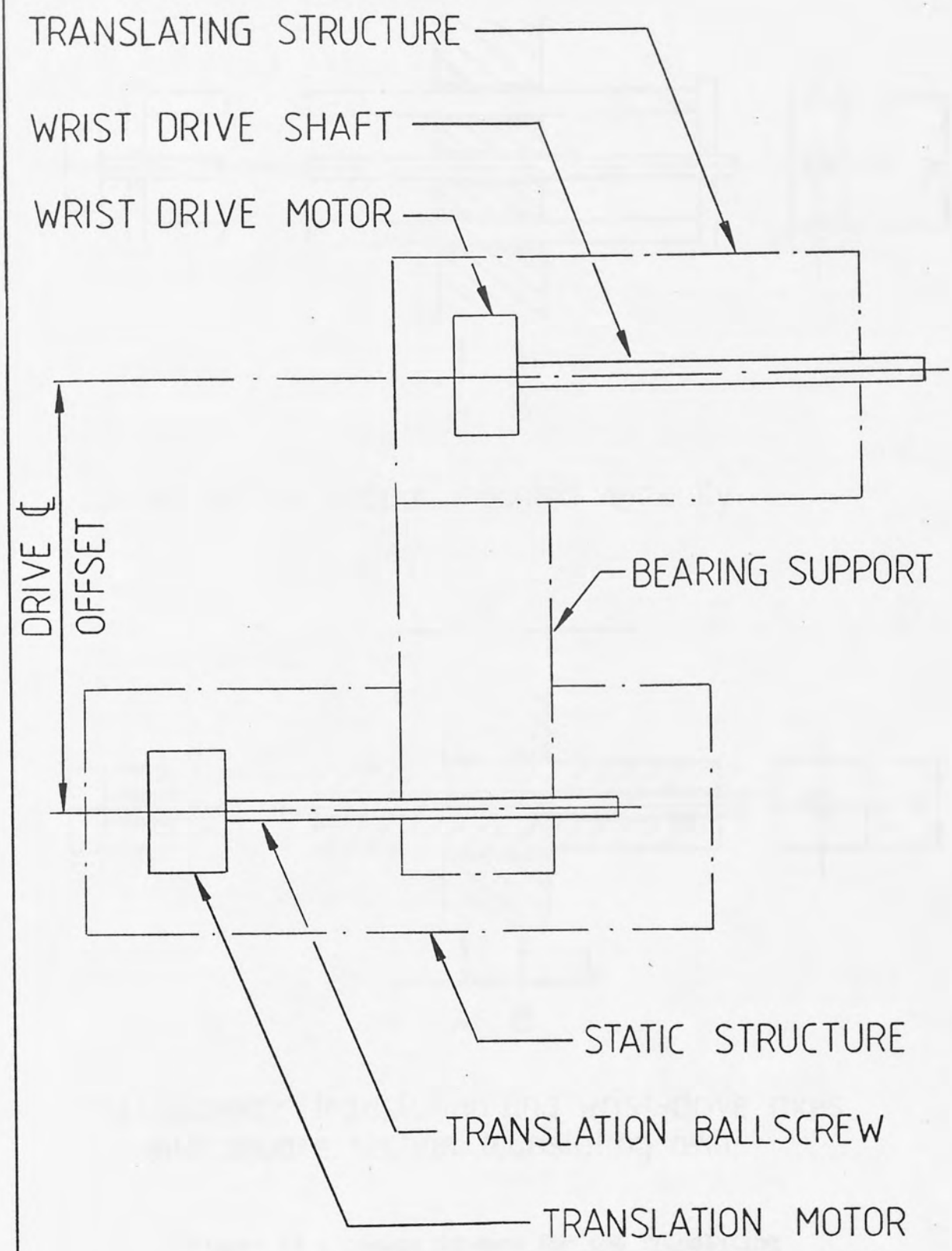
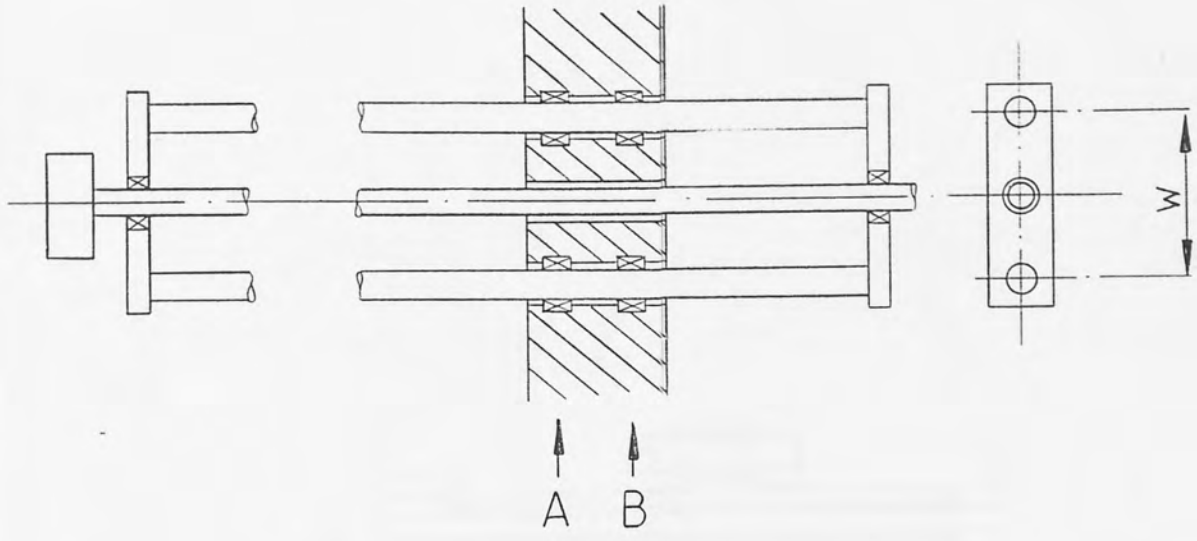
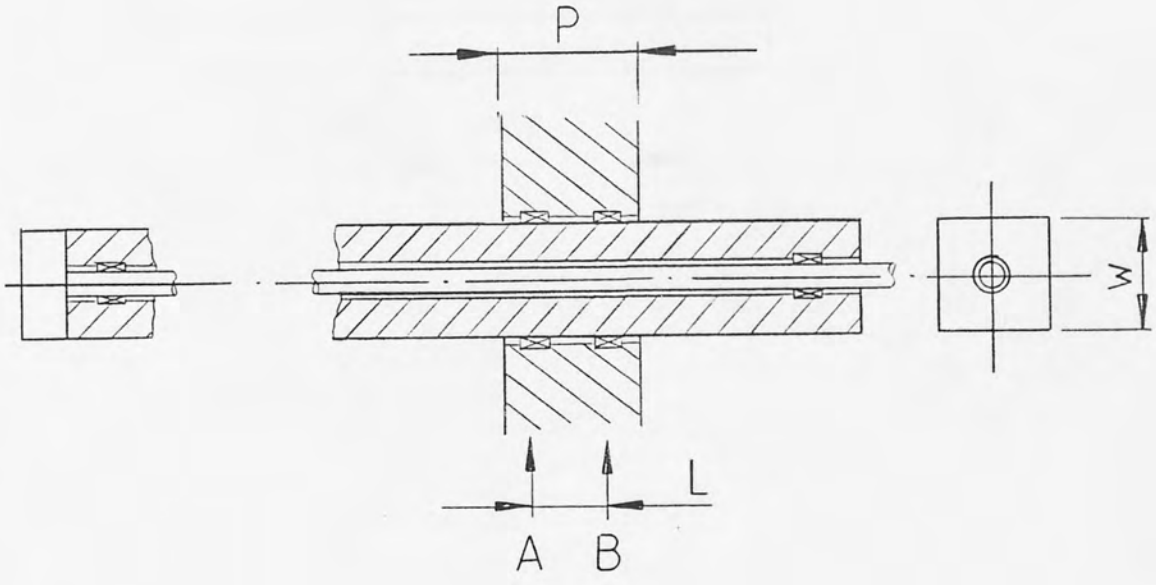


Figure 13 : Schematic Layout of Robot Arm Structure



(a) Two tie-bar support mounted vertically



(b) Concentric translation and wrist-drive axes with square section translating arm

Figure 14 : Design Schemes for the Translating Structure of the Robot Arm

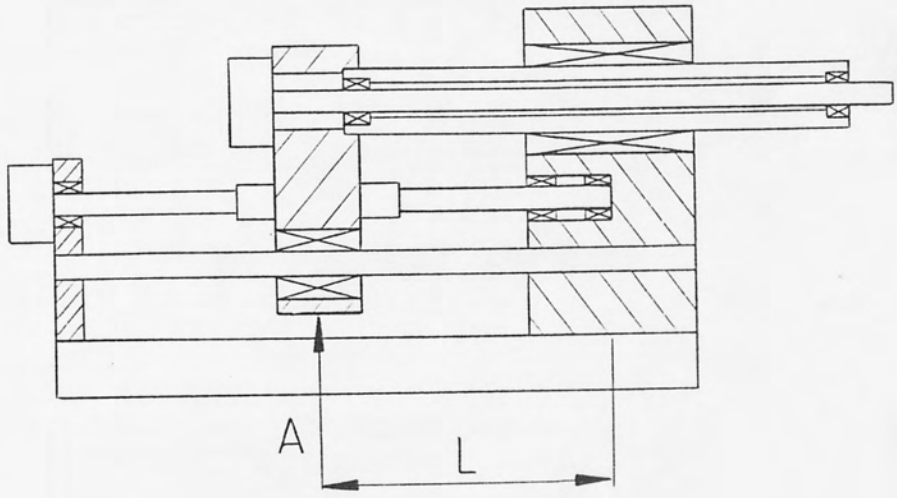
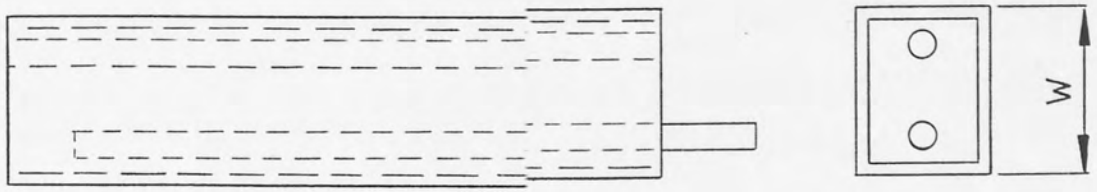
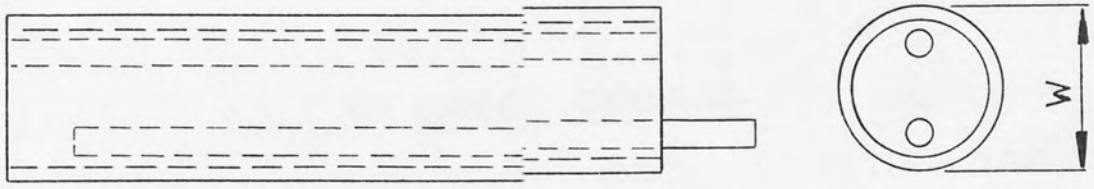


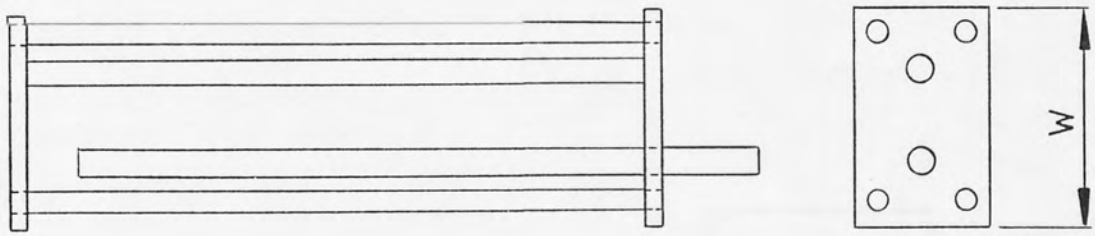
Figure 15 : Bearing Support for the Robot Arm



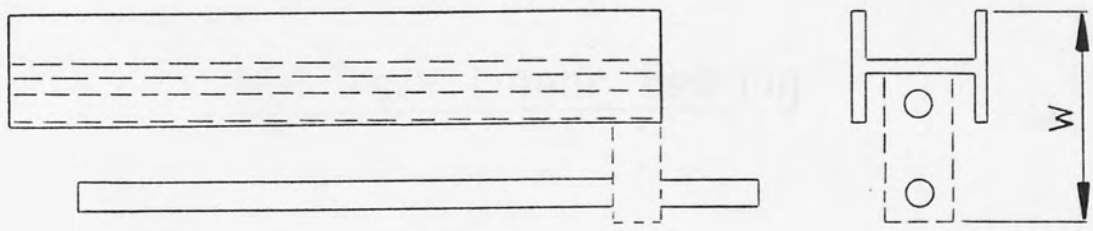
a) box section



b) tubular section

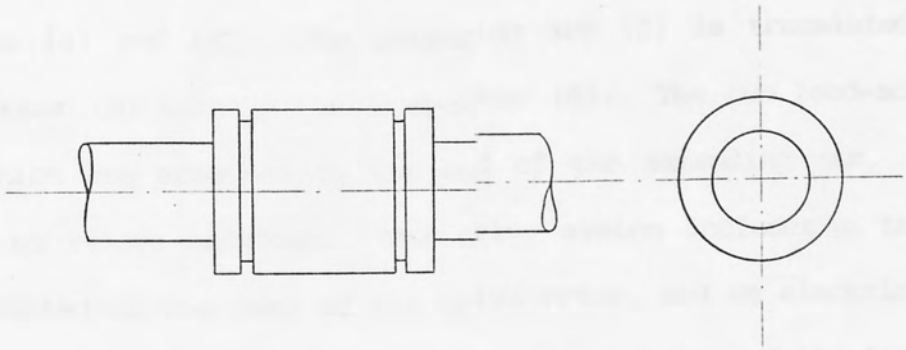


c) tie bars

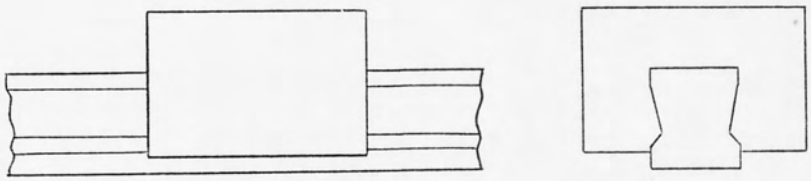


d) channel section

Figure 16 : Design Schemes for the Static Structures of the Robot Arm



Circular Linear Bearing



Flat Track Linear Bearing

Figure 17 : Linear Bearing Options for the Robot Arm

5.2 Design Description

Referring to Figure 18, the static arm (1) attaches to the column arm carrier at point C. The extending arm (2) is supported by two linear bearings (3) and (4). The extending arm (2) is translated by the drive motor (8) turning the lead-screw (6). The two lead-screw nuts (7), which are attached to the end of the extending arm, are pre-loaded to remove backlash. The drive system includes a tachometer (10) mounted to the rear of the drive motor, and an electrical fail-safe brake (9), which is mounted between the motor and the lead- screw to lock the extending arm in position in the power-off condition. The position of the extending arm is measured by the displacement transducer (14).

The drive to the wrist is supplied by the wrist drive motor (11), the wrist drive gearbox (12) and the wrist drive shaft (13).

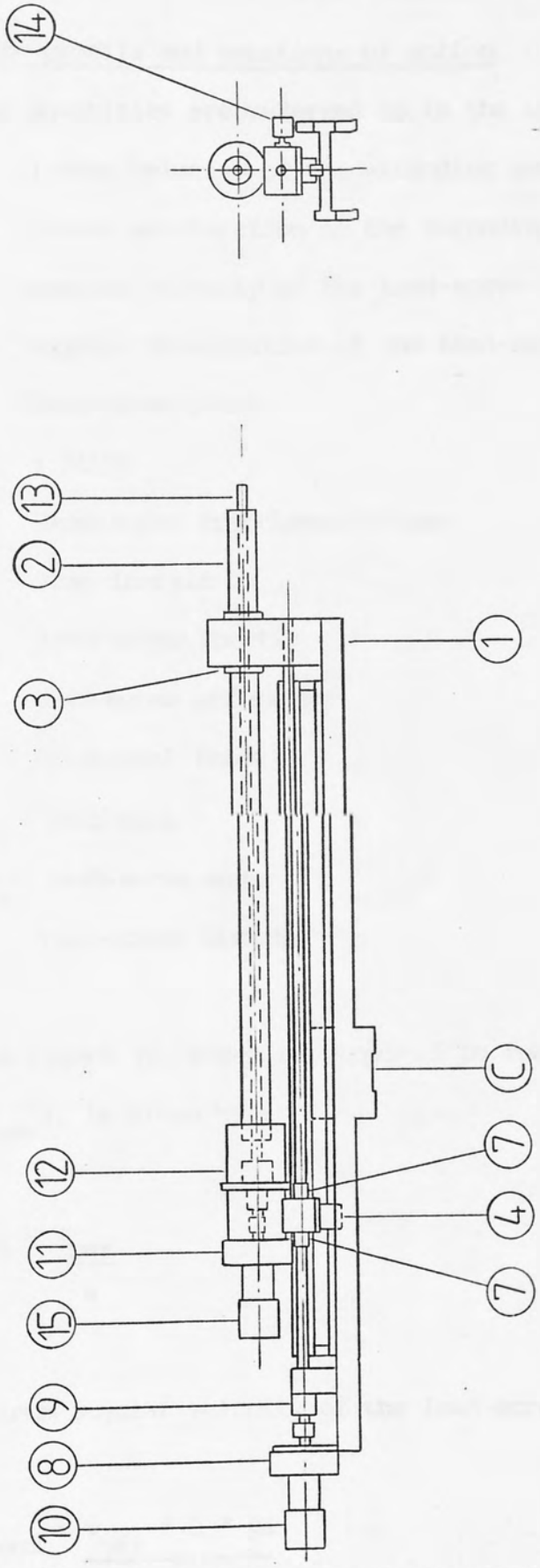


Figure 18 : Robot Arm Arrangement Drawing

5.3 Drive System

5.3.1 Velocity profile and equations of motion

The following quantities are referred to in the analysis:

- v linear velocity of the extending arm
- a linear acceleration of the extending arm
- w angular velocity of the lead-screw
- α angular acceleration of the lead-screw
- p lead-screw pitch
- pi 3.14159
- T_f lead-screw frictional torque
- I_l load inertia
- I_s lead-screw inertia
- e lead-screw efficiency
- F frictional force
- M_l load mass
- M_s lead-screw mass
- d lead-screw diameter

Referring to Figure 19, the time required to reach the maximum linear velocity (v_{max}), is given by :

$$t_1 = \frac{v_{max}}{a} \qquad (1)$$

and the maximum angular velocity of the lead-screw is given by :

$$w_{max} = \frac{v_{max} * 2 * pi}{p} \qquad (2)$$

Combining (1) and (2), the maximum required angular acceleration of the lead-screw is given by :

$$\alpha_{max} = \frac{w_{max}}{t_l} = \frac{v_{max} * 2 * pi}{p * t_l} \qquad (3)$$

The torque required to drive the lead-screw is given by :

$$T = T_f + (I_l + I_s) * \alpha \qquad (4)$$

The frictional torque $T_f = \frac{F * p}{2*e*pi}$ (5)

The load inertia $I_l = 2.53 * 10^{-2} * M_l * p^2$ (6)

and the screw inertia $I_s = M_s * d^2/8$ (7)

The torque required to drive the lead-screw is a maximum when the load carried by the robot is a maximum ie. 15 kg.

The maximum power required of the motor to drive the lead-screw is given by :

$$P_{max} = T_{max} * w_{max} \qquad (8)$$

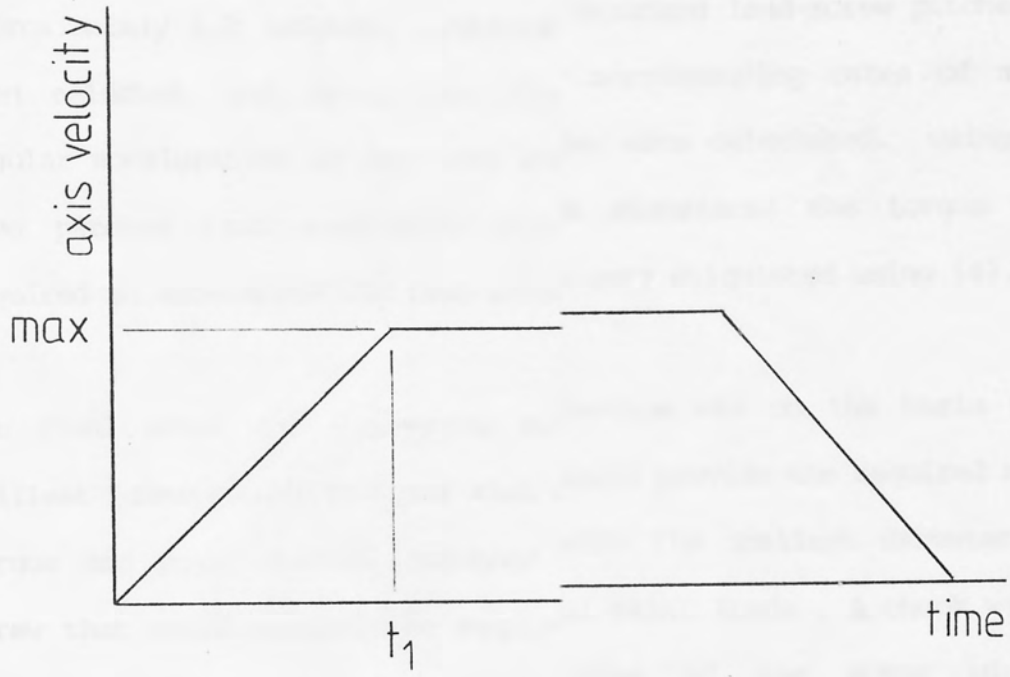


Figure 19 : Theoretical Velocity Profile of Robot Axes

5.3.2 Lead-screw and motor selection

The maximum values of linear velocity and acceleration of the robot arm were selected so that a full stroke of the arm would take approximately 1.5 seconds. Various standard lead-screw pitches were then selected, and using (3), the corresponding rates of maximum angular acceleration of the lead screw were calculated. Using these same pitches (and associated screw diameters) the torque values required to accelerate the lead-screw were calculated using (4).

The final motor and lead-screw selection was on the basis of the smallest (dimensionally) motor that would provide the required maximum torque and power values, together with the smallest diameter lead-screw that would support the required axial loads. A check was made on the theoretical temperature rise of the motor windings, particularly during load acceleration, in order to prevent overheating of the enclosed motor.

5.4 Control System Requirements

Two methods of measuring the linear displacement of the extending arm were considered :

i) using a rotary encoder mounted on the lead-screw drive motor, together with a precision ground lead-screw; or

ii) using a linear displacement transducer to measure the extension of the arm directly.

The second option was selected since the combined cost of the linear transducer and less-accurate, rolled lead-screw was significantly less than the cost of the very accurate, ground lead-screw alone.

A tachometer is mounted on the rear of the motor to provide motor speed feed-back.

An electrically hard-wired mechanical limit switch is provided at each end of the arm stroke, together with a pair of software controlled, photo-cell type limit switches at each end of the arm stroke. The first photo-cell warns that the arm is near to its limit, but the control system will still allow it to move towards the end of the stroke. The second photo-cell signals the control system to halt the arm and to only allow return movement. The hard wired limit switches will only be activated if the arm overruns.

A photocell mounted approximately in the centre of the arm stroke provides a zero marker for the displacement transducer.

6.0 ROBOT COLUMN

6.1 Design Schemes

Having finalised the design of the robot arm, consideration was given to the design of the structure that would support the arm and translate it in both the vertical plane (linear movement) and the horizontal plane (rotary movement). This structure was considered in two parts: the robot column would translate the arm structure in the vertical plane, the robot rotary unit would rotate the arm in the horizontal plane. One of the design decisions that had to be made was whether the column would translate the rotary unit as well as the arm or whether the rotary unit would rotate the column as well as the arm. Figure 20 (a) and (b) illustrates these two choices.

The two major constraints placed upon the design of the column/rotary unit structure were the height of the arm centre line at minimum column stroke and the length of the column stroke. Target values for these two limits were set at 0.5m and 1m respectively (Figure 11 illustrates this point).

Both the design options shown in Figure 20 (a) and (b) have the robot arm mounted on top of the robot column. The height of the arm centre line at minimum column stroke is thus directly dependent on the length of the column stroke and the height of the rotary unit. Options (c) and (d) in Figure 20 are attempts to reduce the height of the arm at minimum column stroke.

In order to achieve the stated minimum arm height of 0.5m, all four designs (Figure 20 (a) - (d)) would require the column stroke to be

less than 0.5m. This did not meet the stated requirement for a column stroke of 1m so all four designs were rejected.

Since with a floor mounted robot, the robot arm must be mounted above the rotary unit (in order to give 360 degree rotation), the major constraint on minimum arm height is the robot column stroke.

It was thus clear that in order to meet the two design constraints, the arm could not be mounted on top of the column and consequently, in order to obtain 360 degree rotation, that the column as well as the arm would have to be rotated by the rotary unit. Figure 20 (e) and (f) show two such design schemes.

The robot arm support structure (and its detail design) was thus divided into two parts : the robot column and the robot base (the rotary unit). The remainder of this chapter concerns the design of the column; the design of the robot base is dealt with in Chapter 7.

Two options were considered for the static structure of the robot column: a tie-bar arrangement and a channel section. As with the design of the robot arm (Section 5.1) the tie-bar option was rejected on the grounds of lack of stiffness and of shaft straightness inaccuracy. The two design schemes based on channel sections are shown in Figure 21.

Both design schemes utilise four linear recirculating ball bushings running on circular section shafts (four shafts in option (a), two in option (b)) since bearing life calculations based on the design weight of the arm and robot payload showed that the cost of four

larger ball bushings was less than eight smaller ball bushings (as could be accommodated in option (a)). Again from the point of view of cost, option (a) requires two more linear shafts than option (b). In addition, since the minimum channel thickness that was practical from a manufacturing point of view, was also found to be capable of supporting the full arm mass and payload, duplication of this channel (as in option (a)) was found to be unnecessary. Option (b) was thus selected as the basis of the column design.

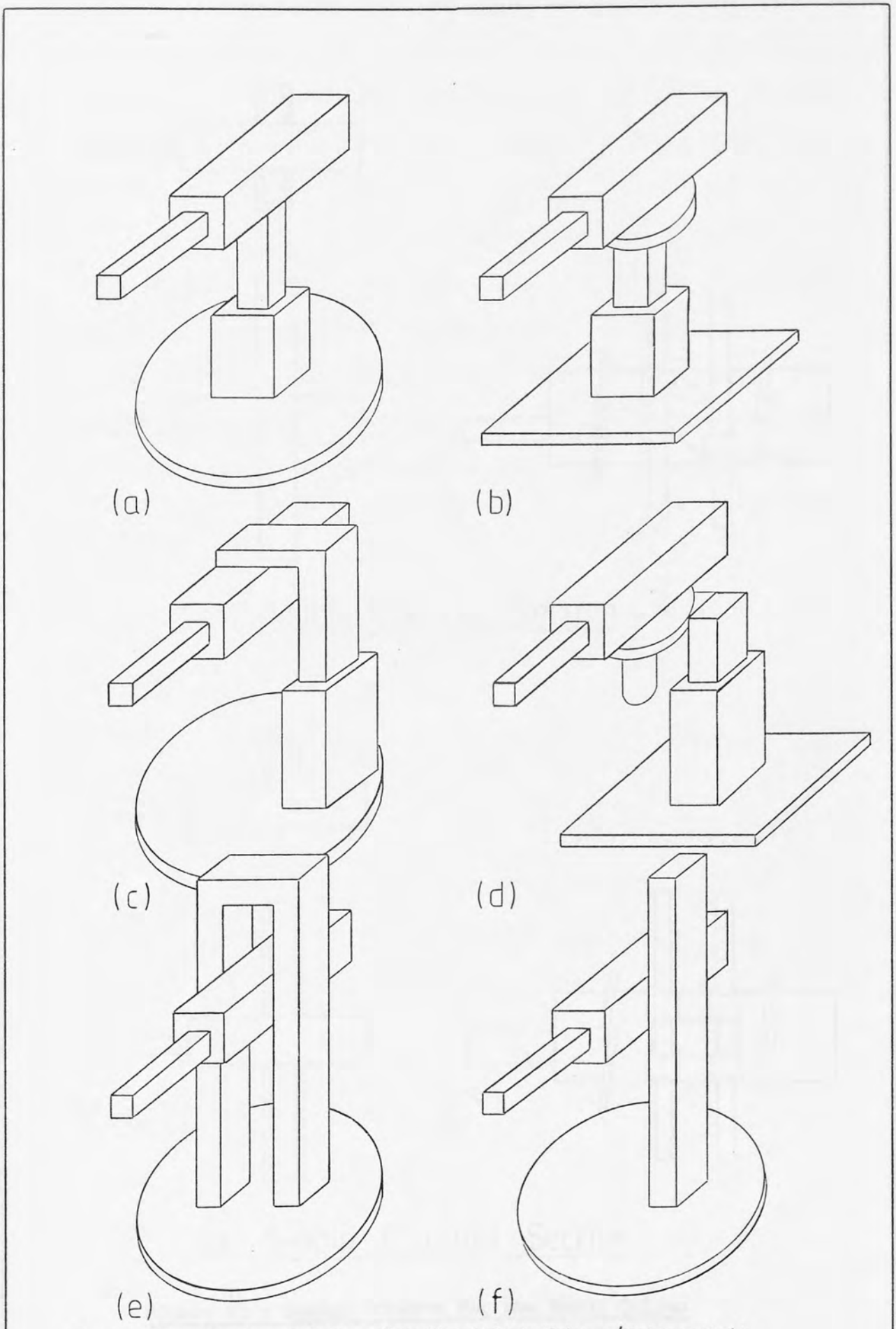
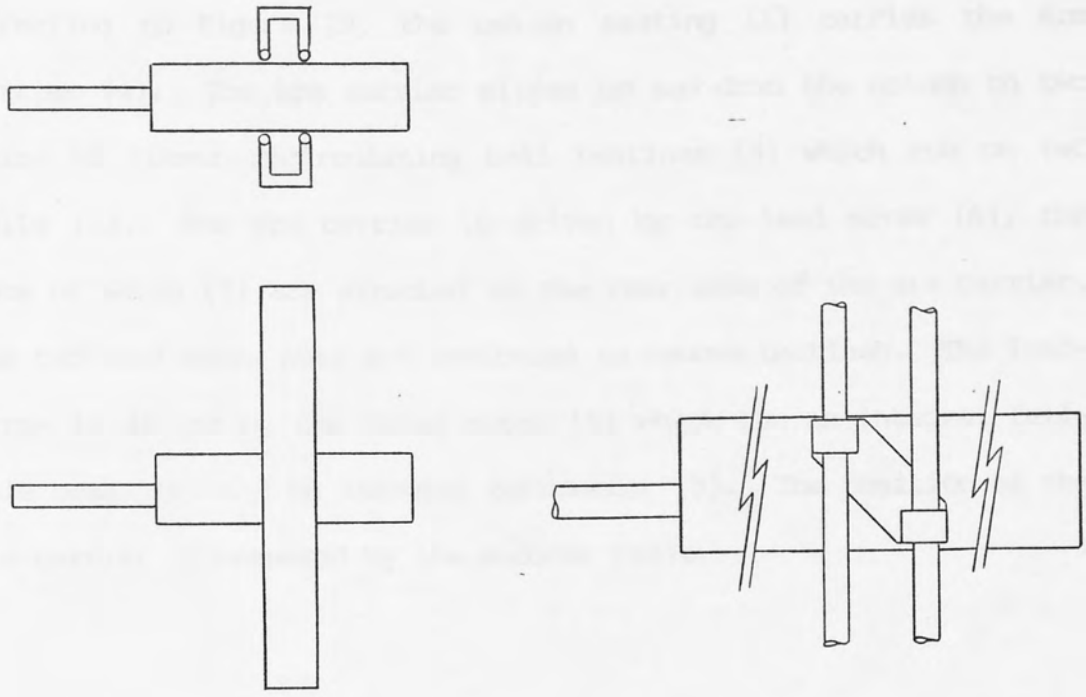
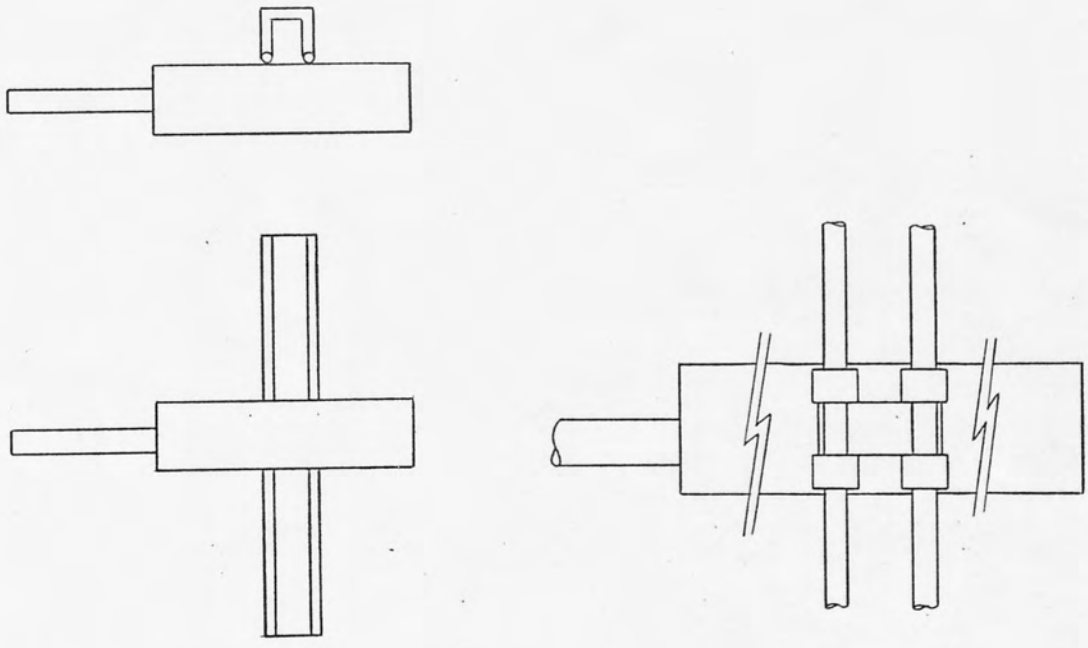


Figure 20 : Design Schemes for the Robot Column/Rotary Unit



a) Double Channel Section



b) Single Channel Section

Figure 21 : Design Schemes for the Robot Column

6.2 Design Description

Referring to Figure 22, the column casting (1) carries the arm carrier (4). The arm carrier slides up and down the column on two pairs of linear recirculating ball bearings (3) which run on two rails (2). The arm carrier is driven by the lead screw (6), the nuts of which (7) are attached to the rear side of the arm carrier. The two lead screw nuts are preloaded to remove backlash. The lead-screw is driven by the drive motor (5) which has an integral fail-safe brake (8) and an integral tachometer (9). The position of the arm carrier is measured by the encoder (10).

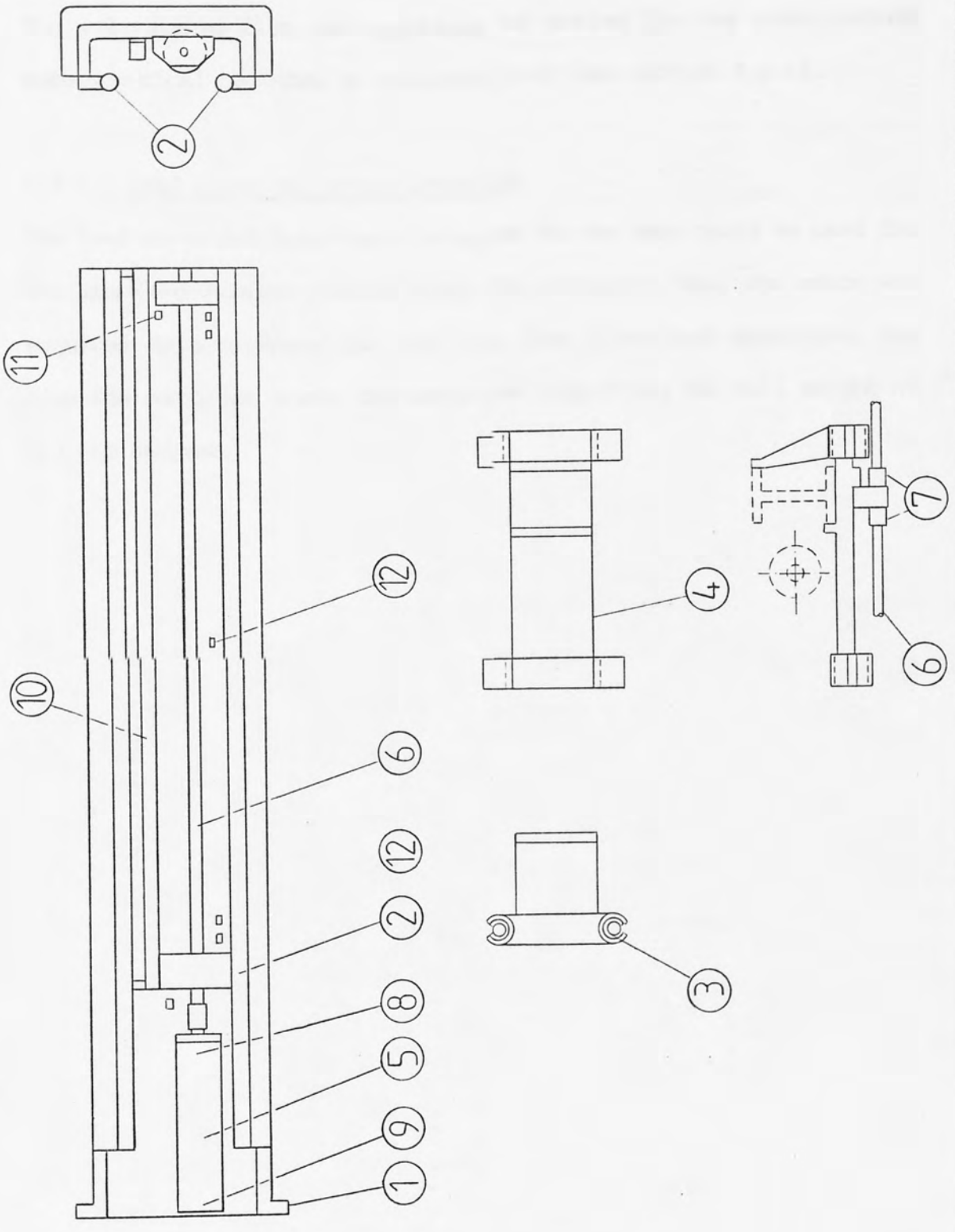


Figure 22 : Robot Column Arrangement Drawing

6.3 Drive System

6.3.1 Velocity profile and equations of motion

The velocity profile and equations of motion for the robot column were identical to those of the robot arm (see Section 5.3.1).

6.3.2 Lead screw and motor selection

The lead screw and motor were selected on the same basis as used for the arm (see Section 5.3.2), with the exception that the motor was required to accelerate the arm, not from a no-load condition, but from the condition where the motor was supporting the full weight of the arm at rest.

6.4 Control System Requirements

As with the robot arm (Section 5.4) a linear displacement transducer was selected as the means of measuring the displacement of the arm carrier along the column. Hard-wired mechanical limit switches and software controlled photo-cells were provided as for the arm (Section 5.4).

7.0 Robot Base

7.1 Design Schemes

The main design decision to be made with regard to the robot base was what type of power transmission system should be used. Three options were considered (Figure 23). The two design constraints were physical size and the maximum torque required to rotate the base: the maximum diameter of the base was set to be 0.5m and the maximum height to be 0.5m. This latter value was determined by the required height of the arm centre line at minimum column stroke (Section 6.1). The torque requirement analysis (Section 7.3.1) suggested that a speed reduction of the order of 200:1 would be needed.

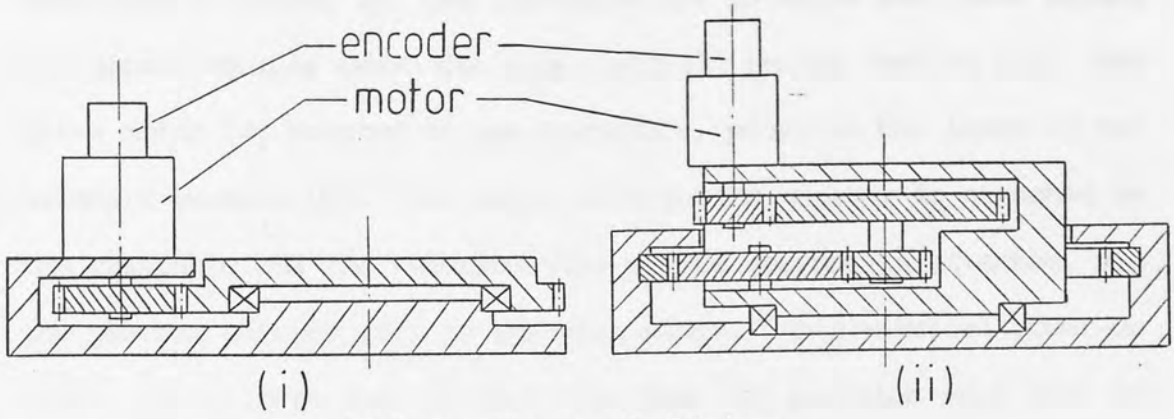
Let us consider the three power transmission options. A flexible belt drive system has the advantage of being backlash-free, however the required reduction could not be achieved within the dimensional constraints of the robot base. The ring-gear and pinion option (i) would require a large drive motor since again the necessary reduction could not be achieved within the dimensional constraints. Only the ring-gear and pinion option (ii) and the harmonic gear box are capable of providing the necessary reduction within the dimensional constraints. Both these options have inherent gear backlash.

Of these two options, the harmonic gearbox has a lower backlash and allows a far simpler design and was thus selected as the basis for the base drive system.

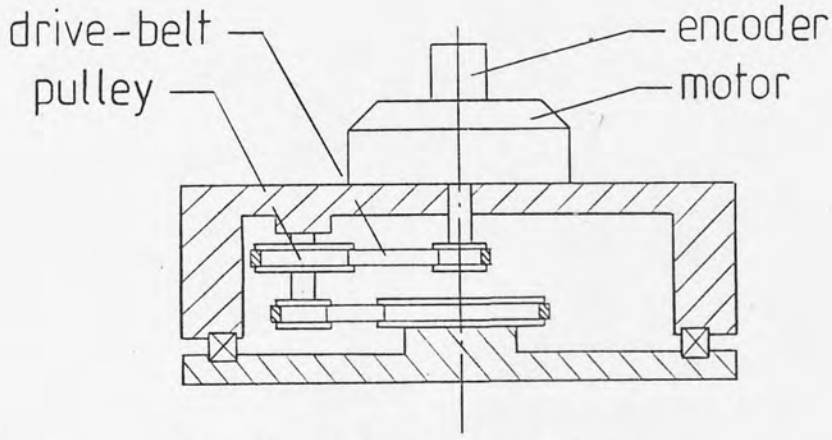
Since no practical mechanical solution to the problem of gear

backlash could be found, the decision was taken that the robot control system would be designed to compensate for the backlash in the base. In addition an electrical brake was to be provided to hold the base in position since this could not be done using the motor.

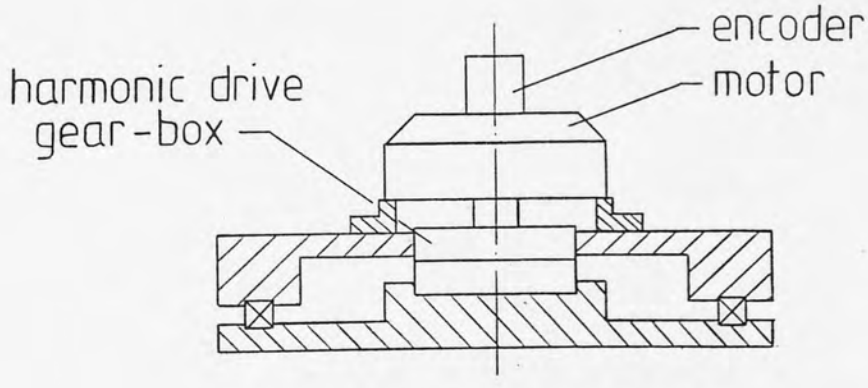
It was decided to design the base around a single, four-point-contact, thin section, ball-race bearing rather than a more traditional arrangement of thrust and ball or roller bearings, on the basis of physical size, design simplicity and cost. In order to increase the stiffness of the base and thus the robot accuracy (as advocated by George [34]), this bearing was preloaded.



a) Ring - Gear & Pinion



b) Flexible Belt Drive



c) Harmonic Drive Gearbox

Figure 23 : Robot Base Power Transmission Options

7.2 Design Description

Referring to Figure 24, the turntable (1) to which the robot column is bolted, rotates about the base plate (2) on the bearing (3). The drive motor (4) mounted on the turntable, provides the input to the harmonic gearbox (5). The output side of the gearbox is attached to the turntable and the reaction side of the gearbox is attached, via the gearbox carrier (6), to the base plate. An electrical power-on brake (7) is provided to lock the base in position when not in motion. An incremental encoder (8) is mounted to the back of the drive motor.

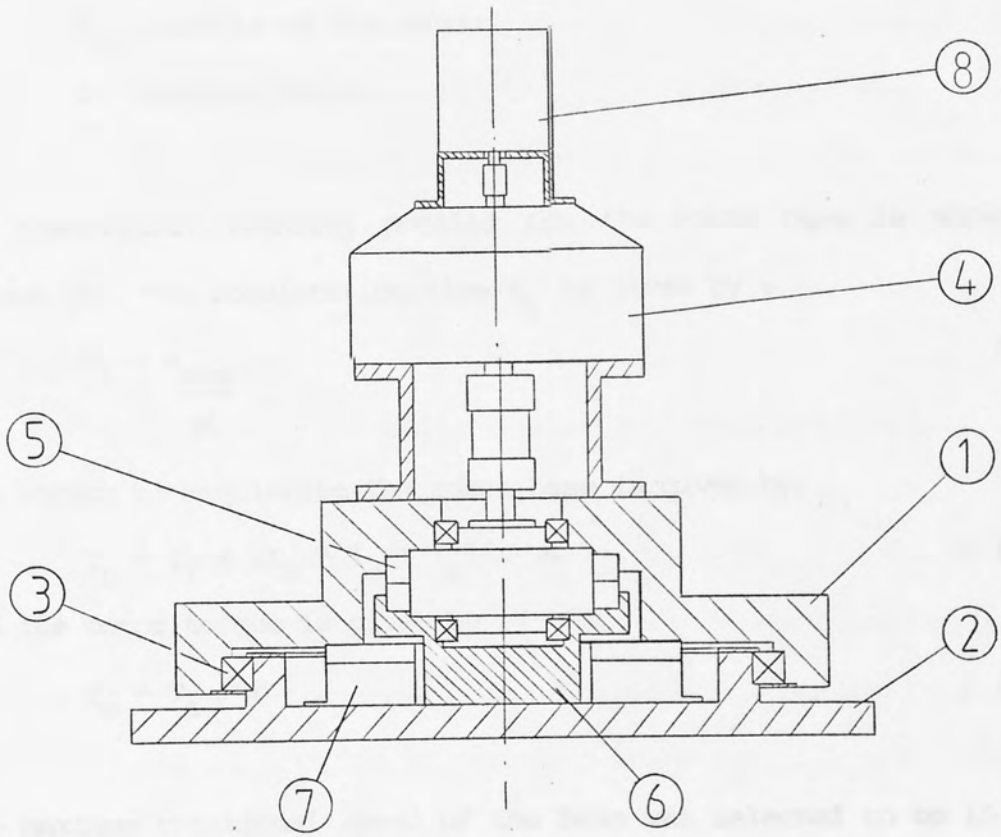


Figure 24 : Robot Base Arrangement Drawing

7.3 Drive System

7.3.1 Velocity profile and equations of motion

The following quantities are referred to in the analysis:

- w angular velocity of the base
- α angular acceleration of the base
- T_o gearbox output torque
- T_m motor output torque
- T_f frictional torque of the base main bearing
- I_b inertia of the base
- I_c inertia of the column
- r gearbox ratio

The theoretical velocity profile for the robot base is shown in Figure 19. The acceleration time t_1 is given by :

$$t_1 = \frac{w_{max}}{\alpha} \tag{9}$$

The torque to accelerate the robot base is given by:

$$T_o = T_f + (I_b + I_c + I_a) \times \alpha \tag{10}$$

and the motor torque is given by :

$$T_m = T_o / r \tag{11}$$

The maximum rotational speed of the base was selected to be 15 rpm, so that the maximum motor speed (w_m) is given by:

$$w_m = r \times pi/2 \tag{12}$$

7.3.2 Selection of motor and gearbox

Since there is only one manufacturer of harmonic gearboxes, the selection of the gearbox was the controlling item. In (10), the only variable is the angular acceleration of the robot base, thus a gearbox was selected, that has a torque output capability that was sufficient to provide a high enough rate of acceleration so as to keep t_1 in (9) below 0.5 s. A motor was then selected with appropriate maximum torque and speed values using (10) and (11). A printed circuit type motor was used so as to keep the height of the base unit to a minimum, thus allowing the arm to be lowered as far as possible. A check was made on the theoretical temperature rise of the motor windings during base acceleration in order to prevent the motor overheating.

7.3.3 Drive backlash

The harmonic drive gearbox is not a backlash-free device. The quoted value of maximum backlash is 3 minutes of arc. This represents approximately, at the tip of the robot arm, +/- 9 mm of free play. Since this backlash is inherent in the base design, it was decided to use the robot control system to compensate for the backlash, and to provide a power-on brake to lock the base in position when stationary.

7.4 Control System Requirements

In order to measure the required positional repeatability of +/- 0.1mm at an arm radius of 2m, a 17 bit encoder would be required. This was the basis of the design shown in Figure 24 but this idea was later rejected on the basis of cost. Instead a 10 bit encoder was mounted as shown, and it was intended that software counters would be used to accommodate the fact that the encoder would rotate fully several times for one revolution of the robot base. A tachometer was also to be mounted on the back of the motor for the purpose of speed feed-back.

Two mechanical limit switches were mounted on the rotating base turntable (Figure 24) (activated by strikers mounted on the base plate) that provide signals for the two extremes of the base rotation.

8.0 ROBOT WRIST

8.1 Design Schemes

As stated in Section 5.1, the decision was taken that both the wrist pitch and the wrist roll axes (Figure 25 (i)) would be driven by a single motor. The main requirement of this area of the robot design was thus the translation of one axis of movement into two independent axes of movement.

Figure 25 (iii) shows the initial concept of the wrist. The design was such that either the pitch axis or the roll axis could be selected but not both together. With the pitch brake locked, rotation of the drive shaft results in wrist roll; with the roll brake locked, rotation of the drive shaft results in wrist pitch.

This wrist design had three problem areas:

i) the physical size of the wrist did not allow for the provision of a position measuring device for the roll axis (both encoders and potentiometers were considered);

ii) the pitch axis could not be locked using a single brake since a brake having the required pitch torque capacity could not be found that would fit within the wrist design envelope; and

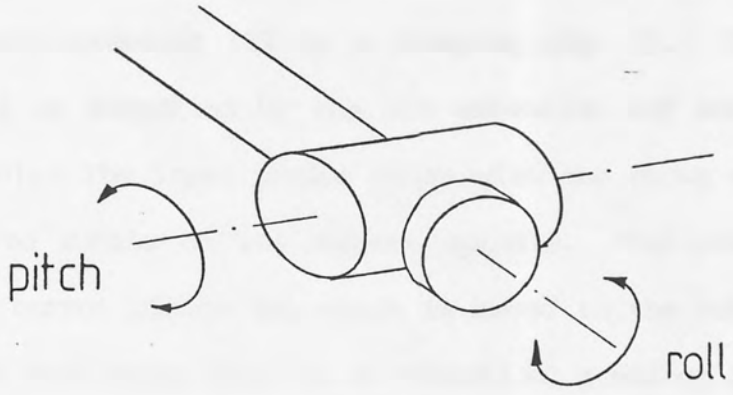
iii) there was inherent gear backlash in the design.

Item (ii) was resolved by the use of a pair of brakes one mounted either side of the wrist. This however, added to item (i) since there was no longer room within the design for a pitch position measuring device. It was realised that the problem of wrist position measurement could be resolved by driving a single

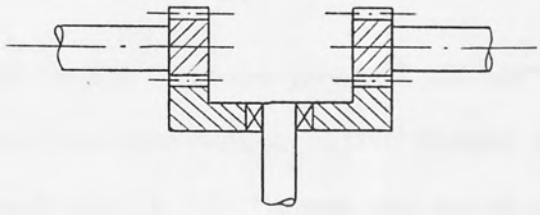
incremental encoder from the wrist drive shaft and mounted on the back of the drive motor, provided that the gear backlash could be reduced to an acceptable level.

In an attempt to resolve item (iii), an alternative to the use of three bevel gears was considered (Figure 25 (ii)), namely the use of a pair of spur gear pinions and a crown wheel. This design would allow the use of spring-loaded, anti-backlash pinion gears thus eliminating the backlash problem. Unfortunately a supplier of such pinions that would accommodate the calculated wrist driving torques could not be found. It was thus decided (as with the robot base - Section 7.1) that the problem of gear backlash would have to be accommodated by the robot control system.

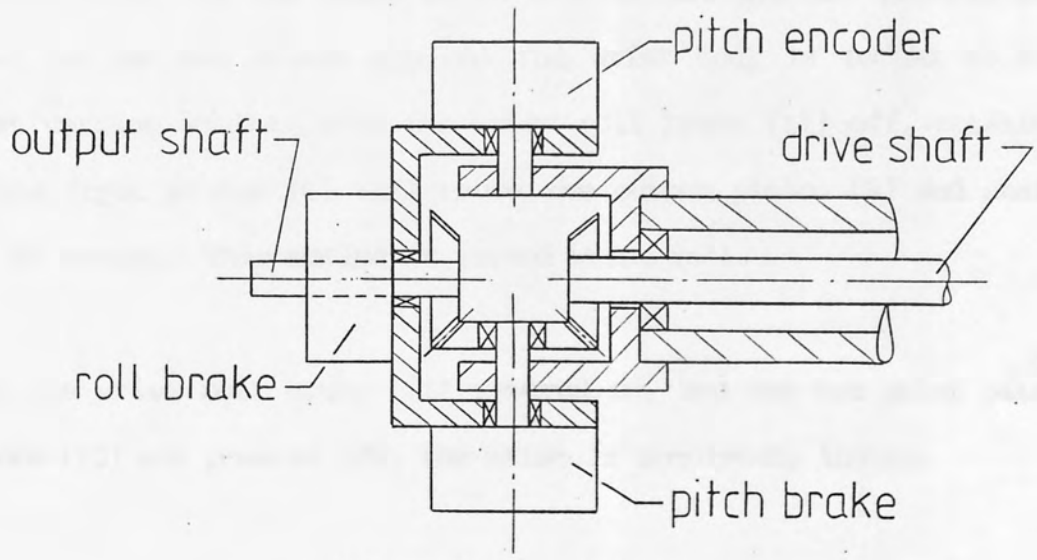
In order to minimise both the physical size and weight of the wrist a number of options were considered for the wrist braking elements: electro-magnetic brakes, electro-magnetic clutches and air operated and vacuum operated brakes. Pancake type air cylinders were found to be incapable of generating the required braking torques, as was the case with vacuum units. The wrist was thus designed around electro-magnetic brake units.



i) Wrist axes



ii) Crown wheel & 2 spur gears



iii) Wrist concept

Figure 25 : Robot Wrist Design Schemes

8.2 Design Description

Referring to Figure 26, the wrist carrier (4) is attached to the end of the arm extension (3) by a clamping rig (2). The wrist drive shaft (1) is supported by the arm extension and carries the input pinion (6). The input pinion mates with the crown wheel (7) which is free to rotate on its support spindle. The crown wheel mates with the output pinion (8) which is keyed to the output shaft (9). The wrist roll brake (11) is an electrical power-on brake that acts to lock the output shaft (9). The two wrist pitch brakes (10) are electrical fail-safe brakes which act to lock the wrist body (5) to the wrist carrier (4).

When the wrist roll brake (11) is powered on the output pinion (8) is locked and with the two wrist pitch brakes (10) powered off, rotation of the input pinion (6) causes the wrist body (5) to rotate relative to the wrist carrier (4). This motion is termed wrist pitch. When the two wrist pitch brakes (10) are not powered off (that is the two brakes are on) the wrist body is locked to the wrist carrier so that with the wrist roll brake (11) off, rotation of the input pinion (6) will cause the output pinion (8) and shaft (9) to rotate. This motion is termed wrist roll.

With the wrist roll brake (11) powered on, and the two wrist pitch brakes (10) not powered off, the wrist is completely locked.

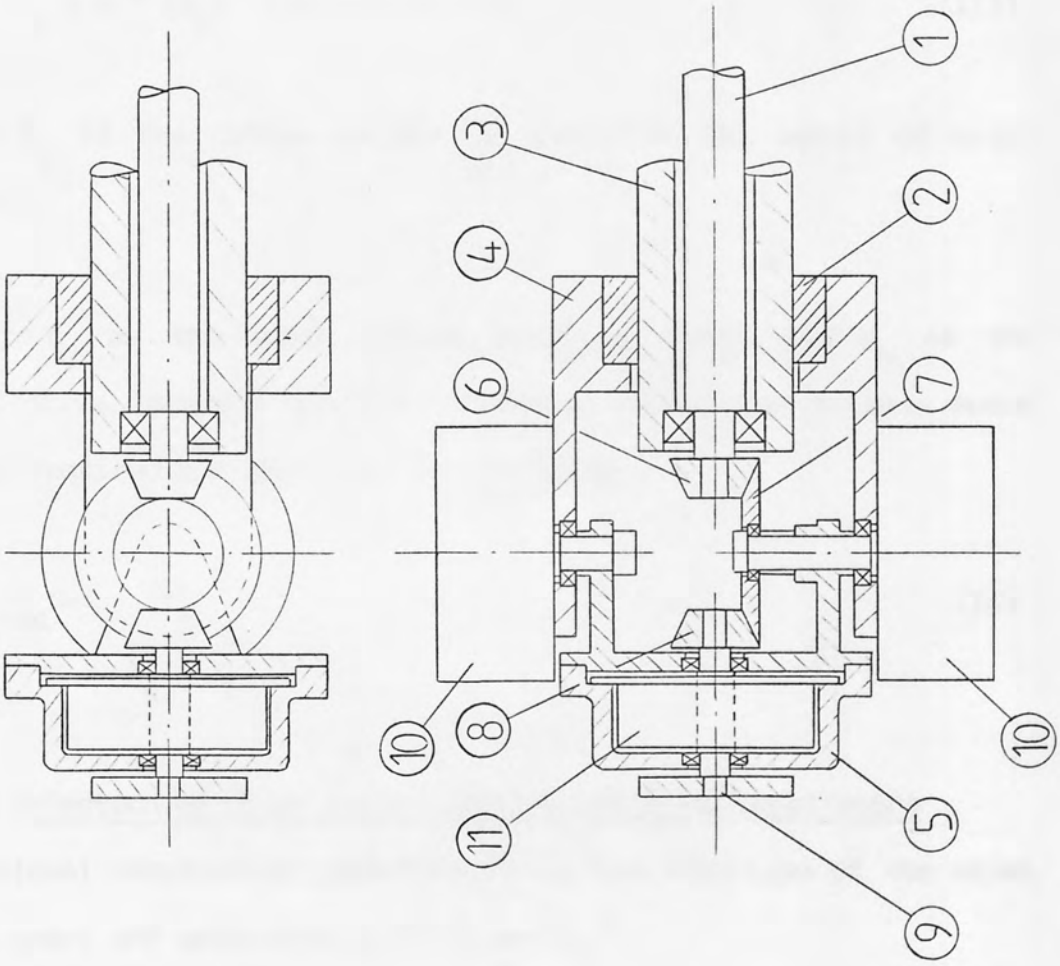


Figure 26 : Robot Wrist Arrangement Drawing

8.3 Drive System

8.3.1. Velocity profile and equations of motion

The theoretical velocity profiles for both the wrist pitch and the wrist roll motions are as for the other robot axes (Figure 19).

Referring to Figure 27, the maximum torque required for the wrist pitch is given by :

$$T_p = M * (R_p)^2 * \alpha + M * g * R_p \qquad (13)$$

where R_p is the radius of the payload from the centre of wrist pitch.

Taking r_g as the wrist pinion reduction ratio and r_h as the wrist drive harmonic gearbox reduction ratio, the maximum motor torque required for wrist pitch is given by :

$$T_m = \frac{T_p}{r_g * r_h} \qquad (14)$$

8.3.2 Selection of drive motor, gearbox and wrist bevel gears

Dimensional constraints predominated in the selection of the wrist bevel gears and associated gear ratio (r_g).

In (13) the maximum value of M was known (15 kg) and from the holding capacity of the two wrist pitch brakes (Figure 26 item 10), the maximum value of R_p was known. Having set the wrist bevel-gear gear ratio, the drive motor and gearbox were selected so as to maximise the angular acceleration whilst maintaining the maximum

rotational speed of the wrist pitch at 15 rpm.

The torque required for wrist roll, even without the advantage of the wrist bevel gear gear ratio was less than that required for wrist pitch.

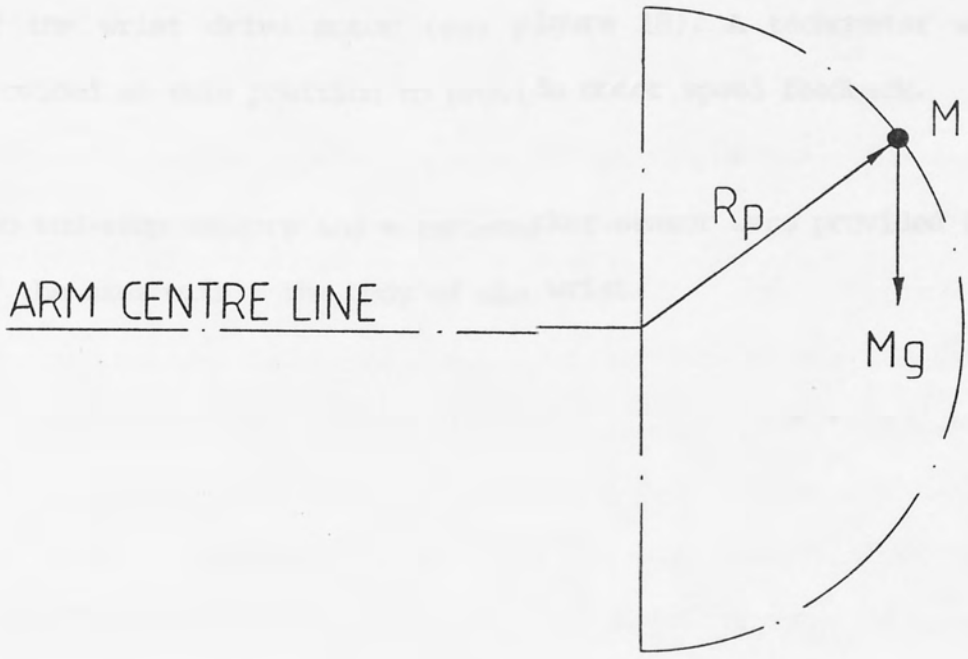


Figure 27 : Robot Wrist Pitch Torque Requirement

8.4 Control System Requirements

As stated in Section 8.1 the required physical dimensions of the wrist prevented the mounting of position measuring devices directly onto either the pitch or the roll axes, and it was decided that a single encoder would be driven from the wrist drive shaft. Again, physical dimensions dictated that the encoder be mounted on the back of the wrist drive motor (see Figure 18). A tachometer was also provided at this position to provide motor speed feedback.

Two end-stop sensors and a zero-marker sensor were provided for both of the axes within the body of the wrist.

9.0 ROBOT CONTROL SYSTEM

9.1 General Description

The design and development of the robot control system did not form part of this project and the following description is given only as background information. A full understanding of and participation in the design of the control system was however necessary to ensure compatibility with the mechanics of the robot.

The robot control system is hierarchial in nature, as may be seen from Figure 28. Each robot axis is controlled by a slave computer which is within the axis drive servo-loop. This somewhat unusual arrangement was developed both in order to overcome the need for what would have been a very expensive, high bit resolution analogue-to-digital converter (due to the long length of the linear axes and the high positional resolution of the robot), and to give flexibility for control software. The slave computer is responsible for the comparison of signals and outputting of speed demand within the servo-loop.

The central computer communicates either with the robot programming unit or the program storage unit, and downloads the position demands to the individual robot axes during program execution.

9.2 Individual Axis Control

Figure 29 illustrates the basic elements of the control system that are common to each of the robot axes.

The position of the axis is read by the micro-processor from the encoder via the encoder interface and hardware counter. The latter element is to free the micro-processor of the chore of constantly monitoring the encoder output. The micro-processor compares the present axis position with the desired axis position and outputs a value to the digital-to-analogue converter that is proportional to this error. The analogue voltage that represents this error is then presented as a demand signal to the servo-drive unit, having first been passed by the brake/end stop card that controls brake switching and checks if the endstops are clear (again removing this chore from the microprocessor). The servo-drive unit then outputs the desired current to the motor with the tachometer providing speed feed-back.

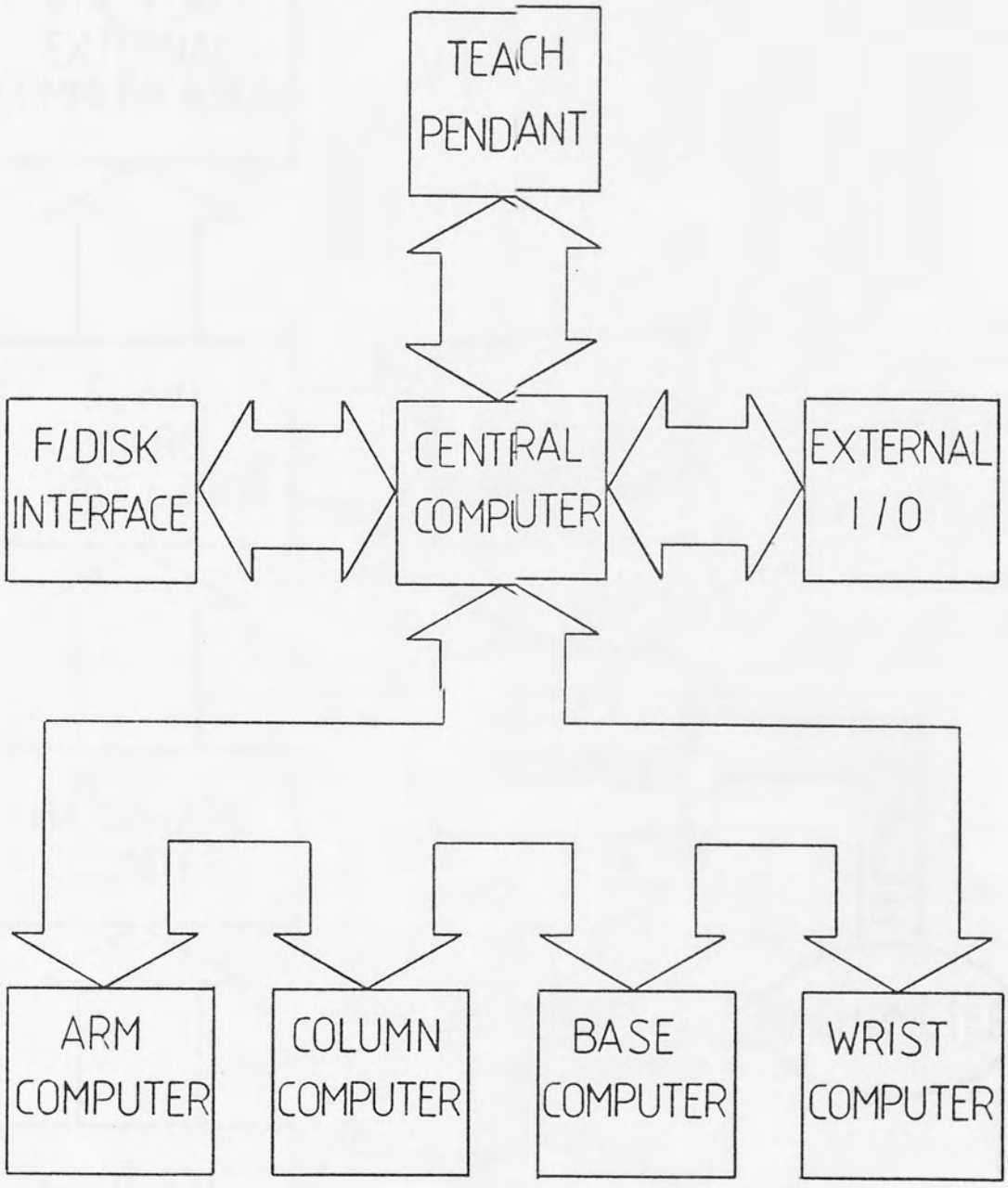


Figure 28 : Control System Schematic Drawing

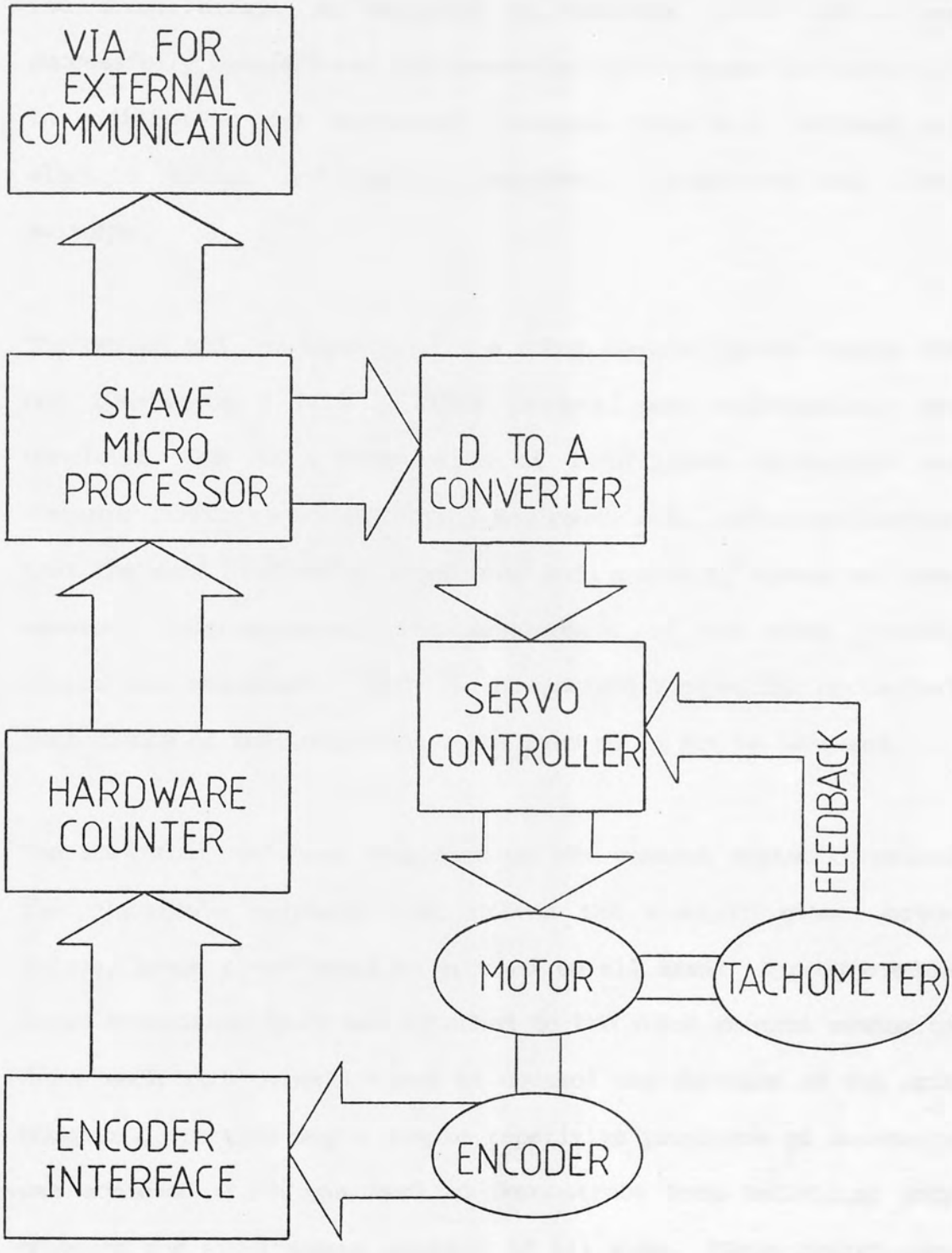


Figure 29 : Single Axis Control Schematic Drawing

10.0 THE PROTOTYPE ROBOT

10.1 Discussion

The robot design as detailed in Chapters 5,6,7 and 8 was successfully manufactured and assembled and is shown in Figure 30. In addition to the mechanical elements, this work included all electric motors and brakes, encoders, transducers and limit switches.

The design and development of the robot control system (which did not constitute a part of this project) was unfortunately not completed. Due to a combination of insufficient electronic and computer resources within Cirrus Equipment Ltd., and a realisation that the work involved in developing such a control system had been severely underestimated, the development of the robot control system was abandoned. Without the control system the mechanical performance of the individual robot axes could not be assessed.

The work that had been completed on the control system comprised the electronic hardware that drives the electric motor servo-drives, hence power could be applied to all axes. A programmable logic controller (PLC) was attached to the robot control system to drive each axis using a timer to control the duration of the axis movement. In this way a simple repetitive programme of movements was achieved which was used to demonstrate both individual axis movement and simultaneous movement of all axes. (This control work was carried out for the robot to be displayed at the Automan '83 exhibition and did not form part of this project). Thus, in a limited way the robot mechanical design was proven, however, it was not possible to demonstrate the accuracy and repeatability of the

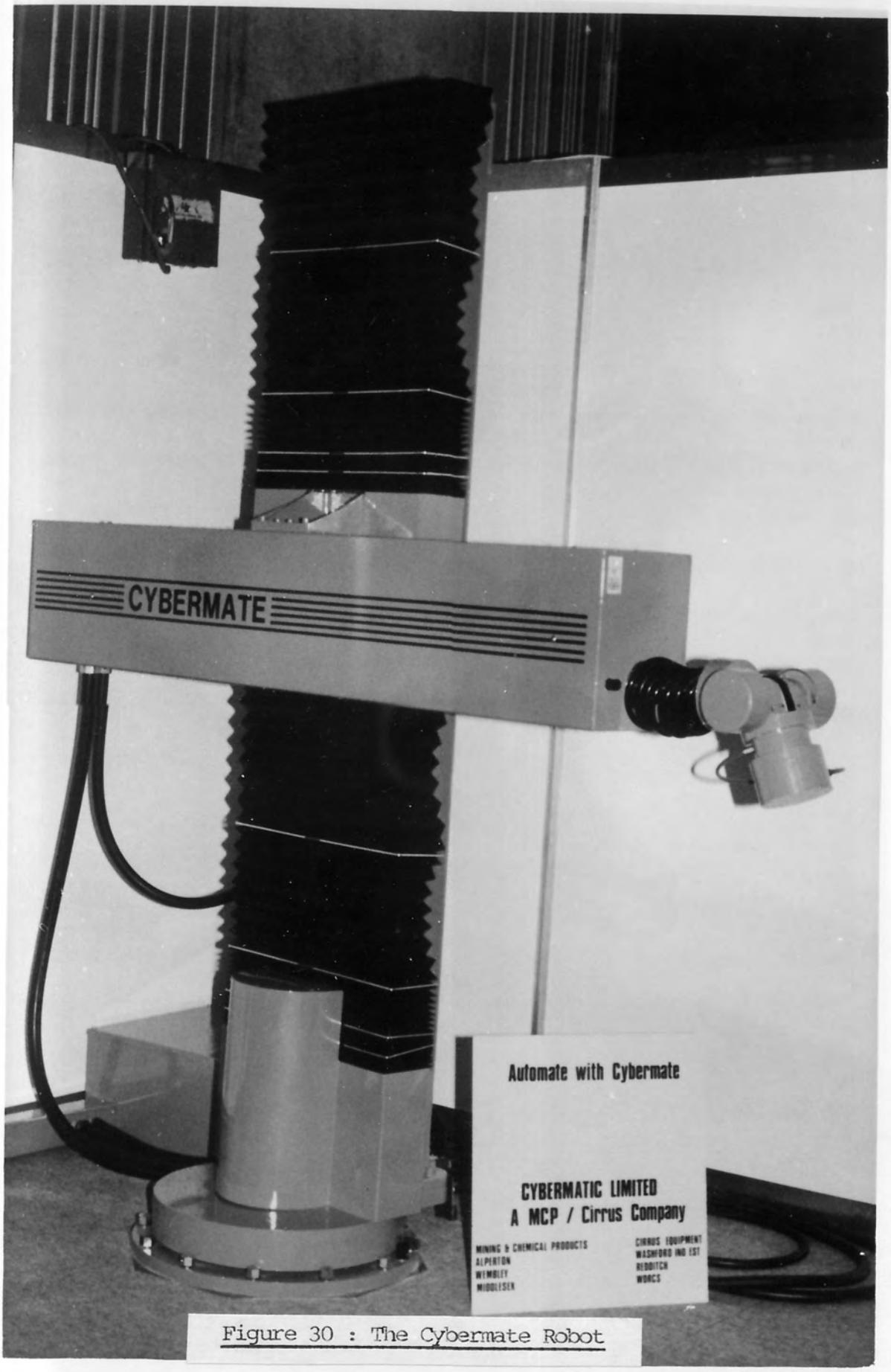


Figure 30 : The Cybermate Robot

robot axes. This limited movement of the robot axes highlighted three areas of interest :

i) backlash in the robot base;

ii) backlash in the robot wrist; and

iii) lack of stiffness in the robot arm and the robot column;

10.1.1 Backlash in the robot base

The harmonic drive gearbox used in the robot base is not a backlash free device. It was the intention that this backlash be overcome by software control. The preliminary work carried out on the software for such compensation showed the need for an encoder to be mounted on both sides of the backlash, that is on the driving and the driven side. The robot had been designed with an encoder only on the gearbox driving side (the motor). A design to provide two encoders was prepared but was not implemented.

10.1.2 Backlash in the robot wrist

As with the robot base, the harmonic gearbox used for the wrist drive had inherent backlash. In addition the bevel gears of the wrist pitch and roll axis also have backlash.

It was not possible to accommodate an encoder on either of the wrist axes on the driven side of the gearbox because of the need to minimise the physical size of the wrist. In order to overcome the backlash problem the following control strategy was proposed :

i) both wrist axes are locked by brakes;

ii) power is applied to the drive motor to take up the gearbox

backlash (the power level would be just below the axes brake holding torque levels);

iii) the brake on the axis required to be driven is released;

iv) the wrist drive encoder is read;

v) the axis is driven to the target position; and

vi) the position of the axis (from the encoder) is stored in the wrist slave computer memory.

This strategy was not tested due to the abandonment of the robot control system development work.

10.1.3 Robot arm and column stiffness

It was apparent that both the robot arm and column deflected under:

i) the addition of a payload to the robot; and

ii) the extension of the robot arm (the structure deflecting under it's own weight).

Such deflections had been anticipated since the robot structure had been designed for minimum mass in order to minimise the drive power requirements.

11.0 THE SINGLE DIE-PIN TEST RIG

11.1 Discussion of Moving Die Pins

As has been explained in Section 3.5, in order to compensate for the inherent positional inaccuracy of the robot when loading a blade into an encapsulation die it was proposed that the datum and clamping pins of the die should be capable of controlled movement. Such moving pins could then be used to finely position (and also gauge) the blade, to an accuracy far greater than that achievable by the robot.

The requirement is thus for a pin that can be accurately positioned over a short stroke (approximately 1mm), and held rigidly in position during the encapsulation process. In addition, as outlined in Section 3.1, either the pin must be suitable for inclusion in a system that uses electrical conduction to sense contact of the pin and the blade, or an alternative form of sensing must be found. This latter possibility was rejected on the grounds that electrical conduction is both simple and ideal for control system purposes. Any moving pin system must therefore be able to utilise the techniques of electrical contact sensing. The existing form of electrical insulation of pins, namely ceramic coating (see Section 3.1) would not be suitable for running the pin in a bearing (because of the abrasive nature of the ceramic) and since the ceramic coating process is not ideal, an alternative was being sought.

11.2 Moving Die Pin Design Schemes

11.2.1 The "Plastic Pin"

Figure 31 shows the principle of this design of pin. The carbide rod (the spherical end of which contacts the blade) is bonded to, but electrically insulated from, the steel pin outer sleeve by a thin layer of thermo-plastic.

A number of test pins were made using a plastic called Victrex [35] which was supplied in 0.005" thick sheets. Carbide and metal outer sleeve samples were prepared that had a clearance of approximately 0.012" on the sleeve internal diameter. The Victrex sheet was wrapped around each pin and the pins carefully fitted into the outer sleeves. Each pin and sleeve was supported in a specially made jig that clamped the carbide and the outer sleeve in such a way as to ensure that the carbide was held centrally in the outer sleeve bore. The pin and jig were then fired in an oven at 800 degrees Centigrade for approximately 5 minutes after which time the plastic had bonded to both the carbide and the outer sleeve. Thus the carbide was secured to and insulated from the outer sleeve. The metal outer sleeve could then be machined and ground as required for the pin to run in standard metal bearings (thus removing the problems associated with ceramic coated pins).

The test pieces that were prepared proved the principle that the carbide could be bonded to and insulated from the outer sleeve. A number of datum pins for a commercial encapsulation die were prepared using this method and highlighted an unforeseen problem. With this design of pin, the insulating material (the plastic) is exposed within the die cavity (point A in figure 31), and since it

is so thin in section (0.005"), it can be easily bridged by metal swarf or fragments. Only by increasing the thickness of the plastic layer to about 3-4 mm could the possibility of swarf shorting the pin be removed. This was impractical from the point of view of how to apply the plastic to the pin and sleeve (plastic injection moulding was considered) and because of the increased overall diameter of the pin (which is normally limited to approximately 8 mm because of the need to arrange pin centres close to one another). This design of pin was thus rejected as being unsuitable for use in an industrial environment.

11.2.2 "Plastic Bushes"

Figure 32 shows an alternative scheme for supporting a moving pin whilst electrically insulating it from the die block. This method uses commercially available plain bushes with a bore lining of acetyl resin [36] to support a standard die-pin in the die block. This method of pin support was used in the first pin test-rig that was built.

11.2.3 The drive system

Since air had been ruled out on the basis of compressibility causing positioning problems, the choice was between electric or hydraulic drives. Since linear movement is required, the use of an electric motor would require rotary to linear movement conversion, thus introducing complexity, particularly since the drive system must be back-lash free. Some form of linear electric drive would not have this problem.

A major constraint placed upon the drive system is that since it is

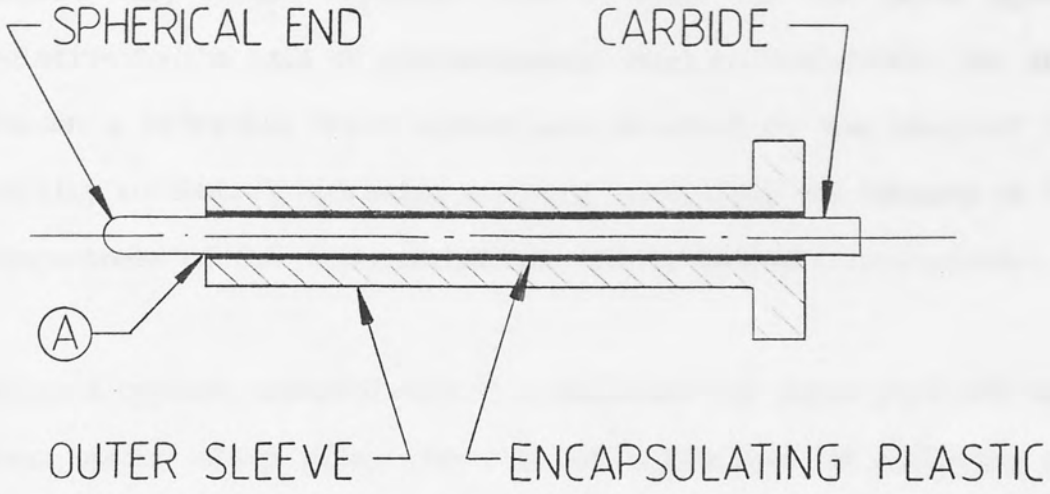


Figure 31 : A "Plastic Pin"

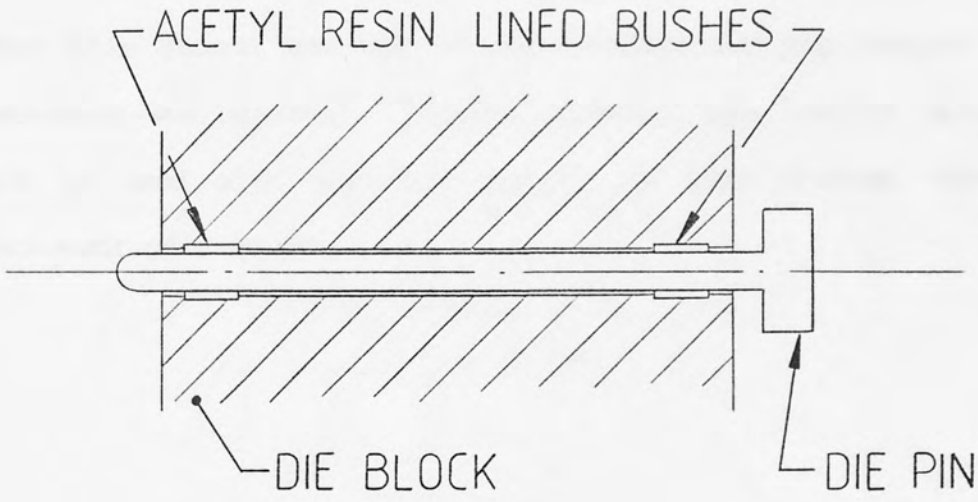


Figure 32 : "Plastic Bushes"

often required in a commercial encapsulation die that pins be placed very close together, the diameter of the drive system relative to the axis of pin movement, must be minimised. For this reason a hydraulic drive system was selected on the basis of the ability to control hydraulic movement accurately and because of the compactness of the drive mechanism, namely hydraulic cylinders.

Since a typical encapsulation die contains six datum pins and upto four spring clamp pins, the cost of the method of achieving pin movement must be carefully considered. As stated in Section 3.5, since the mechanism for blade positioning is to be included in every die and the loading robot may serve upto four dies, it is necessary for the positioning mechanism to be produced economically in order that the encapsulation system cost is not greatly increased.

For this reason the use of servo-valves for the control of pin movement was rejected. Instead, standard flow control valves were to be used with software control to give precise, controlled movement of each pin.

11.3 Computer Hardware Description

The computer development system that was used for both the writing of the control software and driving of the test-rigs is shown schematically in Figure 33. The computer system is based on the Motorola 6809 8-bit microprocessor and supports the FLEX disk operating system.

All software was written using the programming language Pascal.

The Dual VIA board was included to provide real time interrupts as a basis for real time timers.

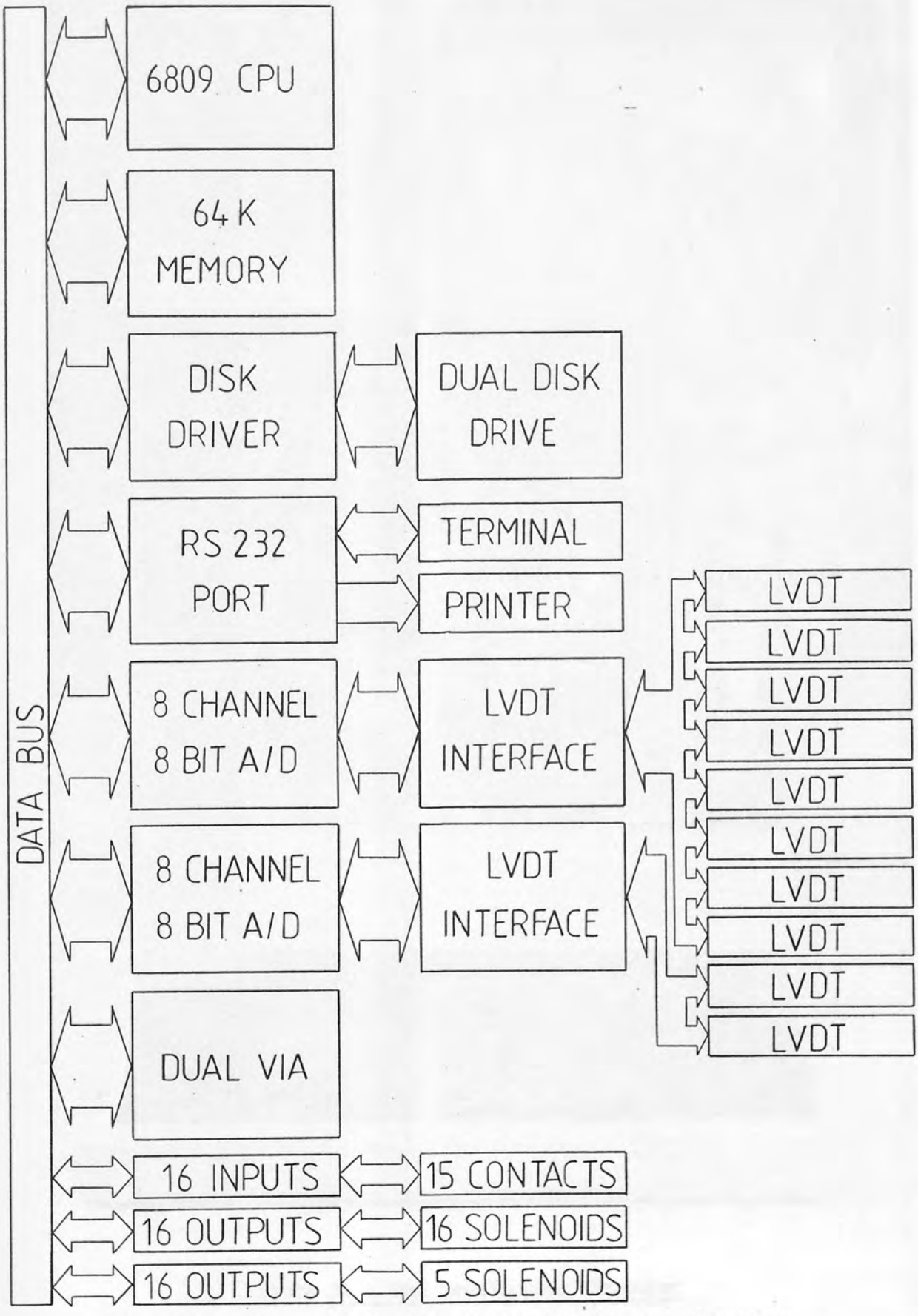


Figure 33 : Schematic Drawing of the Development Computer

Figure 34 : The Development Computer

11.4 Test Rig 1

The rig (Figure 35) comprised a sliding die-pin mounted with plastic bushes in a metal block together with an oil-filled, miniature air cylinder that was connected to the head of the die pin, and a linear variable differential transformer (LVDT) that measured the position of the pin head. The operation of the LVDT is described in [37] and [38]. Oil was used in preference to air to minimise positional errors due to fluid compressibility. The cylinder was controlled by a directional flow control solenoid valve that switched the flow of oil from a hydraulic pump (operating at approximately 100 psi).

The rig was used for the initial development of the software necessary to drive and control the pin and to examine the problems of controlling the position of such a pin to tight tolerances. Figure 36 shows the logic that was used to control the pin. During each test the pin position was recorded by the computer at a sampling rate of 10 ms and the results plotted out on a dot matrix printer at the end of the test.

The LVDT that was used had a working stroke of 1mm and was read using an 8-bit analogue-to-digital converter (A/D). The pin position could thus be read to an accuracy of 0.00015". This accuracy could be improved if required by setting the full range of the A/D over a shorter working stroke of the LVDT, however 1mm was considered to be necessary for the degree of positional error envisaged.

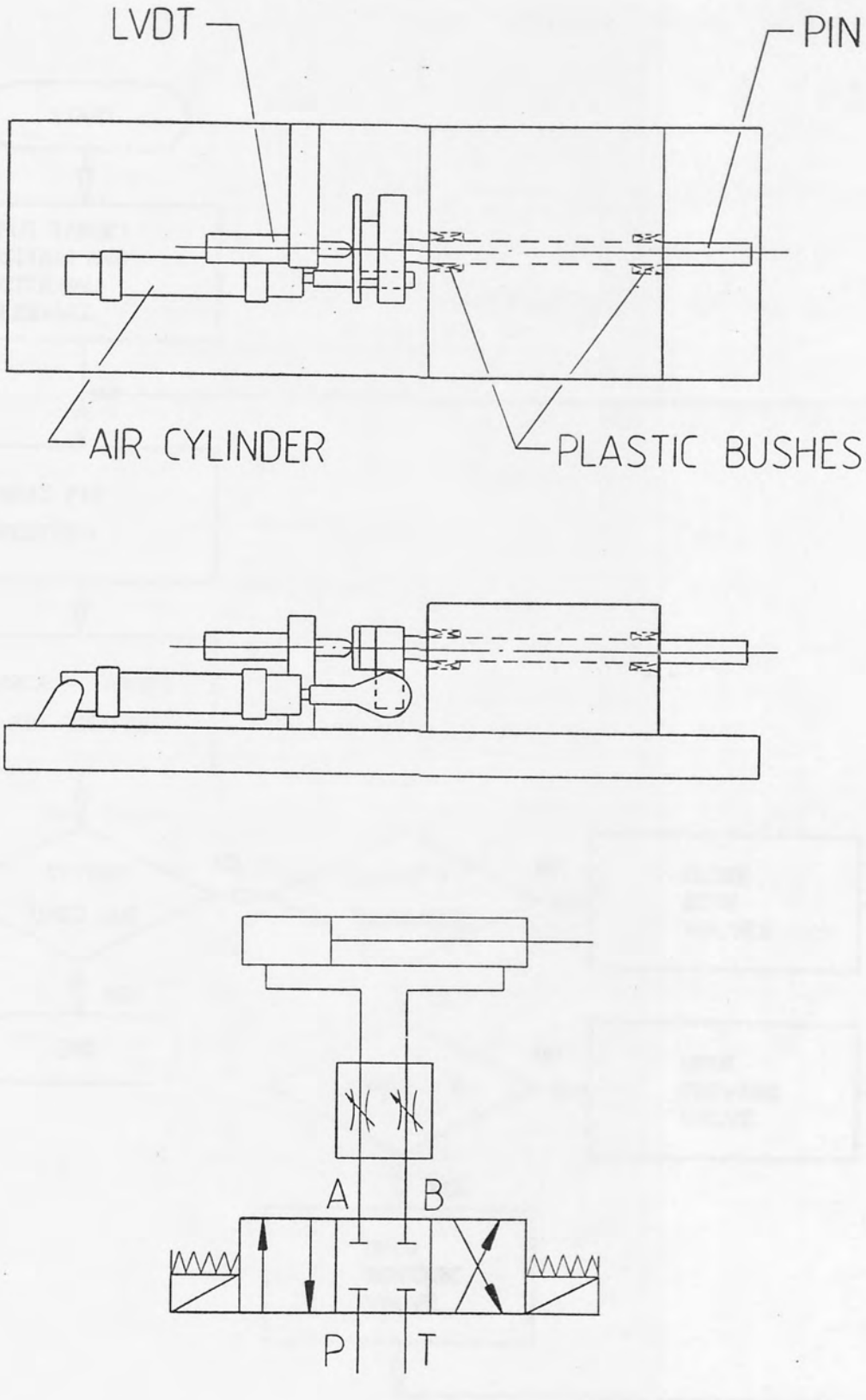


Figure 35 : The First Single Die Pin Test Rig

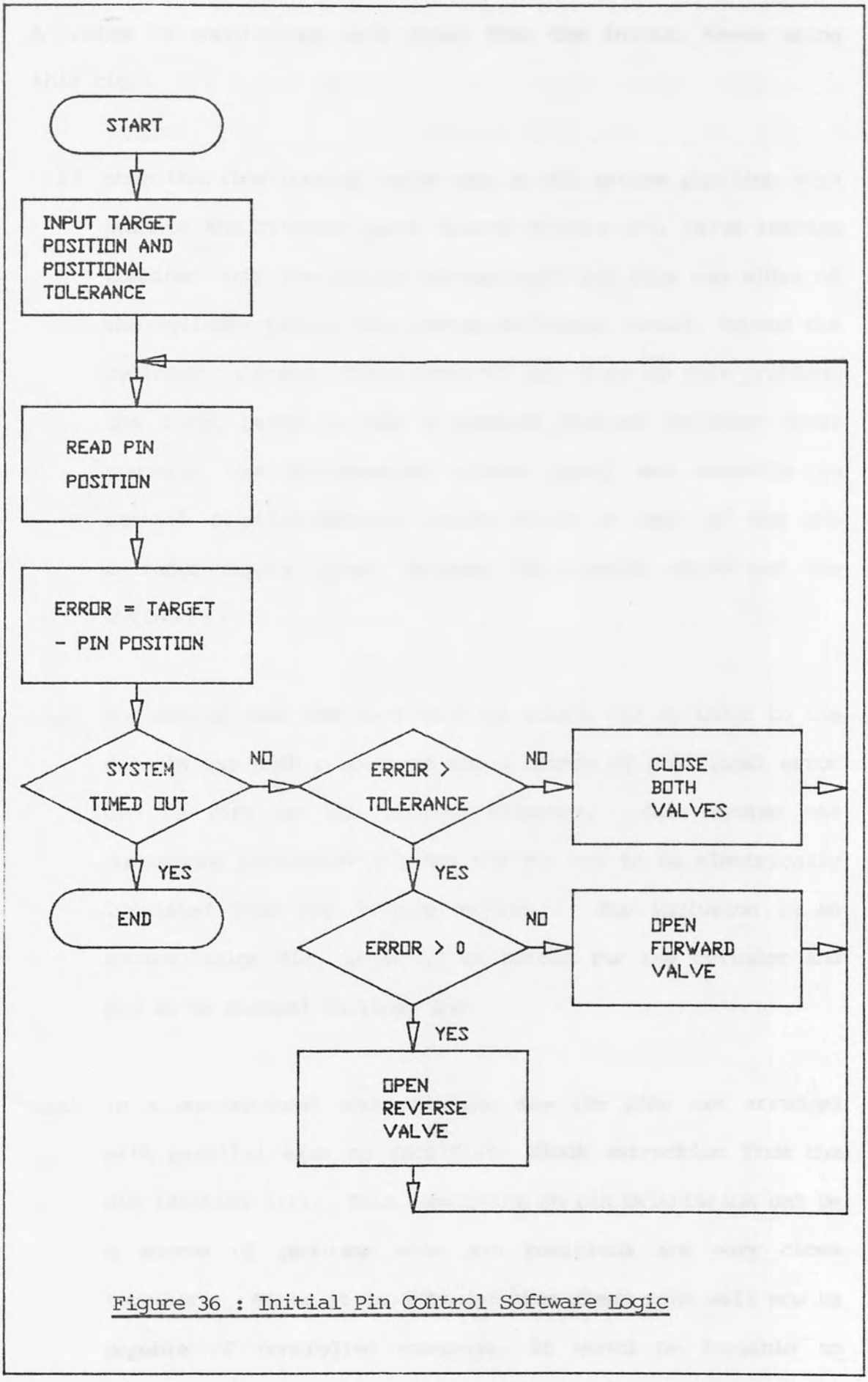


Figure 36 : Initial Pin Control Software Logic

A number of conclusions were drawn from the initial tests using this rig :

i) when the flow control valve was in its centre position with both of the cylinder ports closed (Figure 35), valve leakage combined with the single rod-end cylinder (the two sides of the cylinder piston thus having different areas), caused the cylinder to creep. There were two solutions to this problem, the first being to use a doubled rod-end cylinder (thus removing the differential piston area) and secondly to install a pilot-operated check valve in each of the two cylinder supply lines, between the control valve and the cylinder;

ii) the method that had been used to attach the cylinder to the die-pin was both cumbersome and a source of positional error due to play in the linking elements. This system was cumbersome particularly since the pin had to be electrically insulated from the driving cylinder. For inclusion in an encapsulation die, it would be better for the cylinder and pin to be mounted in-line; and

iii) in a conventional encapsulation die the pins are arranged with parallel axes to facilitate block extraction from the die (Section 3.1). This constraint on pin orientation can be a source of problems when pin positions are very close together. Since it is intended that these pins will now be capable of controlled movement, it would be feasible to arrange the pins on non-parallel axes and withdraw them all

from the die cavity prior to encapsulation block removal. Pin extraction from the cast block would require an extraction force of approximately 800 N per pin [39] equating to a hydraulic pressure of approximately 900 psi (based on the cylinder size under consideration). Thus, the driving cylinder would need to be a true hydraulic rather than an oil-filled air cylinder.

From these three conclusions, it was decided to build a second test-rig.

11.5 Test-Rig 2

11.5.1 Mechanical description

Following the conclusions drawn from the first test-rig, it was decided to mount the driving cylinder in-line with the die-pin. Since the necessity of attaching the die-pin to, and electrically insulating it from, the cylinder piston rod would require a cumbersome connection, it was decided that the cylinder piston rod should itself form the die-pin. This required that either the piston rod be electrically insulated from the body of the cylinder, or the whole of the cylinder be electrically insulated from the die block. The former could be achieved by replacing the bronze neck-gland bushes of the cylinder with the "plastic bushes" as used in the first test rig. However, practical constraints ruled out this option since the "plastic bushes" are only available in metric sizes in the size range required, and a suitable metric miniature hydraulic cylinder could not be found. Thus it was decided to electrically insulate the whole of the cylinder from the die block. A schematic drawing of the test-rig is shown in Figure 37.

The hydraulic cylinder (3) is accurately located in the die block (1) by the ceramic bush (11). The cylinder is fixed to the die block by the clamp (9) and the ceramic insulating bushes (10). The cylinder rod (4) which forms the die-pin is prevented from rotating about its own axis by the linear bearing (8) and guide bar (7). The pin position is measured over the positioning range by the LVDT (6). Shorting of the pin within the die cavity is prevented by the ceramic bush (12). The die front plate (2) completes the die cavity facilitating encapsulation block removal and carries the spring-loaded pin assembly. The position of the spring-loaded pin (13) is

measured by the second LVDT (14).

Ceramic bushes were used for insulation purposes since this material can be accurately machined and is dimensionally stable over the working temperature range of an encapsulation die.

A double rod-end cylinder was used primarily in order to mount the LVDT for measuring the pin position. In addition this removed any problems that could arise due to piston differential area.

To overcome the problem of hydraulic valve leakage two pilot-operated check valves were added to the hydraulic circuit. A double rod-end hydraulic cylinder was used (allowing a maximum system hydraulic pressure of 2000 psi).

Initial testing of the rig showed that when the pin was not being driven, the LVDT was registering a constant low amplitude oscillation of the pin. It was suspected that this was due to the hydraulic pump thus an alternative pressure source was sought. Initially a one-to-one air/oil pressure system was installed, however, it was found that the air supply available could not supply sufficient pressure. Tests showed that a pressure of at least 65 psi was needed to overcome the pin friction. The air supply could only supply air at a pressure that fluctuated between 70 and 100 psi. Thus the air/oil pressure intensifier was substituted (this being an oscillation free device). This however did not cure the apparent pin oscillation which was subsequently traced to a fault on the LVDT driver card.

The air supply to the pressure intensifier is controlled by a solenoid valve so that the system can be pressurised under computer control. Removal of the voltage from the valve causes the air pressure to be vented to atmosphere as a fail-safe measure, and causes re-charging of the intensifier with oil (from the reservoir).

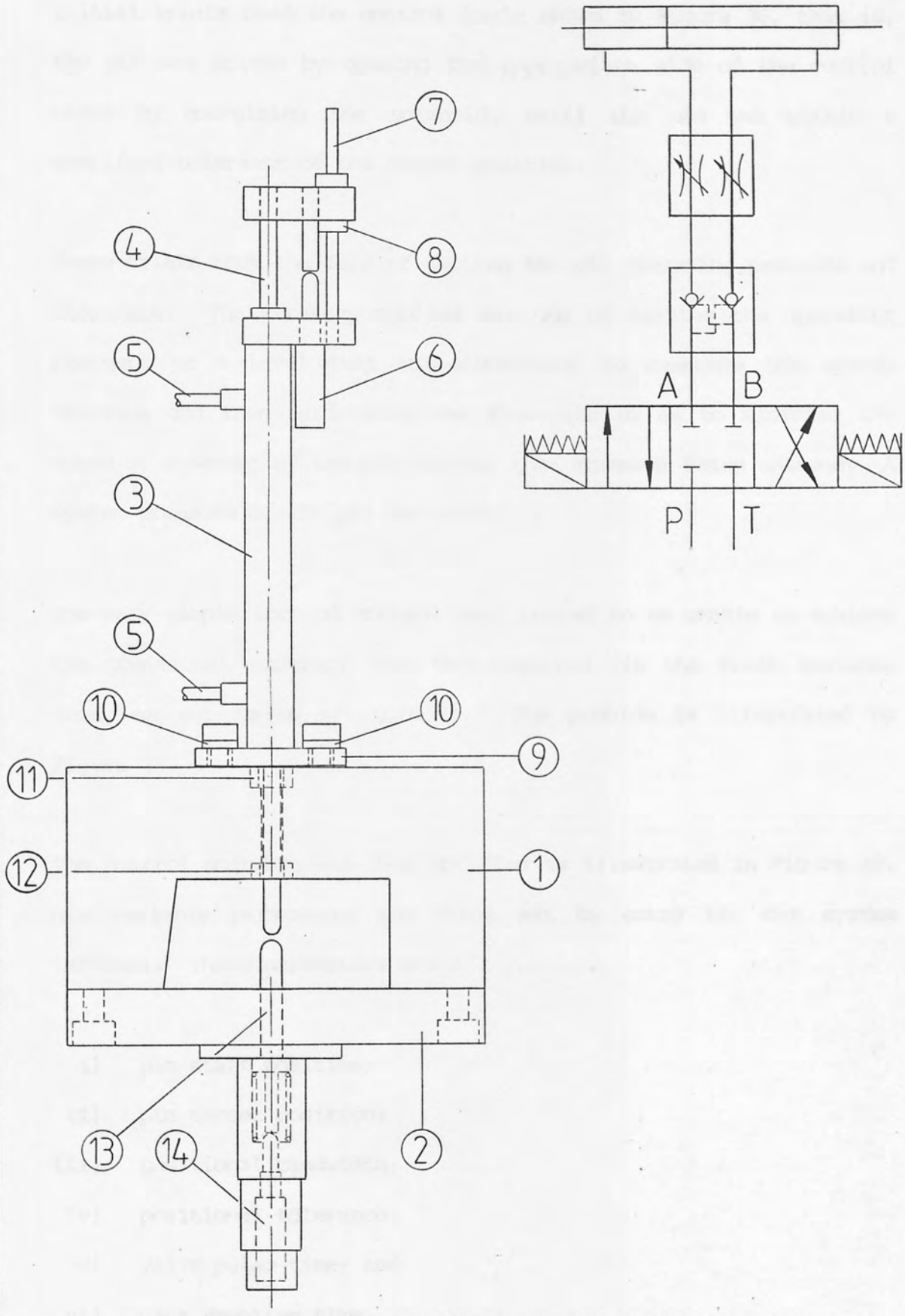


Figure 37 : The Second Single Die Pin Test Rig

11.5.2 Control software

Initial trials used the control logic shown in Figure 36, that is, the pin was driven by opening the appropriate side of the control valve by energising the solenoid, until the pin was within a specified tolerance of the target position.

These trials took the form of setting the pin operating pressure and flow-rate. The strategy applied was one of setting the operating pressure to a level that was sufficient to overcome the system friction and then minimising the flow-rate so as to minimise the speed of movement of the pin without that movement being unsteady. A system pressure of 450 psi was used.

The very simple form of control used proved to be unable to achieve the positional accuracy that was required (in the first instance this was set to be +/- 0.001"). The problem is illustrated by Figure 38.

The control software was thus modified as illustrated in Figure 39. Six variable parameters are first set by entry via the system terminal. These parameters are :

i) pin start position;
ii) pin target position;
iii) positional bandwidth;
iv) positional tolerance;
v) valve pulse-time; and
vi) data sampling time.

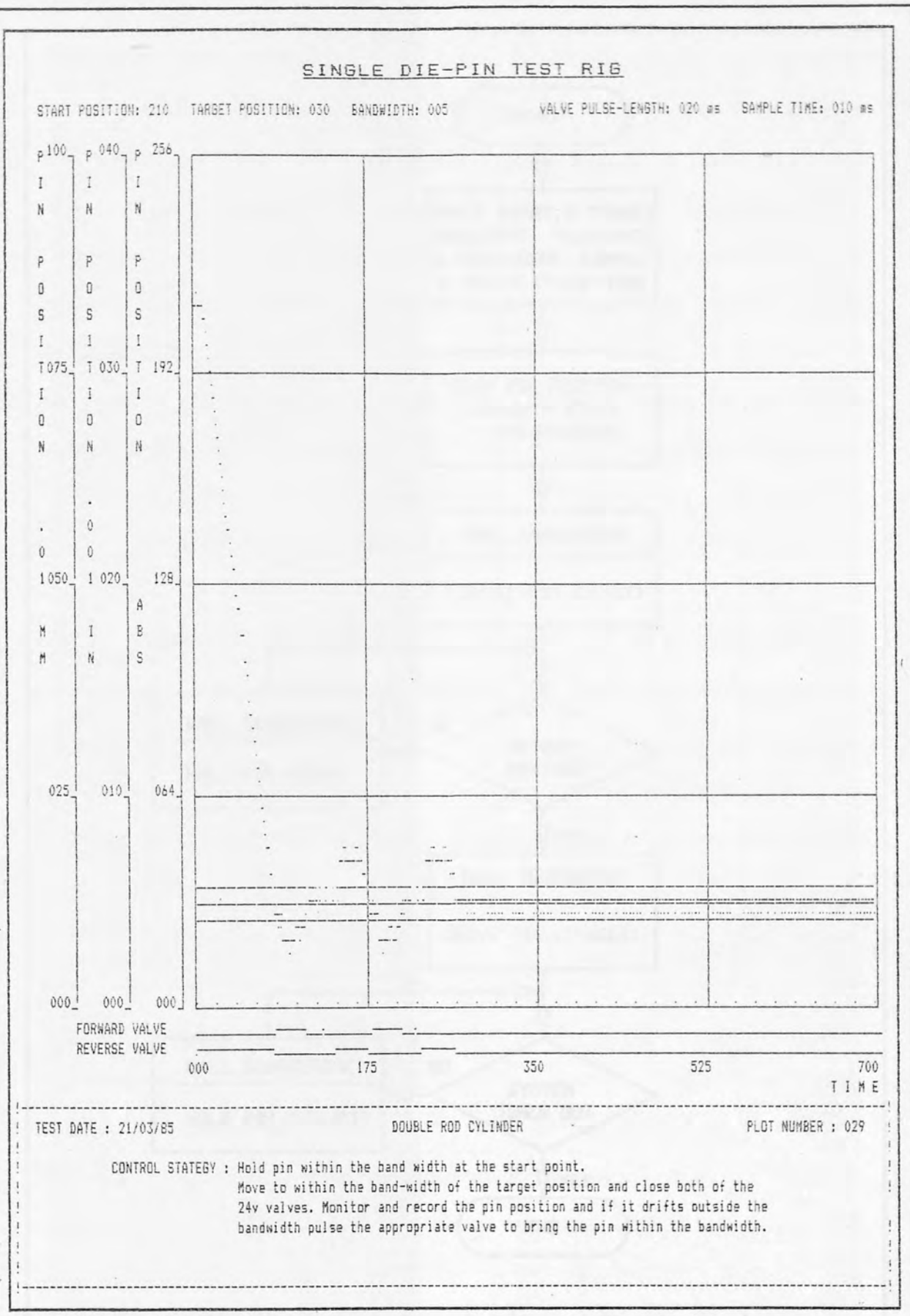


Figure 38 : Sample Print Out using Simple Control Logic

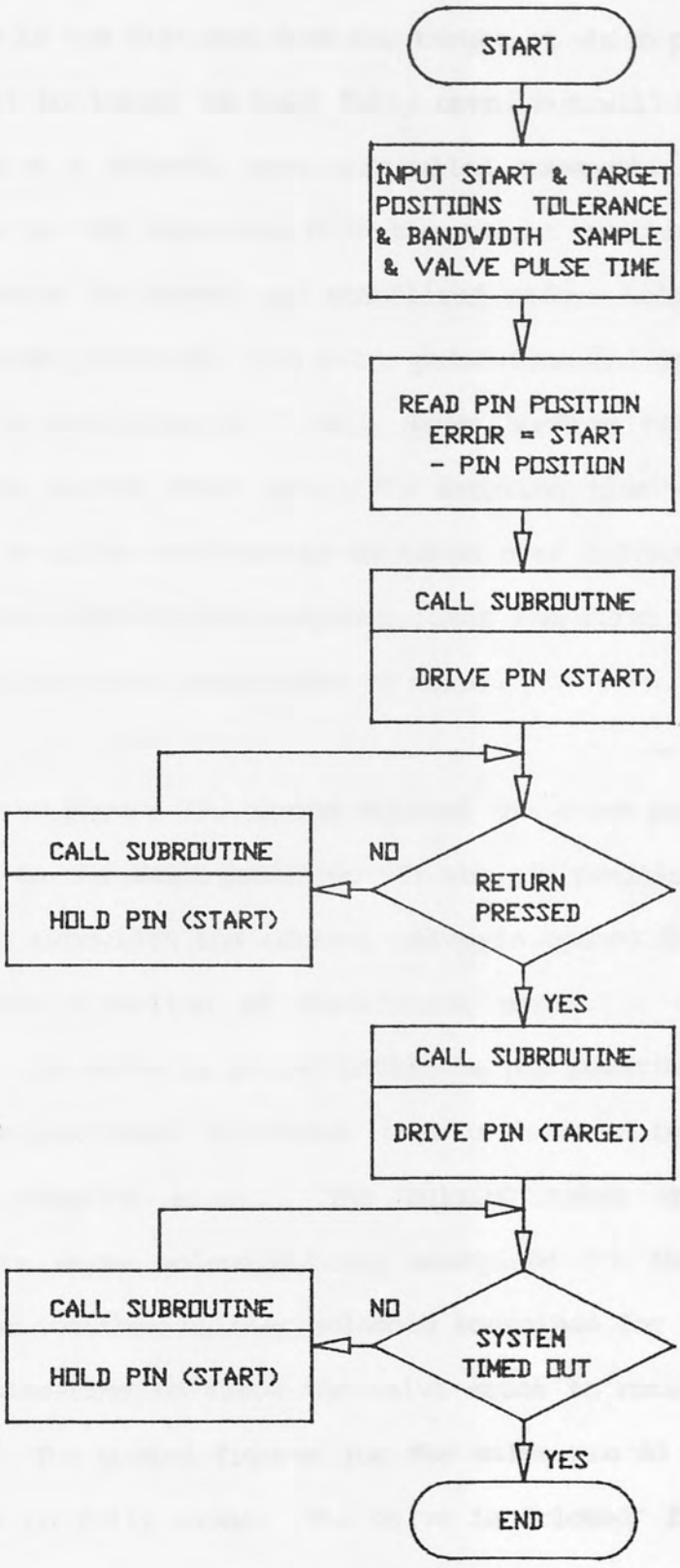


Figure 39 : Test Rig 2 Control Software Flowchart

The first two parameters are self-explanatory. The positional bandwidth is the distance from the target at which point the control valve will no longer be held fully open, but will be pulsed on and off to give a slower, more controlled movement. The positional tolerance is the distance from the target position at which the control valve is closed and stabilised before being driven to the actual target position. The valve pulse-time and data sampling time are set in multiples of 5 ms. Seven hundred readings are taken before the system times out. The sampling time was entered as a variable to allow readings to be taken over different time periods for example, initial pin response (over the first second) and long term stability (over one minute or more).

Referring to Figure 39, having entered the above parameters the pin is driven to the start position. If the pin position is outside the positional bandwidth the control valve is opened fully to move the pin in the direction of diminishing error. Once within the bandwidth the valve is pulsed until the pin position has reached or passed the positional tolerance (this is detected by the error being zero or changing sign). The pulsing takes the form of the appropriate valve solenoid being energised for the length of the pulse-time and then neither solenoid energised for twice the length of the pulse-time to allow the valve spool to return to its centre position. The quoted figures for the valve are 40 ms to fully open and 50 ms to fully close. The valve is "closed" for twice as long as it is held open to prevent the valve cumulatively opening (as happened in initial trials). Experiment showed that a pulse-time of 5 ms was insufficient to move the pin and of 15 ms caused too great a movement of the pin, thus a pulse-time of 10 ms was used.

Having reached the positional tolerance point the control valve is stabilised first by de-energising the driving solenoid and pulsing the other solenoid twice to drive the valve spool to the centre position, and secondly by alternately pulsing both sides of the valve four times to equalise the pressure on the two sides of the cylinder piston (experiment showed that stability of the valve was always achieved within four oscillations). The position of the pin is monitored during the stabilising action and if the sign of the pin position error changes, stabilising is aborted since movement of the pin in the previously driven direction has obviously been arrested. Stabilising of the valve was carried out as it was found that the valve had a tendency to stick open in the direction in which it was last driven. This may be seen in Figure 40. The change from the valve being fully open to pulsing, once the bandwidth position is achieved, is clearly visible. The plot shows that once the target position is achieved the driving solenoid is de-energised but the pin is still moving as the valve closes, and continues to move when the valve closing time has elapsed.

Once the valve has stabilised it is driven by pulsing until it reaches or passes the target point, when the valve is again stabilised. The pin position is then monitored and if it moves outside the tolerance band about the target position, the valve is pulsed until the target position is again achieved and the valve is then stabilised.

The above control strategy is used first to move the die-pin to the specified start position, where it is held until an input from the system terminal signals that the pin is to be driven to the target

position, where it is held until the system times out. The pin positions that have been recorded during the movement of the pin from the time that the input from the terminal starts movement from the start position, until the system times out, can then be plotted out. A typical plot is shown in Figure 41. Referring to Figure 41, the ordinate of the graph represents the pin movement in units of 0.001". The start and target positions, bandwidth and tolerance are shown in the LVDT count units (i.e. the digital output from the A/D) with 255 counts equating to 0.040" of pin movement. The three dashed lines indicate the target and tolerance band positions. The state of the forward and reverse solenoids are shown at the bottom of the plot, and the values of the six variable parameters are shown at the top of the plot. In this test the pin was moving away from the spring-loaded support pin. The step in pin movement at the 0.020" position is due to the loss of contact with the support pin. Stabilising of the valve first at the tolerance value and secondly at the target value can clearly be seen. In both cases the reverse direction solenoid was open prior to stabilising thus the forward solenoid was pulsed twice (to arrest the movement of the pin) followed by alternation of the two solenoids in order to centralise the valve spool. As can be seen, approximately 0.030" of controlled pin movement was achieved within two seconds.

A listing of the software is given in Appendix I.

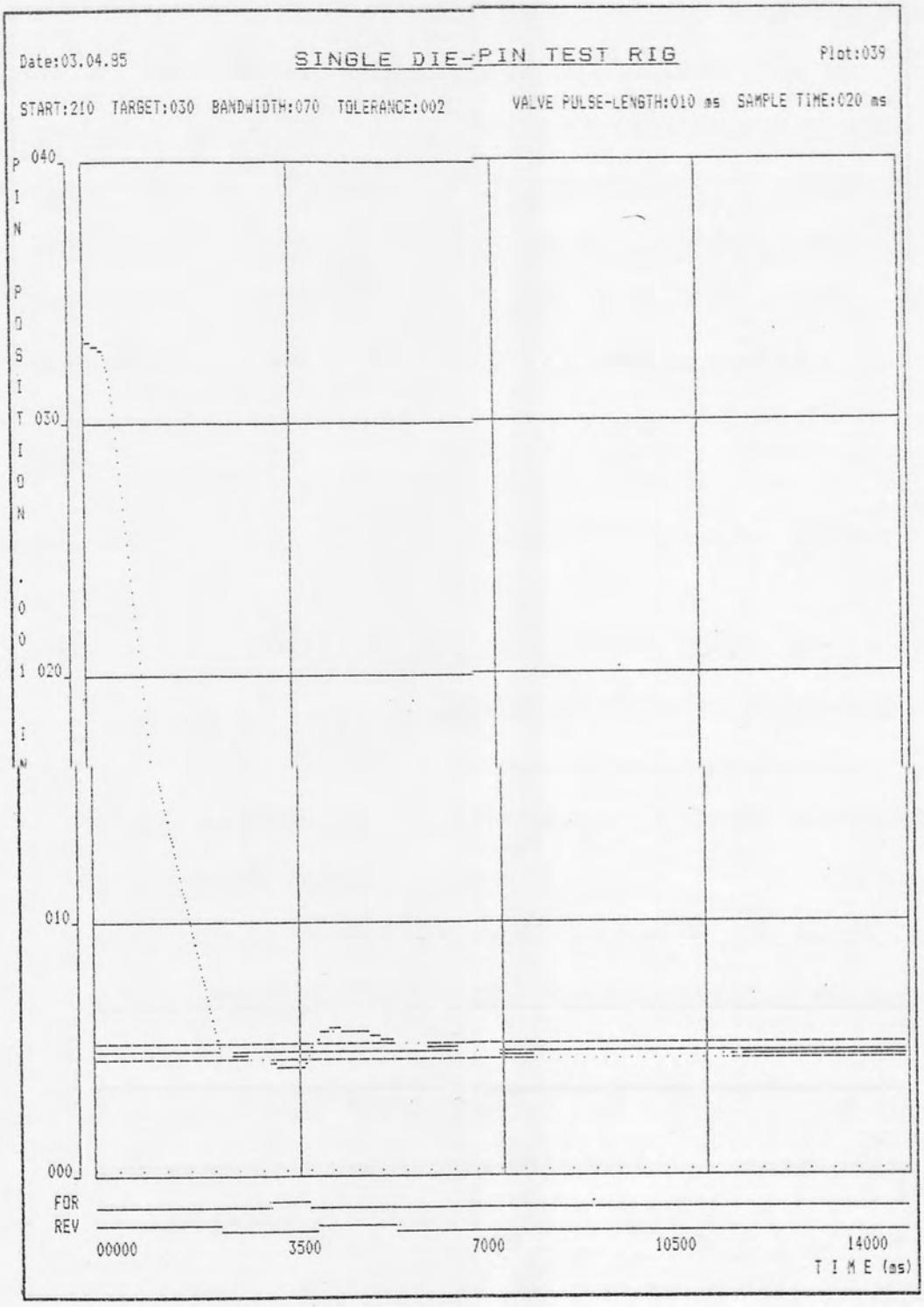


Figure 40 : Sample Print Out Showing Valve Sticking

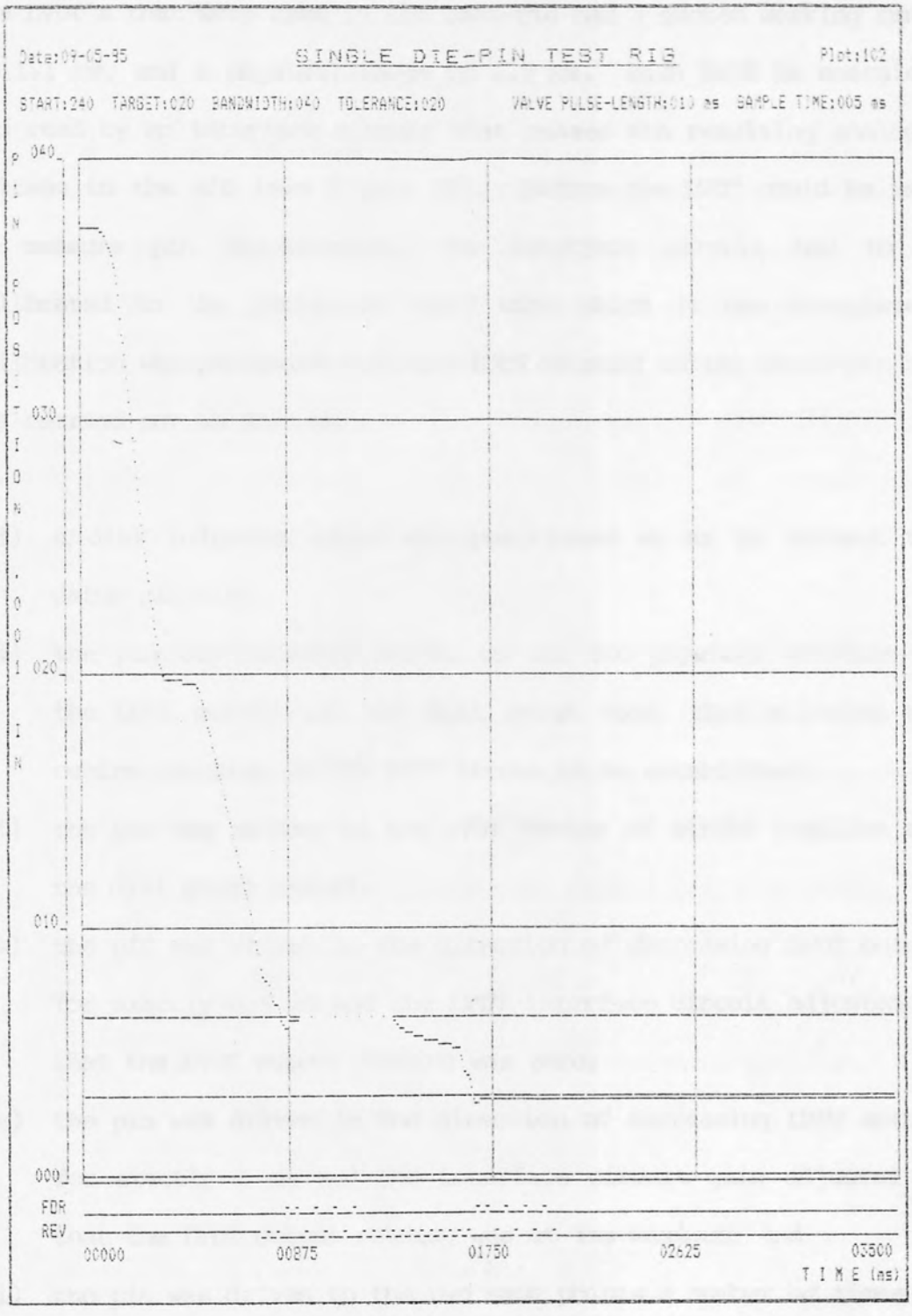


Figure 41 : Sample Print Out using Refined Control Strategy

11.5.3. LVDT Calibration

The LVDT's that were used in the test-rig had a quoted working range of 1.1 mm, and a physical range of 1.7 mm. Each LVDT is energised and read by an interface circuit that passes the resulting analogue voltage to the A/D (see Figure 33). Before the LVDT could be used to measure pin displacement, the interface circuit had to be calibrated to the particular LVDT with which it was associated. Calibration was performed with the LVDT mounted on the datum pin and was carried out as follows :

i) a dial indicator gauge was positioned so as to contact the datum pin tip;
ii) the pin was manually driven to the two physical extremes of the LVDT stroke and the dial gauge read, thus allowing the centre position of the LVDT stroke to be established;
iii) the pin was driven to the LVDT centre of stroke position and the dial gauge zeroed;
iv) the pin was driven in the direction of decreasing LVDT output for exactly 0.5 mm and the LVDT interface circuit adjusted so that the LVDT output reading was zero;
v) the pin was driven in the direction of increasing LVDT output for exactly 1 mm and the interface circuit gain adjusted so that the LVDT output reading was at its maximum; and
vi) the pin was driven to the two same points a number of times to check the calibration.

In order to manually drive the pin and to read the LVDT output, a computer program was written (listed in Appendix II). This allowed the pin control value to be single pulsed in each direction by

pressing the appropriate key on the terminal keyboard, and gave a continuous read-out on the terminal screen of the LVDT output.

11.5.4 Pin position calibration

Having calibrated the LVDT interface circuitry, the LVDT's had to be mounted on the respective pins in the test-rig and the exact reading of both LVDT's established at the two pin datum positions (Figure 42 (a)). Again the computer program listed in Appendix II was used to both manually drive the datum pin and to read the LVDT values. The procedure for calibrating the datum pin LVDT is as follows (with reference to Figure 42 (b)):

i) with the datum pin retracted from the die cavity, inspection quality slip-gauges are placed against the die datum face and used to zero a dial - indicator gauge at the datum pin datum position;

ii) the datum pin is driven forward to contact the dial gauge. It is not necessary for the pin to be positioned at the dial gauge zero position. Let the dial gauge reading be E_p that is the distance of the datum pin from its datum position;

iii) convert E_p to LVDT output units (C_p) (255 counts = 0.040"); and

iv) adjust LVDT mounting position so that the LVDT reading is approximately 127 + C_p counts (127 counts is the LVDT mid-stroke position). Let this LVDT reading be C_a. The datum pin LVDT datum reading is then given by $C_a - C_p$, and will be referred to as the datum pin LVDT calibration value.

Having established the datum pin LVDT calibration value the same must be done for the support pin LVDT. The procedure is as follows :

i) without moving the datum pin, slip-gauges approximating to the nominal blade thickness (let the error be E_b) are inserted between the datum and support pins;

ii) convert E_b to LVDT counts (C_{bb});

iii) adjust the LVDT mounting position so that the LVDT reading is approximately 127 + C_b + C_{lp} counts. Let this reading be C_s; and

iv) the support pin LVDT calibration value is given by $C_s - C_b - C_p$.

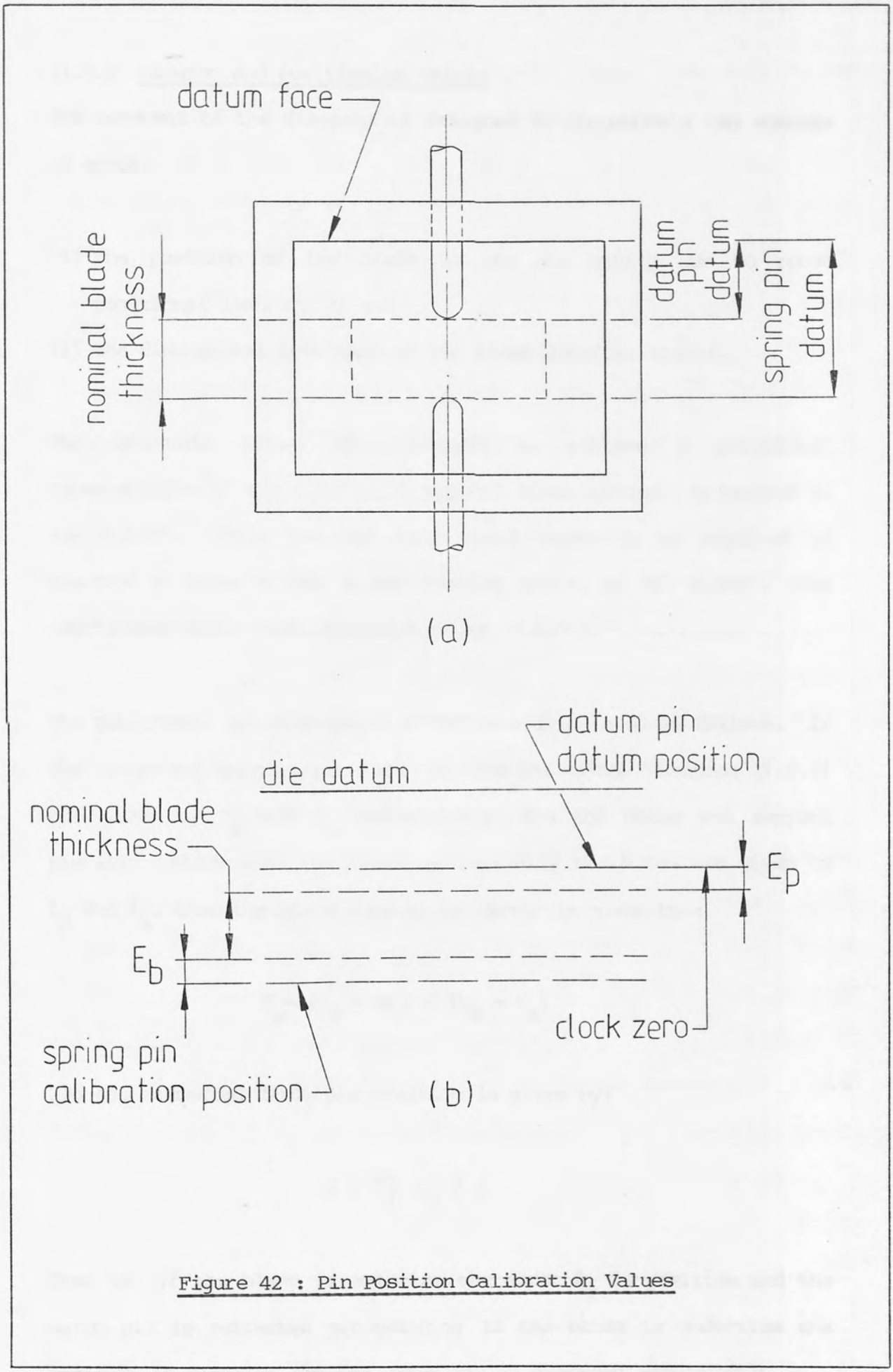


Figure 42 : Pin Position Calibration Values

11.5.5 Gauging and positioning trials

The movement of the die-pins is designed to accommodate two sources of error:

i) the position of the blade in the die cavity due to robot positional inaccuracy; and
ii) the dimensional tolerance of the blade aerofoil itself.

The Cybermate robot was designed to achieve a positional repeatability of +/- 0.004". A typical blade aerofoil tolerance is +/- 0.005". Thus the die pins would typically be required to position a blade within a pin working stroke of +/- 0.009". (The LVDT's used had a working stroke of +/- 0.020").

The positional and dimensional errors are calculated as follows. If the datum and support pin LVDT calibration values (Section 11.5.4) are given by C_d and C_s respectively, and the datum and support pin LVDT values when the blade is initially contacted are given by L_d and L_s, then the blade dimensional error is given by :

$$E_w = (L_d - C_d) - (L_s - C_s)$$

and the corrected datum pin position is given by:

$$X = C_d - E_w / 2$$

That is, if the blade is oversize the error E_w is positive and the datum pin is retracted accordingly; if the blade is undersize the error E_w is negative and the datum pin is extended accordingly.

Figure 43 shows the the logic of the single pin gauging and positioning software.

The gauging software was initially tested by the use of inspection quality slip-gauges to represent the blade. Once the software had been shown to be working correctly it was modified in order to continuously cycle the test-rig to examine the consistency of the results of blade gauging. A listing of this software is given in Appendix III. By using slip-gauges that represented an oversize blade such that both pins were constantly in contact with the blade, the rig could be left to cycle indefinately without manual intervention. A typical print-out from such a test is shown in Figure 44.

The initial cycle trials highlighted a problem which was primarily caused by the support pin having its calibration value set near to one end of its working stroke rather than at its mid-position. Occasionally, prior to gauging of the blade, the pair of pins would move to a point where the support pin LVDT had passed the end of its working stroke. Once past this point the output from the LVDT was static at 255 counts. However, the output from the datum pin LVDT continued to increase as the pins moved. This resulted in inaccurate gauging and final positioning of the blade. The problem was resolved first by ensuring that both LVDT calibration values occurred at the LVDT working stroke mid-positions, and secondly by modifying the control software so that the datum pin, having contacted the blade, moves to its datum position (the LVDT calibration value), prior to blade gauging.

Gauging and positioning trials were carried out using four simulated blade thicknesses (0.209", 0.215", 0.220" and 0.230") with 0.220" being the nominal blade thickness used for pin position calibration (Section 11.5.4). These trials showed the test-rig to be capable of consistently gauging to an accuracy of +/- 0.00015" (one LVDT count) and of consistently positioning the blade to an accuracy of +/- 0.00045" (three LVDT counts). A sample plot is shown in Figure 45.

Following completion of these tests preliminary casting trials were undertaken to investigate the effects of molten and solidifying metal on the die pin movement.

The stability of the die-pins in the empty die cavity and in the cavity when filled with solidifying "Cerrotru" encapsulation alloy were comparable. No difficulty was experienced with extracting either the datum or the support pin from the solidified encapsulation block. During pouring and cooling of the "Cerrotru" the die block temperature rose to approximately 40 degrees centigrade (the metal pouring temperature was 161 degrees centigrade). Both the datum and support pin LVDT's remained at room temperature. In a commercial die (with a cycle time of approximately 30 seconds), water cooling of the die might be necessary in order to prevent heating of the die-pin LVDT's. Alternatively, or in addition, the spring loaded support pins could be insulated from the die block using ceramic bushes, as with the datum die-pins.

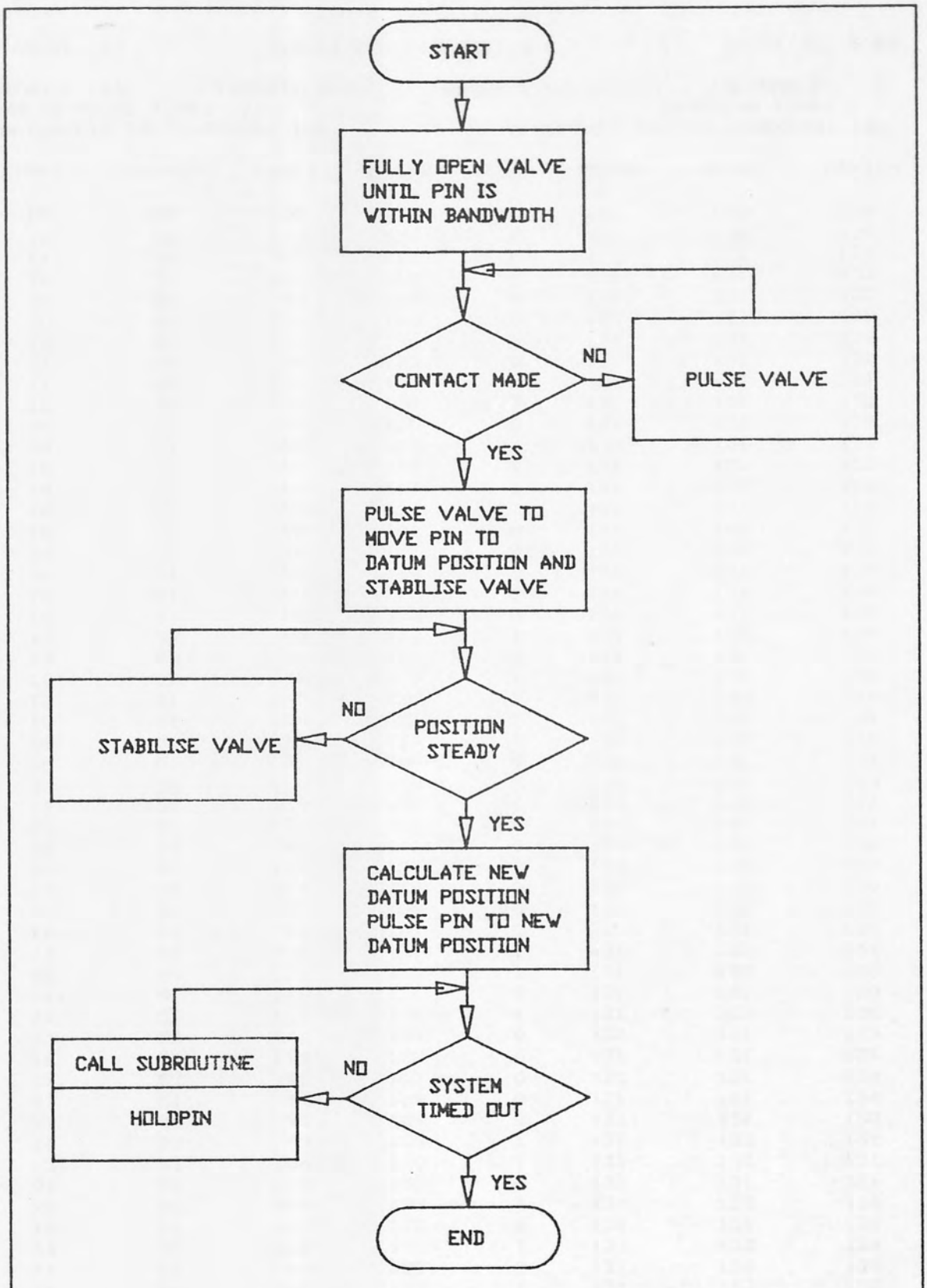


Figure 43 : Software Flowchart for Blade Gauging

TABLE: 10 SINGLE DIE-PIN TEST RIG DATE: 30. 5.85

START: 15 TARGET: 200 BANDWIDTH: 150 TOLERANCE: 3
VALVE PULSE TIME: 10 SAMPLING TIME: 5
DATUM PIN CALIBRATION: 131 SUPPORT PIN CALIBRATION: 146

START	CONTACT	LVDT1	LVDT2	TOL	DATUM	TARGET	FINISH
19	52	120	136	-1	131	130	128
15	50	113	129	-1	131	130	129
17	51	97	112	0	131	131	131
26	52	101	116	0	131	131	131
30	50	98	113	0	131	131	132
30	50	121	136	0	131	131	130
16	51	121	136	0	131	131	132
24	51	98	113	0	131	131	132
17	50	110	125	0	131	131	131
15	52	94	109	0	131	131	132
26	51	105	120	0	131	131	128
28	51	93	108	0	131	131	131
15	50	100	114	1	131	132	133
19	50	103	117	1	131	132	134
14	50	99	114	0	131	131	134
18	52	91	106	0	131	131	130
22	52	110	125	0	131	131	130
28	51	99	114	0	131	131	132
15	51	100	115	0	131	131	129
13	52	98	113	0	131	131	132
13	52	98	112	1	131	132	129
13	50	96	111	0	131	131	130
16	50	96	110	1	131	132	130
13	51	121	135	1	131	132	131
15	51	108	123	0	131	131	129
16	51	110	124	1	131	132	131
18	51	89	104	0	131	131	129
34	51	120	135	0	131	131	132
14	50	109	123	1	131	132	132
30	51	96	111	0	131	131	131
32	51	96	111	0	131	131	130
35	51	121	135	1	131	132	133
17	50	119	134	0	131	131	130
25	52	93	108	0	131	131	130
16	52	92	107	0	131	131	133
17	51	109	123	1	131	132	131
15	50	119	133	1	131	132	130
31	51	116	131	0	131	131	130
24	52	117	131	1	131	132	135
33	53	114	129	0	131	131	133
16	50	114	129	0	131	131	134
27	52	92	107	0	131	131	132
27	51	94	109	0	131	131	134
27	50	94	109	0	131	131	130
24	52	92	106	1	131	132	132
33	53	106	120	1	131	132	131
27	51	118	133	0	131	131	131
25	53	108	122	1	131	132	134
12	51	107	122	0	131	131	130
34	50	116	130	1	131	132	134
31	52	90	105	0	131	131	129
16	50	108	122	1	131	132	130
29	51	93	107	1	131	132	132
17	51	119	134	0	131	131	129

Figure 44 : Typical Print Out for Cyclic Testing of Blade Gauging

TABLE: 11 SINGLE DIE-PIN TEST RIG DATE: 7. 6.85

START: 15 TARGET: 200 BANDWIDTH: 150 TOLERANCE: 3
VALVE PULSE TIME: 10 SAMPLING TIME: 5
DATUM PIN CALIBRATION: 76 SUPPORT PIN CALIBRATION: 87

START	CONTACT	LVDT1	LVDT2	TOL	DATUM	TARGET	FINISH
13	52	46	57	0	76	76	73
11	50	48	58	1	76	77	74
16	51	68	78	1	76	77	74
14	50	63	73	1	76	77	77
24	50	60	70	1	76	77	76

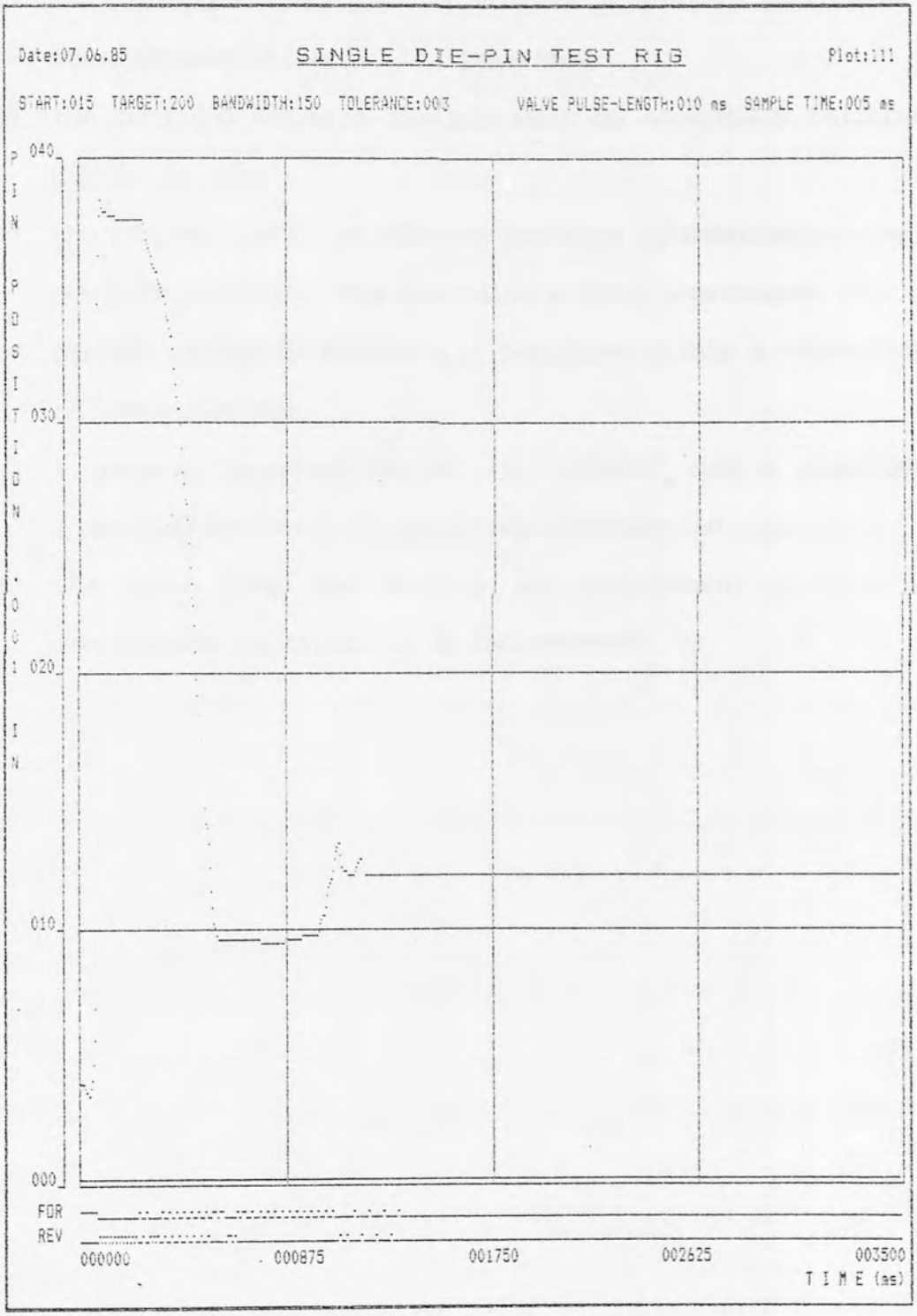


Figure 45 : Sample Plot of Pin Positioning during Blade Gauging

11.6 Test Rig Conclusions

The following conclusions were drawn from the test-rig trials :

i) a 1 mm measuring stroke is required for the die-pins for the levels of positional and blade tolerance errors anticipated;

ii) each LVDT interface circuit must be accurately calibrated to the LVDT with which it will interface;

iii) the position of each die pin must be accurately calibrated within the die;

iv) the proposed method of die-pin position calibration is simple, accurate and fast, thus providing a great improvement over the present system of setting pin positions within a conventional encapsulation die;

v) a gauging repeatability of +/- 0.00015" and a positioning repeatability of +/- 0.00045" was demonstrated; and

vi) the cycle time for gauging and positioning a blade was demonstrated to typically be two seconds.

12.0 THE PROTOTYPE HANDLING SYSTEM

12.1 The Encapsulation Die

12.1.1 Test Blades

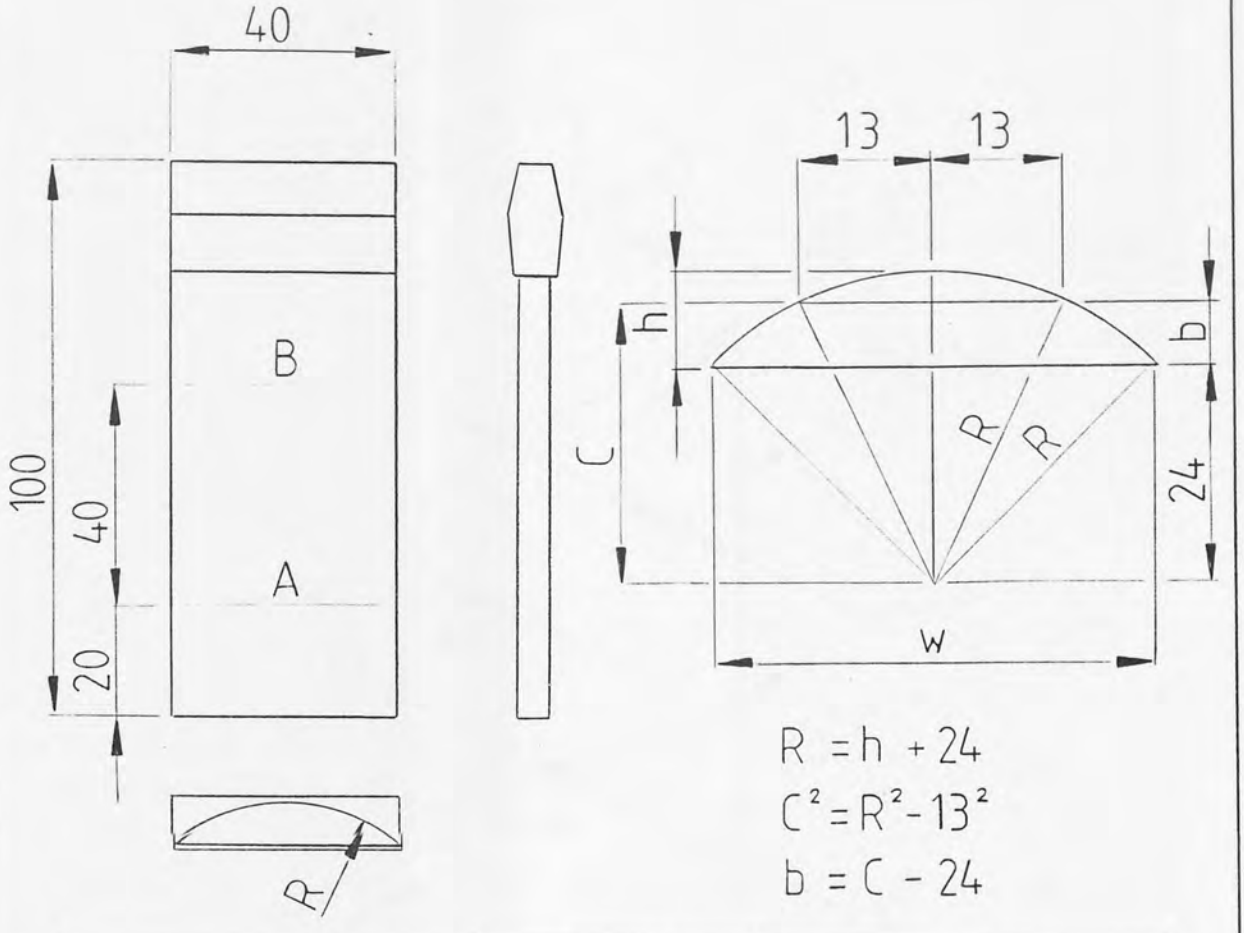
In order to simplify the design of the experimental encapsulation die it was decided to manufacture a number of test blades that would simulate the form of a turbine blade. The design of these blades is shown in Figure 46 together with the calibrated blade dimensions. The blades were produced in mild steel.

12.1.2 Mechanical design

The encapsulation die is an enhancement of the Test-Rig 2 (Section 11.4). The single spring-loaded pin assembly was replaced by a second hydraulic cylinder assembly (so that this pin could be retracted to allow repositioning of the blade in the perpendicular plane) and four other pairs of hydraulic cylinder assemblies were added. This arrangement represents three aerofoil datum pins and two leading-edge datum pins, together with their associated clamp pins (see Figure 47 for the pin identification). This arrangement gives two dimensional control of the blade position (x and y axis movement).

The encapsulation die design was again simplified by the decision that a platform pin would not be included. Three dimensional movement of the blade was considered to be the next step once two dimensional control had been demonstrated. A knife edge was thus included in the base of the die to simulate the platform location (see Figure 49).

Figures 48 and 49 show the encapsulation die.



$R = h + 24$

$C^2 = R^2 - 13^2$

$b = C - 24$

		measured		calculated		
	D=2R	h	w	R	C	b
Blade 1 A	59·900	5·900	33·896	29·900	26·926	2·926
B	59·875	5·870	33·896	29·870	26·893	2·893
Blade 2 A	59·800	5·819	33·985	29·819	26·836	2·836
B	59·775	5·781	33·985	29·781	26·794	2·794
Blade 3 A	59·700	5·845	33·934	29·845	26·865	2·865
B	59·675	5·804	33·922	29·804	26·819	2·819
Blade 4 A	59·600	5·738	34·036	29·738	26·746	2·746
B	59·575	5·738	34·036	29·702	26·706	2·706

Figure 46 : Test Blade Design

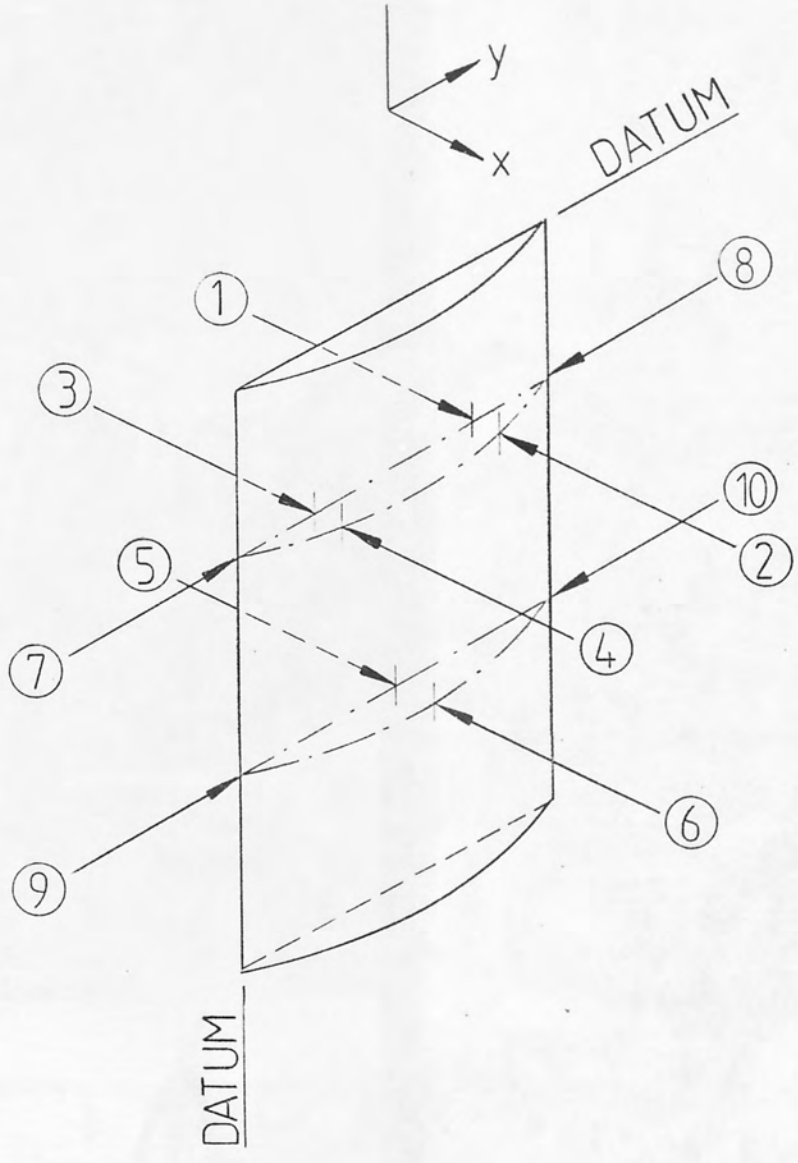


Figure 47 : Test Die Pin Identification

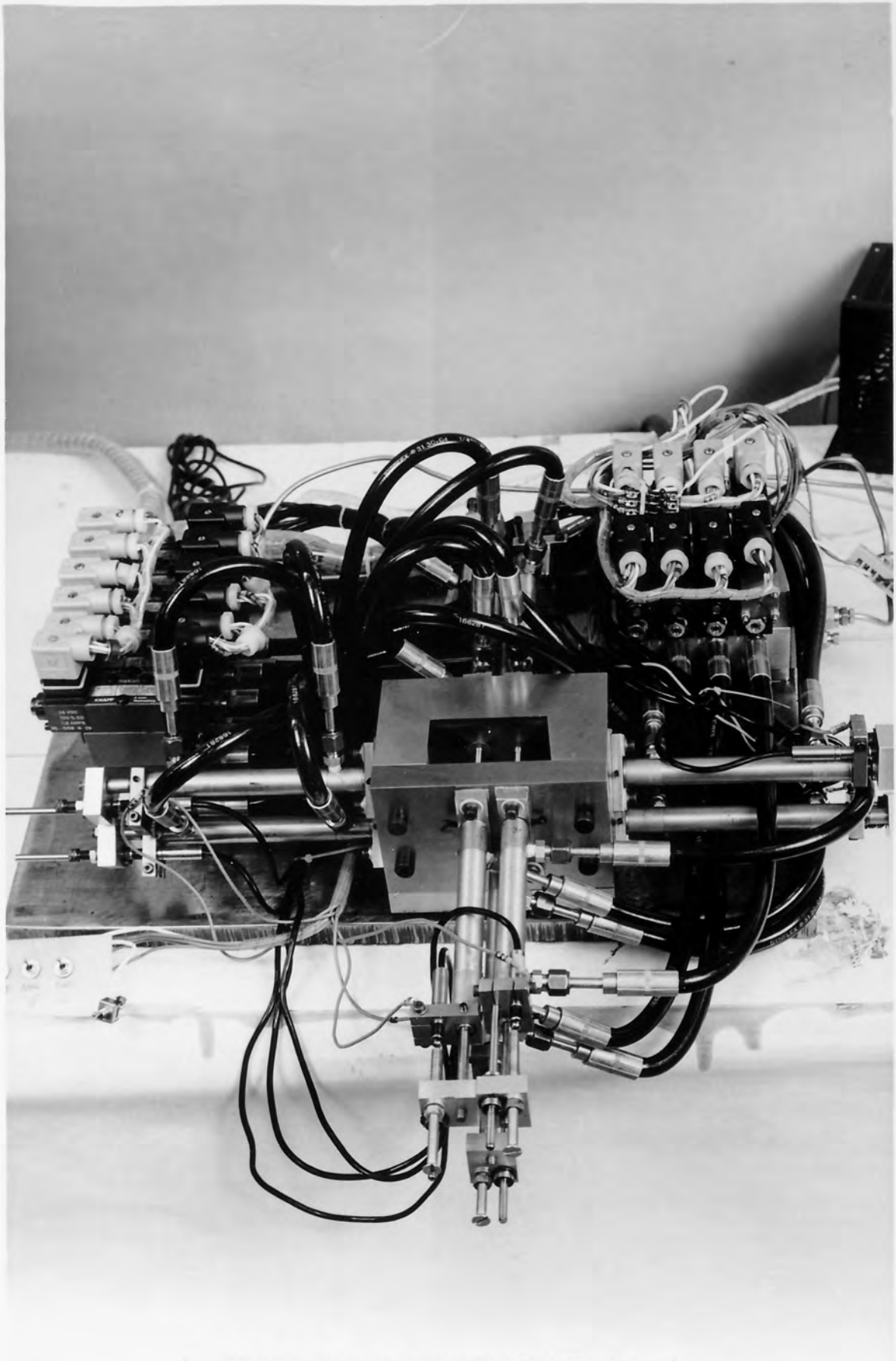


Figure 48 : The Encapsulation Die

12.1.3 Control software strategy

Figure 50 shows the flowchart for the software control of multiple pins. The control loop steps are as follows :

i) once an input signal has been received that a blade is waiting to be loaded, all pins are driven forward until each pin is within the transducer range and then each pin is backed off 0.010" from the nominal pin position. An output signal that the die is ready to be loaded is set;

ii) when an input signal has been received that a blade has been loaded into the die all pins are pulsed forward to clamp the blade;

iii) if all pins contact the blade within a set time limit an output signal is set that the robot may release the blade and withdraw, otherwise all the pins are retracted from the die cavity and an output signal is set that an error has occurred, that is, the blade has been mis-loaded (the loop will then restart from step (i));

iv) once an input signal has been received that the robot has withdrawn the blade is moved to the nominal datum position and an output signal set;

v) the blade is gauged and if necessary repositioned and an output signal is set;

vi) when an input signal has been received that pouring of the die is in progress, the position of each pin is recorded every 60 ms for 3 minutes; and

vii) all pins are withdrawn from the die cavity and the results of the pin monitoring are printed.

The software was designed with all necessary interlocks between the die and the robot being included. Input signals from the robot were simulated using manually operated switches and output signals from the die were indicated by signal lamps.

A listing of the software (program MULTIPIN) is given in Appendix V.

12.1.4 Control software : modifications to Test Rig 2

Control of individual pins differs in three ways :

i) in order to move the blade in one axis, the pins of the other axis must first be withdrawn from contact with the blade (procedure BACKOFF-PINS), for example to move the blade in the X-axis the Y-axis pins must first be withdrawn. The back-off distance was set to 0.005";

ii) each datum pin and clamp pin may be moved individually or as a pair. When clamping the blade, backing-off or withdrawing from the die cavity, each pin is moved individually; when positioning the clamped blade the appropriate datum and clamp pin are moved as a pair (procedure MOVE-PINS-SLOW); and

iii) the combined fast, coarse positioning followed by slow, pulsed positioning of pins was not used. For bringing the pins onto the transducers or pin withdrawal each valve was opened fully. For clamping of the blade, backing-off of pins or blade positioning, valve pulsing was used.

The blade is gauged using the same logic as using on Test-Rig 2 (Section 11.5.5).

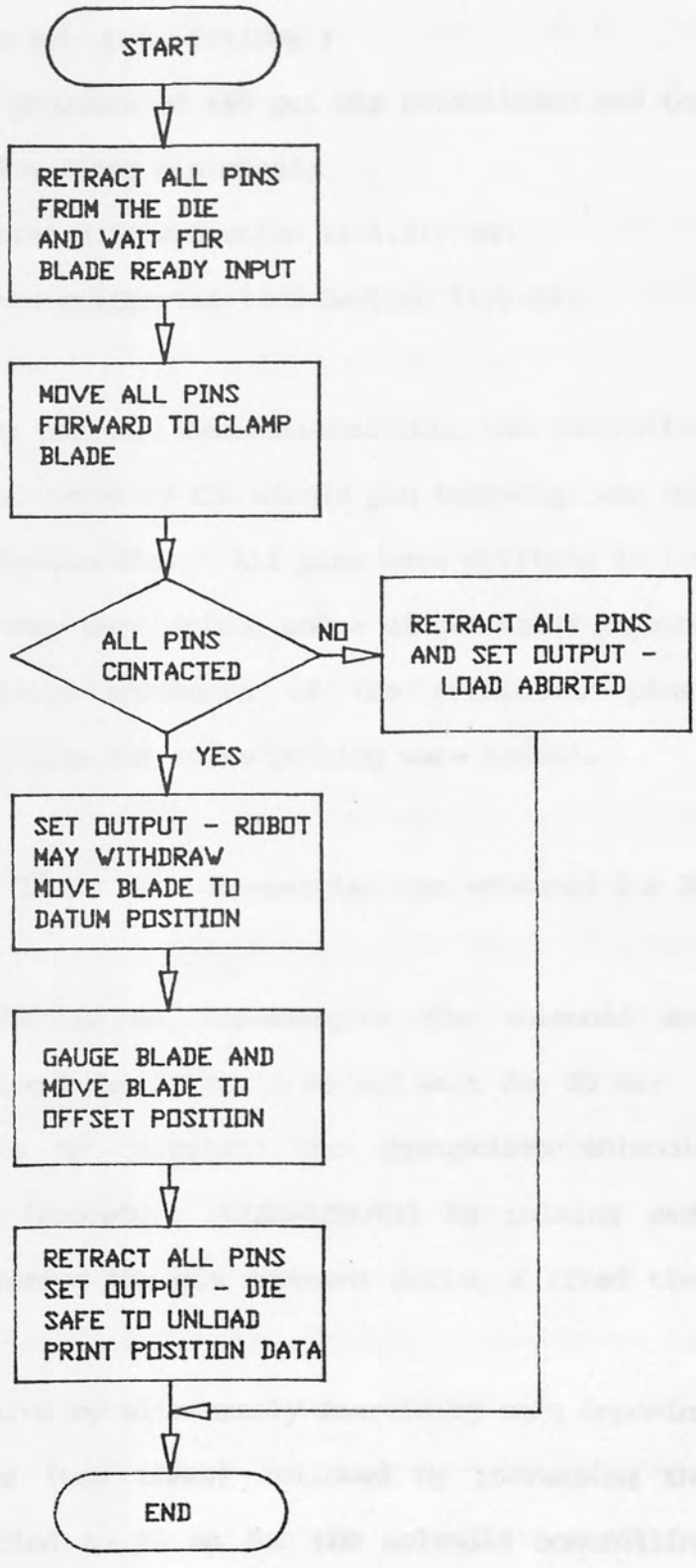


Figure 50: Multiple Pin Control Software Flowchart

12.2 Software Development

The encapsulation die was set up as follows :

i) a hydraulic system pressure of 650 psi was established and the individual valve flow rates minimised;

ii) each LVDT was calibrated (see Section 11.5.3); and

iii) each pin position was calibrated (see Section 11.5.4).

The initial trials of the software demonstrated that the controlled movement of each pin as achieved by the single pin test-rig, was not occurring in the encapsulation die. All pins were drifting in the last direction in which they were driven and a single valve pulsing produced unacceptably large movements of the particular pins. Several alternative strategies for valve pulsing were tested:

i) open the valve for 10 ms then de-energise the solenoid for 20 ms;

ii) open the valve for 10 ms, de-energise the solenoid and energise the opposing solenoid for 5 ms and wait for 20 ms;

iii) allow the software to calculate the appropriate solenoid energising period (procedure PULSE-LENGTH) by pulsing each solenoid and monitoring the pin movement during a fixed time period (1.5s); and

iv) oscillating the valve by alternately energising each opposing solenoid for 5 ms (ten times) followed by increasing the energised time period to 15 ms for the solenoid controlling the desired direction of movement until the pin has moved.

The first strategy was that used with the single pin test-rig. A single pulse produced movements as great as 100 LVDT counts (0.015") with the pin continuing to move after completion of the pulse.

The second strategy was devised in order to improve the response of the valve spool following the valve opening pulse. Trials with the single pin test-rig had shown that a pulse-time of 5 ms was too short to cause movement of the pin (Section 11.5.2). Thus following the 10 ms valve opening pulse, the opposing solenoid was energised for 5 ms in order to assist the spool in its return to the centre position, without causing movement of the pin in the opposite direction. Following this short, opposing pulse, the valve was allowed to settle for 20 ms as before.

The third strategy was an attempt to use the power of the controlling computer to overcome the problem of the accuracy with which the flow restrictors could be adjusted. As shown in Figure 37, each valve was fitted with a pair of variable flow restrictors. These restrictors were adjusted to give a minimum flow rate that was just sufficient to allow the pin to move. Adjustment was by means of a fairly coarse screw thread and locking nut, thus fine adjustment was difficult to achieve. Since the flow rate (as set by the restrictor) and valve pulse time are inversely related this strategy was devised to allow the computer to compensate for differences between the settings of individual restrictor/valve pairs by adjusting the valve pulse-time. The procedure SET-VALVE-PULSE-LENGTHS was written to test this strategy. A nominal pulse time of 15 ms was set for each pin. The forward pulse-time for each pin was considered in numerical pin sequence, followed by the reverse pulse-time of each pin. The logic for assessing one pin was as follows:

i) read the pin position;

ii) pulse the valve once;

iii) wait for 1.5 s to allow the pin to move and settle;

iv) read the pin position;

v) if the change in position is greater that 10 LVDT counts (0.0015") reduce the pulse-time by 1 ms (to a minimum of 5 ms);

vi) if the change in position is less than 4 LVDT counts (0.0006") increase the pulse-time by 1 ms; and

vii) if the pulse-time has been changed, repeat steps (i) to (vi).

During the above cycle the position of the pin was monitored and if near to the end of the LVDT stroke in the driven direction, the pin was moved to the other end of the LVDT range before pulsing continued.

Trials showed significant differences in the number of attempts needed to set the pulse-times for individual pins. This setting operation took as long as three minutes and as little as ten seconds per pin. Having set these values, the pulse-times were printed out (see Figure 51). Some solenoids show consistency (for example pins 1-7 forward and 1, 3-6 reverse), others do not.

Having set the individual value pulse-times in the above manner, trials showed that the pins could still not be controlled to the previously achieved accuracy. Thus, a fourth control strategy was devised.

This strategy attempted to improve the response time of the valve solenoids. The control of a single pin was as follows :

i) both solenoids are pulsed alternatively at a level that will not cause pin movement (5 ms), thus raising the energy level of the solenoids;

ii) the pulse-time for the solenoid controlling the required direction of movement is then increased to 15 ms and both solenoids continue to pulse alternatively until the pin has moved; and

iii) both solenoids are again pulsed alternately with a pulse-time of 5 ms to stabilise the valve, prior to de-energising both solenoids.

This strategy was also demonstrated to be unable to achieve the required accuracy of control of pin movement.

the first strategy was that used with the single pin test-rig. A single pulse produced movements as great as 100 LVDT counts (0.015") with the pin continuing to move after completion of the pulse.

The second strategy was devised in order to improve the response of the valve spool following the valve opening pulse. Trials with the single pin test-rig had shown that a pulse-time of 5 ms was too short to cause movement of the pin (Section 11.5.2). Thus following the 10 ms valve opening pulse, the opposing solenoid was energised for 5 ms in order to assist the spool in its return to the centre position, without causing movement of the pin in the opposite direction. Following this short, opposing pulse, the valve was allowed to settle for 20 ms as before.

The third strategy was an attempt to use the power of the controlling computer to overcome the problem of the accuracy with which the flow restrictors could be adjusted. As shown in Figure 37, each valve was fitted with a pair of variable flow restrictors. These restrictors were adjusted to give a minimum flow rate that was just sufficient to allow the pin to move. Adjustment was by means of a fairly coarse screw thread and locking nut, thus fine adjustment was difficult to achieve. Since the flow rate (as set by the restrictor) and valve pulse time are inversely related this strategy was devised to allow the computer to compensate for differences between the settings of individual restrictor/valve pairs by adjusting the valve pulse-time. The procedure SET-VALVE-PULSE-LENGTHS was written to test this strategy. A nominal pulse time of 15 ms was set for each pin. The forward pulse-time for each pin was considered in numerical pin sequence, followed by the reverse pulse-time of each pin. The

logic for assessing one pin was as follows:

i) read the pin position;

ii) pulse the valve once;

iii) wait for 1.5 s to allow the pin to move and settle;

iv) read the pin position;

v) if the change in position is greater that 10 LVDT counts (0.0015") reduce the pulse-time by 1 ms (to a minimum of 5 ms);

vi) if the change in position is less than 4 LVDT counts (0.0006") increase the pulse-time by 1 ms; and

vii) if the pulse-time has been changed, repeat steps (i) to (vi).

During the above cycle the position of the pin was monitored and if near to the end of the LVDT stroke in the driven direction, the pin was moved to the other end of the LVDT range before pulsing continued.

Trials showed significant differences in the number of attempts needed to set the pulse-times for individual pins. This setting operation took as long as three minutes and as little as ten seconds per pin. Having set these values, the pulse-times were printed out (see Figure 51). Some solenoids show consistency (for example pins 1-7 forward and 1, 3-6 reverse), others do not.

Having set the individual value pulse-times in the above manner, trials showed that the pins could still not be controlled to the previously achieved accuracy. Thus, a fourth control strategy was devised.

This strategy attempted to improve the response time of the valve solenoids. The control of a single pin was as follows :

i) both solenoids are pulsed alternatively at a level that will not cause pin movement (5 ms), thus raising the energy level of the solenoids;

ii) the pulse-time for the solenoid controlling the required direction of movement is then increased to 15 ms and both solenoids continue to pulse alternatively until the pin has moved; and

iii) both solenoids are again pulsed alternately with a pulse-time of 5 ms to stabilise the valve prior to de-energising both solenoids.

This strategy was also demonstrated to be unable to achieve the required accuracy of control of pin movement.

```
FORWARD & REVERSE VALVE PULSE LENGTHS
PIN :     1    FORWARD :    15    REVERSE :    15
PIN :     2    FORWARD :    17    REVERSE :     5
PIN :     3    FORWARD :    17    REVERSE :     9
PIN :     4    FORWARD :    27    REVERSE :    18
PIN :     5    FORWARD :    14    REVERSE :    20
PIN :     6    FORWARD :     9    REVERSE :    14
PIN :     7    FORWARD :    13    REVERSE :     7
PIN :     8    FORWARD :    16    REVERSE :    17
PIN :     9    FORWARD :     8    REVERSE :    30
PIN :    10    FORWARD :    19    REVERSE :    17
```

```
FORWARD & REVERSE VALVE PULSE LENGTHS
PIN :     1    FORWARD :    15    REVERSE :    15
PIN :     2    FORWARD :    18    REVERSE :    16
PIN :     3    FORWARD :    17    REVERSE :    11
PIN :     4    FORWARD :    29    REVERSE :    17
PIN :     5    FORWARD :    15    REVERSE :    21
PIN :     6    FORWARD :    10    REVERSE :    16
PIN :     7    FORWARD :    13    REVERSE :    16
PIN :     8    FORWARD :    10    REVERSE :    16
PIN :     9    FORWARD :    19    REVERSE :    13
PIN :    10    FORWARD :    15    REVERSE :    16
```

Figure 51: Valve Pulse Time Values

12.3 Positioning and Gauging Trials

The multiple pin software was tested eventhough the required accuracy of pin movement could not be achieved. Strategies (i) (ii) and (iii) were utilised and the following control sequence achieved in all cases:

i) following the withdrawal of all pins from the die cavity the system waits for the input to signal that a blade is waiting;

ii) on receipt of the above input all pins move forward until each is registered on the LVDT;

iii) each pin is pulsed forward to the nominal pin position and then backed-off by 0.005";

iv) the output is set that a blade may now be loaded;

v) following receipt of the above input all pins are pulsed forward until the blade is contacted; and

vi) the blade is moved to the nominal position of the datum pins by alternately backing-off of the Y axis pins to allow pulsed movement of the paired X axis pins, and backing-off of the X axis pins to allow movement in the Y axis.

The inaccuracy of the pin movements caused the system to remain within step (vi) as a continuous loop since the nominal positions of the datum pins could not be achieved (both the X and Y axis pins were hunting).

12.4 Encapsulation Die Trial Conclusions

Following the trials as described in Section 12.3, practical work on the encapsulation die was halted and the following conclusions drawn:

i) further work is needed to understand more fully the behaviour of the control valves in order to improve the pin positioning accuracy and to explain why the accurate control achieved with the single pin test-rig could not be achieved with the encapsulation die; and

ii) the software for the control of the ten die pins had been successfully demonstrated up to and including blade positioning.

13.0 RECOMMENDATIONS FOR FUTURE WORK

13.1 Work Necessary to Complete the Robot

The following work is required:

i) carry out the necessary modifications to fit the second encoder and the tachometer to the robot base (Sections 7.4 and 10.1.1);

ii) complete the design and development of the robot control system; and

iii) establish the accuracy and repeatability of each robot axis and compare with the design specification (Figure 11);

13.2 Work Necessary to Complete The Encapsulation Die

The following work is required:

i) further investigation of the control of individual pins is required in order to more fully understand the parameters that effect the accuracy of pin movement; and

ii) having achieved accurate control of all the pins, the multiple pin software should be run to complete the testing of the logic (the two unproven areas are blade gauging and the storing and output of positional data taken during the loading cycle).

13.3 Encapsulation Die Loading

The following work is required:

i) simulate robot loading of the blade by positioning the sample blades at known offsets from the datum position, then assess the accuracy and repeatability of the encapsulation die with regard to blade re-positioning and blade gauging;

ii) using the robot to load the die, assess the accuracy and repeatability of the system as a whole.

13.4 Further Development of the Prototype Handling System

Having loaded, repositioned and cast the blade in the encapsulation die, the resultant casting is gauged to establish the final position of the blade relative to the encapsulation block datum faces. Work should be carried out to develop a system that feeds the blade positional errors from the gauge back to the encapsulation die. The gauge errors can then be used to modify the encapsulation die pin datum positions in order to reduce the positional errors of the blade in the encapsulation block.

14.0 CONCLUSIONS

1. The high component manufacturing costs of aircraft engine turbine blades justify the automation of all aspects of the manufacturing process. Any automated blade handling system must not mark the surface of the blade aerofoil and should not require any changes to existing blade design.

2. The turbine blade machining process has been successfully automated from post-encapsulation to pre-decapsulation. Since accurate features exist on the blade following machining, decapsulation automation would be straight forward. Since blade encapsulation itself is an automated process, the only manual element of the blade machining cycle is loading of the blade into the encapsulation die.

4. The accuracy necessary for encapsulation die loading cannot be achieved using a standard robot and the existing design of encapsulation die. This problem can be overcome by adding computer controlled movement to the pins in the encapsulation die.

5. The most suitable geometry for a robot in a work cell environment is the cylindrical geometry (the work cell being circular with a diameter of approximately 4m).

6. The characteristics of the prototype Cybermate robot were determined to suit the above general work cell environment with the loading of turbine blades as a specific application. The characteristics were determined as follows:

i) electric drive;

ii) 5 degrees of freedom;

iii) Point-to-Point control;

iv) 15 kg payload capacity; and

v) +/- 0.1mm repeatability.

7. The required controlled die pin movement may be achieved using commercially available, miniature, hydraulic cylinders and standard hydraulic solenoid valves. The necessary positional feedback is provided by 8-bit resolution LVDT's. A positioning repeatability of +/- 0.00045" was demonstrated.

8. The cylinder rod of a standard hydraulic valves can successfully be used as a die location pin provided that the cylinder body is electrically insulated from the body of the die. Ceramic material is suitable for this purpose.

9. Gauging of the blade aerofoil is possible within the die with subsequence off-setting of the datum position of the blade within the die, thus enabling the encapsulation process to compensate for component inaccuracy. A gauging repeatability of +/- 0.00015" was demonstrated. Such a facility could lead to the automation of the encapsulation of components other than turbine blades, such as nozzle guide vanes.

10. The location pins may be completely withdrawn from the die cavity thus the angular positions of the pins are no longer constrained by the needs of encapsulation block extraction from the die cavity. This could lead to a simplification of die design.

11. Setting of the datum position of each die location pin may be carried out using software thus simplifying the existing die pin setting procedure, with consequent cost savings in both die manufacture and maintenance (this could be further enhanced with feedback from the post-encapsulation gauge).

12. Further work is needed to understand more fully the behaviour of the hydraulic control valves in order to improve the pin positioning accuracy.

13. The control software for a ten pin encapsulation die has been demonstrated, including blade positioning.

Appendix I

Single Die-Pin Rig Software

Absolute Pin Positioning

Program to move the die pin from
a specified start position to a
target position.

```
1.00=program HOLDPIN(input,output);
2.00=
3.00={*************************************************************************}
4.00={*                                                                       *}
5.00={* Program to drive the die pin to a specified start position           *}
6.00={* and to hold it in position until the RETURN key is pressed.           *}
7.00={* Then to drive the pin to the specified target position and            *}
8.00={* to hold it in position until 700 position readings have               *}
9.00={* been stored.                                                          *}
0.00={*                                                                       *}
1.00={*************************************************************************}
2.00=
3.00={$IGLOBVARS}                                  { include global variables   }
4.00={$ISUBS_EXT}                                  { declare procs from PIN_SUBS}
5.00={$IUEXTS}                                     { declare procs from library }
6.00=
7.00={*************************************************************************}
8.00={*                                                                       *}
9.00={*  set-up the screen to facilitate input of the user variables          *}
0.00={*                                                                       *}
1.00={*************************************************************************}
2.00=
3.00=procedure SET_SCREEN;
4.00=
5.00=begin
6.00=
7.00=    CS:=£$1A; CH:=£$1E; CF:=£$0C; BS:=£08;
8.00=
9.00=    write(CS);
0.00=
1.00=    POSITION_CURSOR(£0,£0);   write('    START PIN POSITION:');
2.00=    POSITION_CURSOR(£2,£0);   write('   TARGET PIN POSITION:');
3.00=
4.00=    POSITION_CURSOR(£6,£0);   write('    POSITION BANDWIDTH:');
5.00=    POSITION_CURSOR(£8,£0);   write('    POSITION TOLERANCE:');
6.00=    POSITION_CURSOR(£10,£0);  write('    VALVE  PULSE  TIME:');
7.00=    POSITION_CURSOR(£12,£0);  write('    DATA SAMPLING TIME:');
8.00=
9.00=    POSITION_CURSOR(£16,£0);  write(' CURRENT PIN POSITION:');
0.00=
1.00=end;
2.00=
3.00={*************************************************************************}
4.00={*                                                                       *}
5.00={* enter values for start/target positions, position bandwidth,          *}
6.00={* data storage sampling time and valve pulse time                       *}
7.00={*                                                                       *}
8.00={*************************************************************************}
9.00=
0.00=procedure GET_VARIABLES;
1.00=
2.00=begin
3.00=    POSITION_CURSOR(£0,£24); WRITE_LINE(£$20,5);
4.00=    POSITION_CURSOR(£0,£24);
5.00=    START_POS:=integer(FETCH_NUMBER);
6.00=
7.00=    POSITION_CURSOR(£2,£24); WRITE_LINE(£$20,5);
8.00=    POSITION_CURSOR(£2,£24);
9.00=    TARGET:=integer(FETCH_NUMBER);
```

```
60.00=
61.00=    POSITION_CURSOR(£6,£24); WRITE_LINE(£$20,5);
62.00=    POSITION_CURSOR(£6,£24);
63.00=    BAND_WIDTH:=integer(FETCH_NUMBER);
64.00=
65.00=    POSITION_CURSOR(£8,£24); WRITE_LINE(£$20,5);
66.00=    POSITION_CURSOR(£8,£24);
67.00=    TOLERANCE:=integer(FETCH_NUMBER);
68.00=
69.00=    POSITION_CURSOR(£10,£24); WRITE_LINE(£$20,5);
70.00=    POSITION_CURSOR(£10,£24);
71.00=    PULSE_TIME:=integer(FETCH_NUMBER);
72.00=
73.00=    POSITION_CURSOR(£12,£24); WRITE_LINE(£$20,5);
74.00=    POSITION_CURSOR(£12,£24);
75.00=    S_TIME:=integer(FETCH_NUMBER);
76.00=
77.00=    POSITION_CURSOR(£23,£0);
78.00=    write('                          ');       { clear previous message   }
79.00=
80.00=end;
81.00=
82.00={*************************************************************************}
83.00={*                                                                       *}
84.00={* clear the data storage array at $4000 and fill the target             *}
85.00={* array at $4800 and the two band-width arrays at $5000 & $5800         *}
86.00={*                                                                       *}
87.00={*************************************************************************}
88.00=
89.00=procedure CLEAR_AND_SET_DATA_ARRAYS;
90.00=
91.00=begin
92.00=    COUNTER:=0;                                  { array position marker     }
93.00=
94.00=    repeat
95.00=       POS [COUNTER] :=0;                        { clear pin position array  }
96.00=       FWD_VALVE [COUNTER] :=1;                  { clear valve position array}
97.00=       REV_VALVE [COUNTER] :=1;                  { clear valve position array}
98.00=
99.00=       CHECK:=integer(COUNTER/20);
100.00=
101.00=      if COUNTER-20*CHECK <>0
102.00=      then
103.00=         TARGET_POS [COUNTER] :=TARGET                    {only for plot}
104.00=      else
105.00=         TARGET_POS [COUNTER] :=0;      { chain-dot line for plot  }
106.00=
107.00=      CHECK:=integer(COUNTER/10);
108.00=
109.00=      if COUNTER-10*CHECK <>0
110.00=      then begin
111.00=         UPPER_LIMIT [COUNTER] :=TARGET+TOLERANCE;  {only for plot}
112.00=         LOWER_LIMIT [COUNTER] :=TARGET-TOLERANCE;  {only for plot}
113.00=      end
114.00=      else begin
115.00=         UPPER_LIMIT [COUNTER] :=0;     { chain-dot line for plot  }
116.00=         LOWER_LIMIT [COUNTER] :=0;     { chain-dot line for plot  }
117.00=      end;
118.00=
119.00=      COUNTER:=COUNTER+1;
```

```
120.00=   until COUNTER=700;                    { data array has 700 elements }
121.00=
122.00=end;
123.00=
124.00={*************************************************************}
125.00={*                                                           *}
126.00={*  If the sign of the ERROR has changed the CHANGE_FLAG is set. *}
127.00={*                                                           *}
128.00={*************************************************************}
129.00=
130.00=procedure TEST_FOR_DIRECTION_CHANGE;
131.00=
132.00=begin
133.00=   ERROR:=NEW_POS-ord(LVDT1);
134.00=   CHANGE_FLAG:=£0;
135.00=
136.00=   if (ERROR<0) and (DIR_FLAG=£1) then CHANGE_FLAG:=£1;
137.00=   if (ERROR>0) and (DIR_FLAG=£0) then CHANGE_FLAG:=£1;
138.00=
139.00=end;              { DIR_FLAG=£1 if the ERROR was initially positive }
140.00=
141.00={*************************************************************}
142.00={*                                                           *}
143.00={* pulse one valve for the duration of PULSE_LENGTH (msecs)  *}
144.00={*                                                           *}
145.00={*************************************************************}
146.00=
147.00=procedure PULSE_VALVE (VALVE:hex; PULSE_LENGTH:integer; FLAG:byte);
148.00=
149.00=begin
150.00=   COUNT:=0;
151.00=   WBYTE(((VALVE eor $1)+$4),£0);  { ensure unwanted valve is off }
152.00=   SET_TIMER(4,5);                 { initialise timer, 5  msec int }
153.00=
154.00=   while COUNT < integer(PULSE_LENGTH/5)  do
155.00=                                    { loop for pulse duration   }
156.00=   begin
157.00=      WBYTE((VALVE+$4),£1);                        { open valve  }
158.00=
159.00=      if R_FLAG=£1 then                    { record data        }
160.00=      begin
161.00=
162.00=      if VALVE=$0 then begin               {rev valve selected }
163.00=                  REV_VALVE [COUNTER] :=8;    {store valve open   }
164.00=                  FWD_VALVE [COUNTER] :=1;    {store valve closed }
165.00=                  end
166.00=                  else begin               {fwd valve selected }
167.00=                  FWD_VALVE [COUNTER] :=8;    {store valve open   }
168.00=                  REV_VALVE [COUNTER] :=1;    {store valve closed }
169.00=                  end;
170.00=      end;
171.00=
172.00=      if TIMEOUT_4 then                  { pulse timer interrupt  }
173.00=         begin
174.00=            COUNT:=COUNT+1;
175.00=            SET_TIMER(4,5);                        { reset timer }
176.00=         end;
177.00=
178.00=      if TIMEOUT_3 then          { timer interrupt for data store }
179.00=         begin
```

```
180.00=             if R_FLAG=£1 then POS [COUNTER] :=ord(LVDT1);
181.00=             COUNTER:=COUNTER+1;
182.00=             SET_TIMER(3,S_TIME);          { reset timer              }
183.00=           end;
184.00=
185.00=       if FLAG=£1 then
186.00=         begin
187.00=           TEST_FOR_DIRECTION_CHANGE;
188.00=           if CHANGE_FLAG=£1 then                          { quit loop }
189.00=                               COUNT:=integer(PULSE_LENGTH/5);
190.00=         end;
191.00=    end;                                   { end of pulse loop        }
192.00=
193.00=    WBYTE((VALVE + $4),£0);                { close valve              }
194.00=
195.00=    COUNT:=0; SET_TIMER(4,5);            { reset for valve off pulse }
196.00=
197.00=    if (FLAG=£1) and (CHANGE_FLAG=£1) then      { error sign changed}
198.00=       COUNT:=integer((2*PULSE_LENGTH)/5);      { jump around loop  }
199.00=
200.00=    while COUNT < integer((2*PULSE_LENGTH)/5)  do
201.00=                                          { loop for 2*pulse duration }
202.00=    begin
203.00=       if TIMEOUT_4 then                    { pulse timer interrupt  }
204.00=          begin
205.00=             COUNT:=COUNT+1;
206.00=             SET_TIMER(4,5);                          { reset timer }
207.00=          end;
208.00=
209.00=       if TIMEOUT_3 then           { timer interrupt for data store }
210.00=          begin
211.00=             if R_FLAG=£1 then POS [COUNTER] :=ord(LVDT1);
212.00=             COUNTER:=COUNTER+1;
213.00=             SET_TIMER(3,S_TIME);          { reset timer              }
214.00=          end;
215.00=    end;                                   { end of pulse loop        }
216.00=
217.00=end;
218.00=
219.00={*************************************************************************}
220.00={*                                                                       *}
221.00={* Pulse the pin to the required position ( NEW_POS ) by pulsing          *}
222.00={* the valves for a set time ( PULSE_LENGTH ).  The valve is              *}
223.00={* settled by oscillating using short pulses, once in position.           *}
224.00={*                                                                       *}
225.00={*************************************************************************}
226.00=
227.00=procedure MOVE_PIN_SLOW;
228.00=
229.00=begin
230.00=
231.00=    ERROR:=NEW_POS-ord(LVDT1);
232.00=
233.00=    if TIMEOUT_3 then           { timer interrupt for data store     }
234.00=       begin
235.00=          if R_FLAG=£1 then POS [COUNTER] :=ord(LVDT1);
236.00=          COUNTER:=COUNTER+1;
237.00=          SET_TIMER(3,S_TIME);               { reset timer           }
238.00=       end;
239.00=
```

```
240.00=    if R_FLAG=£1 then                          { record data          }
241.00=    begin
242.00=       FWD_VALVE [COUNTER] :=1;                { store valve is closed}
243.00=       REV_VALVE [COUNTER] :=1;                { store valve is closed}
244.00=    end;
245.00=
246.00=    if ERROR>0 then DIR_FLAG:=£1               { set flag for         }
247.00=               else DIR_FLAG:=£0;              { dir'n of movement    }
248.00=
249.00=    while abs(ERROR) > 0 do              { pulse the pin until either  }
250.00=    begin                                { the ERROR=0 or the sign of  }
251.00=                                         { the ERROR changes.          }
252.00=       ERROR:=NEW_POS-ord(LVDT1);
253.00=       if COUNTER>699 then ERROR:=0;
254.00=
255.00=       TEST_FOR_DIRECTION_CHANGE;
256.00=       if CHANGE_FLAG=£1 then ERROR:=0;          { quit loop          }
257.00=
258.00=       if ERROR > 0 then begin                   { move pin forward }
259.00=                         PULSE_VALVE($1,PULSE_TIME,£1);
260.00=                         PULSE_FLAG:=$1;         { valve has pulsed }
261.00=                         end
262.00=       else
263.00=       if ERROR < 0 then begin                   { move pin backward}
264.00=                         PULSE_VALVE($0,PULSE_TIME,£1);
265.00=                         PULSE_FLAG:=$0;         { valve has pulsed }
266.00=                         end;
267.00=
268.00=    end;                                 { end of pin pulsing loop     }
269.00=
270.00=    if PULSE_FLAG<>$2 then               { if valve has been pulsed to }
271.00=                                         { bring the pin into the tol. }
272.00=                                         { then oscillate the valve in }
273.00=                                         { order to stabilise the pin. }
274.00=    begin
275.00=       CHECK:=0; while CHECK<4  do             { oscillate the valve }
276.00=                    begin
277.00=                       PULSE_VALVE((PULSE_FLAG eor $1),10,£0);
278.00=                       if CHECK=0 then
279.00=                          PULSE_VALVE((PULSE_FLAG eor $1),10,£0);
280.00=                          { pulse twice in opposite direction      }
281.00=                       PULSE_VALVE( PULSE_FLAG,10,£0);
282.00=                       CHECK:=CHECK+1;
283.00=                       ERROR:=NEW_POS-ord(LVDT1);
284.00=                       if (abs(ERROR)>TOLERANCE) or (COUNTER>699)
285.00=                          then CHECK:=4;
286.00=                                  { quit loop if out of tolerance  }
287.00=                    end;
288.00=
289.00=       PULSE_FLAG:=$2;                           { clear pulse flag }
290.00=    end;
291.00=
292.00=end;
293.00=
294.00={*************************************************************}
295.00={*                                                           *}
296.00={*  Hold the pin within the TOL. A one pass loop.            *}
297.00={*                                                           *}
298.00={*************************************************************}
299.00=
```

```
300.00=procedure HOLD_PIN;
301.00=
302.00=begin
303.00=
304.00=    ERROR:=NEW_POS-ord(LVDT1);
305.00=
306.00=       if abs(ERROR)>TOLERANCE then MOVE_PIN_SLOW;
307.00=
308.00=       if TIMEOUT_3 then        { timer interrupt for data store    }
309.00=          begin
310.00=             if R_FLAG=£1 then POS [COUNTER] :=ord(LVDT1);
311.00=             COUNTER:=COUNTER+1;
312.00=             SET_TIMER(3,S_TIME);            { reset timer          }
313.00=          end;
314.00=
315.00=end;
316.00=
317.00={*************************************************************************}
318.00={*                                                                       *}
319.00={* open the valve to move the pin in the direction of                    *}
320.00={* diminishing ERROR until within the BAND-WIDTH. Then use               *}
321.00={* MOVE_PIN_SLOW to pulse the pin to the edge of the TOLERANCE           *}
322.00={* band and stabilise the valve, then MOVE_PIN_SLOW to the target        *}
323.00={*                                                                       *}
324.00={*************************************************************************}
325.00=
326.00=procedure MOVE_PIN_FAST;
327.00=
328.00=begin
329.00=
330.00=    POSITION:=ord(LVDT1);
331.00=    ERROR:=NEW_POS-POSITION;
332.00=    PULSE_FLAG:=$2;                                  { clear pulse flag  }
333.00=
334.00=    while abs(ERROR) > BAND_WIDTH do  {loop until within BAND_WIDTH}
335.00=    begin
336.00=       if ERROR>0 THEN
337.00=          begin
338.00=             WBYTE($4,£0); WBYTE($5,£1);          { drive pin forward }
339.00=
340.00=             if R_FLAG=£1 then                    { record data       }
341.00=             begin
342.00=                FWD_VALVE [COUNTER] :=8;          {store valve is open}
343.00=                REV_VALVE [COUNTER] :=1;        {store valve is closed}
344.00=             end;
345.00=
346.00=          end
347.00=          else begin
348.00=             WBYTE($4,£1); WBYTE($5,£0);          { drive pin backward}
349.00=
350.00=             if R_FLAG=£1 then                    { record data       }
351.00=             begin
352.00=                REV_VALVE [COUNTER] :=8;          {store valve is open}
353.00=                FWD_VALVE [COUNTER] :=1;        {store valve is closed}
354.00=             end;
355.00=
356.00=          end;
357.00=
358.00=       POSITION:=ord(LVDT1);
359.00=       ERROR:=NEW_POS-POSITION;
```

```
360.00=
361.00=       if TIMEOUT_3 then
362.00=          begin
363.00=             if R_FLAG=£1 then POS [COUNTER] :=POSITION;
364.00=             COUNTER:=COUNTER+1;
365.00=             SET_TIMER(3,S_TIME);
366.00=          end;
367.00=
368.00=        if COUNTER>699 then ERROR:=BAND_WIDTH;          { exit loop }
369.00=
370.00=    end;
371.00=
372.00=    TEMP:=NEW_POS;                          { save orig value of dest'n   }
373.00=
374.00=    if ERROR>0 then
375.00=    NEW_POS:=NEW_POS-TOLERANCE              { pulse to edge of TOLERANCE }
376.00=    else if ERROR<0 then                    { band & stabilise valve     }
377.00=    NEW_POS:=NEW_POS+TOLERANCE;
378.00=
379.00=    MOVE_PIN_SLOW;                          { pulse to edge of tolerance }
380.00=
381.00=    NEW_POS:=TEMP;                          { restore orig value of dest }
382.00=    ERROR:=NEW_POS-ord(LVDT1);
383.00=    MOVE_PIN_SLOW;
384.00=
385.00=end;
386.00=
387.00={*************************************************************************}
388.00={*                        MAIN PROCEDURE                                 *}
389.00={*************************************************************************}
390.00=
391.00=begin
392.00=    WBYTE($4,£0); WBYTE($5,£0);      { close both hydraulic valves  }
393.00=    WBYTE($8,£1);                    { open main air valve          }
394.00=
395.00=    VECTOR:=$B000;                       { reset interrupt vector   }
396.00=
397.00=    SET_SCREEN;
398.00=    GET_VARIABLES;
399.00=    CLEAR_AND_SET_DATA_ARRAYS;
400.00=    R_FLAG:=£0;                      { data is not to be recorded   }
401.00=
402.00=    POSITION:=ord(LVDT1);
403.00=    POSITION_CURSOR(£16,£24);
404.00=    write(POSITION);
405.00=
406.00=    COUNTER:=0;                          { initialise array marker }
407.00=    SET_TIMER(3,S_TIME);                 { initialise data timer   }
408.00=
409.00=    NEW_POS:=START_POS;
410.00=    MOVE_PIN_FAST;
411.00=    POSITION_CURSOR(£23,£0);
412.00=    write('HOLDING AT START POSITION');
413.00=
414.00=    POSITION:=ord(LVDT1);
415.00=    POSITION_CURSOR(£16,£24);
416.00=    write(POSITION);
417.00=
418.00=    WBYTE($E785,£1);                     { enable keyboard scan    }
419.00=    CHAC:=SCANKEY;                       { initialise CHAC         }
```

```
420.00=
421.00=    while CHAC <> £$0D do                    { hold pin at START_POS   }
422.00=                                             {  until RETURN pressed   }
423.00=    begin
424.00=       COUNTER:=0;                           { reset data store marker }
425.00=                                             { - storage not required  }
426.00=       HOLD_PIN;
427.00=       CHAC:=SCANKEY;                        { check if key pressed    }
428.00=    end;
429.00=
430.00=    COUNTER:=0;                              { reset data store marker }
431.00=    R_FLAG:=£1;                              { - storage NOW required  }
432.00=    SET_TIMER(3,S_TIME);                     { initialise data timer   }
433.00=
434.00=    POSITION_CURSOR(£23,£0);
435.00=    write('DRIVING PIN                ');
436.00=
437.00=    NEW_POS:=TARGET;
438.00=    MOVE_PIN_FAST;
439.00=
440.00=    POSITION:=ord(LVDT1);
441.00=    POSITION_CURSOR(£16,£24);
442.00=    write(POSITION);
443.00=
444.00=    while COUNTER<700 do                     { data array 700 elements }
445.00=       begin
446.00=          HOLD_PIN;
447.00=       end;
448.00=
449.00=    POSITION_CURSOR(£23,£0);
450.00=    write('TIMED OUT                  ');
451.00=
452.00=    POSITION:=ord(LVDT1);
453.00=    POSITION_CURSOR(£16,£24);
454.00=    write(POSITION);
455.00=
456.00=    WBYTE($8,£0);                            { close main air valve    }
457.00=
458.00=end.
```

```
1.00={* GLOBVARS                                                          *}
2.00=
3.00=
4.00={*************************************************************************}
5.00={*                                                                       *}
6.00={*    SINGLE DIE-PIN TEST RIG          GLOBAL VARIABLES     ARW        *}
7.00={*                                                                       *}
8.00={*************************************************************************}
9.00=
10.00=
11.00=const
12.00=    CR    = £$0D;                  { carriage return }
13.00=    LF    = £$0A;                  { line feed       }
14.00=    EQ    = £$3D;                  { equals =        }
15.00=    ES    = £$1B;                  { escape code     }
16.00=
17.00=    VIA3 = $EE00;
18.00=    VIA4 = $EE10;                  { via's for timer interrupts          }
19.00=
20.00=var
21.00=    CS,                            { clear  screen  character            }
22.00=    CH,                            { cursor home    character            }
23.00=    CF,                            { cursor forward character            }
24.00=    BS,                            { back space character                }
25.00=    CHAC,                          { character that is input/output      }
26.00=
27.00=    ERROR_FLAG,                    { used to check incoming character    }
28.00=    DP_FLAG,                       { flag for decimal point input        }
29.00=    DIR_FLAG,                      { =1 if pin error is positive         }
30.00=    CHANGE_FLAG,                   { =1 if pin error has changed sign    }
31.00=    R_FLAG          :byte;         { flag=1 if data is to be recorded    }
32.00=
33.00=    NEW_POS         : integer;     { required pin position               }
34.00=
35.00=    PRINTER_FLAG    : hex at $CD10;
36.00=    VECTOR          : hex at $DFCE;      {  interrupt vector              }
37.00=
38.00=    LVDT1           : byte at $EB00;     { LVDT reading pin 1             }
39.00=    LVDT2           : byte at $EB01;     { LVDT reading pin2              }
40.00=
41.00=    CONTACT         : byte at $000C;     { =0 when pin is shorted         }
42.00=
43.00=    FORWARD         : byte at $0000;     { valve move pin forward         }
44.00=    REVERSE         : byte at $0001;     { valve move pin backwards       }
45.00=
46.00=    PULSE_FLAG      : hex;               { =$2 if valve not pulsed        }
47.00=
48.00=    TERMINAL        : hex;       { output to terminal not printer        }
49.00=
50.00=    BAND_WIDTH      : integer at $E700; { tol on target position         }
51.00=    TARGET          : integer at $E710; { target pos'n of die pin        }
52.00=    S_TIME          : integer at $E720; { sampling time of timer £3      }
53.00=    START_POS       : integer at $E725; { start position of pin          }
54.00=    PULSE_TIME      : integer at $E730; { pulse-width for valve          }
55.00=    TOLERANCE       : integer at $E735; { positional tol on target       }
56.00=
57.00=    BLADE_TOLERANCE : integer at $E740; { calculated from LVDT's         }
58.00=
59.00=    POSITION,                      { LVDT position from the A/D          }
```

```
60.00=    ERROR              : integer;  { error in pin position              }
61.00=
62.00=    FWD_FLAG,                      { pin moving forward                 }
63.00=    REV_FLAG           : byte;     { pin moving backward                }
64.00=
65.00=    TIMEOUT_3          : boolean at $E791;  { timer interrupt flag      }
66.00=    TIMEOUT_4          : boolean at $E792;  { timer interrupt flag      }
67.00=
68.00=    TEMP,
69.00=    CHECK,
70.00=    COUNT,
71.00=    COUNTER            : integer;  { general counters                   }
72.00=
73.00=    POS          :array [0..700] of integer at $4000;
74.00=                                   {    pin position array for printer }
75.00=    TARGET_POS :array [0..700] of integer at $4800;
76.00=                                   { target position array for printer }
77.00=    UPPER_LIMIT:array [0..700] of integer at $5000;
78.00=                                   { pin pos'n +ve tolerance for plot  }
79.00=    LOWER_LIMIT:array [0..700] of integer at $5800;
80.00=                                   { pin pos'n ive tolerance for plot  }
81.00=    FWD_VALVE  :array [0..700] of integer at $6000;
82.00=                                   { whether fwd valve open or closed  }
83.00=    REV_VALVE  :array [0..700] of integer at $6800;
84.00=                                   { whether rev valve open or closed  }
85.00=
86.00=    LVDT1_CAL,
87.00=    LVDT2_CAL          : integer;  { LVDT reading - nom. blade on datum}
88.00=
89.00=    DAY,
90.00=    MONTH,
91.00=    YEAR,                                  { date for table print    }
92.00=    TABLE_NO           : integer;          { print table identifier  }
93.00=
94.00=    MESS               : string [80];   { general string for printing  }
95.00=    SSTR               : string [80];   { temp string for concat'ing   }
96.00=
97.00=    P_START,
98.00=    P_CONTACT,
99.00=    P_LVDT1,
100.00=   P_LVDT2,
101.00=   P_TOL,
102.00=   P_TARGET,
103.00=   P_FINISH           : integer;       { values for printing out       }
```

```
1.00={* SUBS_EXT                                                          *}
2.00=
3.00=
4.00={*************************************************************************}
5.00={*                                                                       *}
6.00={*     SINGLE DIE-PIN TEST RIG    EXTERNAL PROCS FROM PIN_SUBS        *}
7.00={*                                                                       *}
8.00={*************************************************************************}
9.00=
10.00=
11.00=procedure CHECK_VALUE; external;
12.00=
13.00=function  FIX(STR:string):real; external;
14.00=
15.00=function  FETCH_NUMBER:real; external;
16.00=
17.00=procedure WRITE_LINE(CHAC:string;COUNTER:integer); external;
18.00=
19.00=procedure POSITION_CURSOR(ROW,COLUMN:byte); external;
20.00=
21.00=procedure SEND_OUTPUT_TO_PRINTER; external;
22.00=
23.00=procedure RESTORE_OUTPUT_TO_SCREEN; external;
24.00=
25.00=procedure PRINT(MESSAGE:string); external;
26.00=
27.00=procedure CLEAR_LINE(NUMBER:integer); external;
28.00=
29.00=procedure SET_TIMER (via : 3..4 ; millisec : integer); external;
```

```
1.00=module PIN_SUBS(input,output);
2.00=
3.00={*************************************************************************}
4.00={*                                                                       *}
5.00={*    SINGLE DIE-PIN TEST RIG      SUBROUTINE MODULE                     *}
6.00={*                                                                       *}
7.00={*************************************************************************}
8.00=
9.00={$IGLOBVARS}                        { include global variables            }
10.00={$IUEXTS}
11.00=
12.00=procedure TIMVIA(base : hex); external;
13.00=
14.00={*************************************************************************}
15.00={**                                                                     **}
16.00={**     Originated 9/11/84 by M. Taylor                                 **}
17.00={**                                                                     **}
18.00={**      procedure set_timer (via : 2..4;  millisec : integer )         **}
19.00={**                                                                     **}
20.00={**      Initiates a timer to generate an interrupt after the           **}
21.00={**      specified number of millisecs using the specified VIA.         **}
22.00={**      This routine sets only those VIA attributes required           **}
23.00={**      for timing.  All other register settings are preserved         **}
24.00={**      The timeout of timer 2 causes the VIA to generate IRQ          **}
25.00={**      and IRQ service routine must then poll the VIAs (and           **}
26.00={**      other sources) to determine the source of the IRQ.             **}
27.00={**      If caused by a timeout, it sets the appropriate global         **}
28.00={**      timeout flag.  These flags are set false by set_timer          **}
29.00={**      on setting up the timeout, so may be tested at leisure         **}
30.00={**      by other routines which may have a timeout currently           **}
31.00={**      running.                                                       **}
32.00={**                                                                     **}
33.00={*************************************************************************}
34.00=
35.00=procedure SET_TIMER (via : 3..4;  millisec : integer ); entry;
36.00=
37.00=var
38.00=  base : hex;
39.00=
40.00=begin
41.00=
42.00=  case via of
43.00=    3 : begin
44.00=          timeout_3 := false;
45.00=          base := VIA3
46.00=        end;
47.00=    4 : begin
48.00=          timeout_4 := false;
49.00=          base := VIA4
50.00=        end;
51.00=  end; {case}
52.00=  TIMVIA ( base );     { Transparently sets up via at 'base' as timer
53.00=  IRQ (£1);    { Enable IRQs }
54.00=  WBYTE ( base+$8, chr (millisec mod 256) );  { write low byte }
55.00=  WBYTE ( base+$9, chr (millisec div 256) );  { write hi  byte }
56.00=
57.00=  {** Writing the high byte starts the timeout **}
58.00=
59.00=end; {set_timer}
```

```
60.00=
61.00={*************************************************************}
62.00={*       CHECK IF A CHARACTER (CHAR) IS IN THE RANGE 0-9        *}
63.00={*************************************************************}
64.00=
65.00=procedure CHECK_VALUE; entry;  { if CHAC is in the range 0-9 then  }
66.00=                              { ERROR_FLAG is set to 0             }
67.00=begin                         { DP_FLAG is set & remains at 1 if   }
68.00=                              { a decimal point is found           }
69.00=   ERROR_FLAG:=£1;
70.00=
71.00=      if CHAC=£$2E then begin
72.00=                           if DP_FLAG=£0 then ERROR_FLAG:=£0
73.00=                                         else ERROR_FLAG:=£1;
74.00=                           DP_FLAG:=£1;
75.00=                        end
76.00=
77.00=                     else begin
78.00=                           if CHAC>£$2F then
79.00=                           if CHAC<£$3A then ERROR_FLAG:=£0;
80.00=                        end;
81.00=end;
82.00=
83.00={*************************************************************}
84.00={*            CONVERT A STRING TO A REAL NUMBER                *}
85.00={*************************************************************}
86.00=
87.00=function FIX(STR:string):real; entry;
88.00=
89.00=var
90.00=   LEN,                        { length of incoming string          }
91.00=   CHAC,                       { single byte of string              }
92.00=   POWER,                      { exponent of base 10                }
93.00=   INC      : byte;            { loop counter                       }
94.00=   TEMPR,                      { real temporary storage variable    }
95.00=   DIVISOR,                    { divisor to be used if dec pt found }
96.00=   NUMBER   : real;            { real number being generated        }
97.00=   TEMPI    : integer;         { integer temporary storage variable }
98.00=
99.00=begin
100.00=   LEN:=STR [0];              { read dynamic length of string      }
101.00=   NUMBER:=0;                 { initialise output variable         }
102.00=   DIVISOR:=0;                { initialise possible divisor        }
103.00=   POWER:=LEN-£1;             { initialise power of base 10        }
104.00=
105.00=   for INC:=£1 to LEN do          { process full length of string  }
106.00=
107.00=      begin
108.00=         CHAC:=STR [INC];
109.00=         TEMPR:=10**real(POWER);     { multiplier for power of 10  }
110.00=         TEMPR:=real(round(TEMPR)); { integerise TEMPR             }
111.00=
112.00=         if CHAC=£$2E then DIVISOR:=TEMPR*10   { decimal pt found }
113.00=
114.00=                      else begin
115.00=                              TEMPI:=integer(CHAC-£$30);{ ascii ! }
116.00=                              NUMBER:=NUMBER+TEMPI*TEMPR;
117.00=                              POWER:=POWER-£1;
118.00=                           end;
119.00=      end;
```

```
120.00=
121.00=    if DIVISOR>0 then NUMBER:=NUMBER/DIVISOR;      { insert dec pt   }
122.00=
123.00=    FIX:=NUMBER;                                   { assign real number and return }
124.00=
125.00=end;
126.00=
127.00={*************************************************************************}
128.00={*       PRINT A REPEATED STRING OF A SINGLE CHARACTER                     *}
129.00={*************************************************************************}
130.00=
131.00=procedure WRITE_LINE(CHAC:string;COUNTER:integer); entry;
132.00=
133.00=var
134.00=    INC:integer;                                   { loop counter          }
135.00=
136.00=begin
137.00=    INC:=0;                                        { initialise counter    }
138.00=       repeat
139.00=          write(CHAC);
140.00=          INC:=INC+1;
141.00=       until  INC=COUNTER;
142.00=end;
143.00={*************************************************************************}
144.00={*         INPUT A NUMERICAL VALUE FROM THE KEYBOARD                       *}
145.00={*************************************************************************}
146.00=
147.00=function FETCH_NUMBER : real; entry;
148.00=
149.00=var
150.00=    NUMBER : string;                               { buffer for incoming number  }
151.00=    INC,
152.00=    LEN    : byte;                                 { length of above buffer      }
153.00=
154.00=begin
155.00=
156.00=    NUMBER [0]:=£0;                                { initialise string length    }
157.00=    ERROR_FLAG:=£0;                                { for procedure CHECK_VALUE   }
158.00=    DP_FLAG:=£0;                                   { initialise flag for dec pt input }
159.00=    CHAC:=£0;                                      { initialise input variable   }
160.00=
161.00=    while CHAC<>CR do                              { get single character input  }
162.00=
163.00=       begin
164.00=
165.00=         CHAC:=INKEY;                             { get character from keyboard }
166.00=         CHECK_VALUE;                             { is CHAR in range 0 - 9 ?    }
167.00=
168.00=           if ERROR_FLAG=£0 then begin            { valid character  }
169.00=                                 write(CHAC);     { output to screen }
170.00=                                 LEN:=NUMBER [0];    { add char to   }
171.00=                                 LEN:=LEN + £1;      { number buffer }
172.00=                                 NUMBER [0]:=LEN;
173.00=                                 NUMBER [LEN]:=CHAC;
174.00=                                 end;
175.00=       end;
176.00=
177.00=    WRITE_LINE(£$20,5);
178.00=
179.00=    FETCH_NUMBER:=FIX(NUMBER);     { assign NUMBER and return          }
```

```
180.00=
181.00=end;
182.00=
183.00={*************************************************************************}
184.00={*      POSITION THE SCREEN CURSOR BY ROW AND COLUMN                      *}
185.00={*************************************************************************}
186.00=
187.00=procedure POSITION_CURSOR (ROW,COLUMN:byte); entry;
188.00=
189.00=var
190.00=    INC : byte;                          { counter                        }
191.00=
192.00=begin
193.00=    write(CH);                  { send cursor home                       }
194.00=
195.00=       if ROW<>£0 then begin
196.00=                       INC:=£0;
197.00=
198.00=                       repeat
199.00=                          writeln; INC:=INC+£1;
200.00=                       until INC=ROW;
201.00=
202.00=                       end;
203.00=
204.00=       if COLUMN<>£0 then begin
205.00=                          INC:=£0;
206.00=
207.00=                          repeat
208.00=                             write(CF); INC:=INC+£1;
209.00=                          until INC=column;
210.00=
211.00=                          end;
212.00=end;
213.00=
214.00={*************************************************************************}
215.00={*     DIRECT OUTPUT TO PRINTER NOT TERMINAL SCREEN                       *}
216.00={*************************************************************************}
217.00=
218.00=procedure SEND_OUTPUT_TO_PRINTER; entry;
219.00=
220.00=begin
221.00=    TERMINAL:=PRINTER_FLAG; { store terminal output routine address }
222.00=    PRINTER_FLAG:=$CCE4;    { insert printer output routine address }
223.00=    ! JSR $CCCO             { initialise printer - FLEX routine     }
224.00=end;
225.00=
226.00={*************************************************************************}
227.00={*     RE-DIRECT OUTPUT TO TERMINAL SCREEN FROM PRINTER                   *}
228.00={*************************************************************************}
229.00=
230.00=procedure RESTORE_OUTPUT_TO_SCREEN; entry;
231.00=
232.00=begin
233.00=    PRINTER_FLAG:=TERMINAL;
234.00=end;
235.00=
236.00={*************************************************************************}
237.00={*         PRINT A TEXT MESSAGE TO THE PRINTER                           *}
238.00={*************************************************************************}
239.00=
```

```
240.00=procedure PRINT(MESSAGE:string); entry;
241.00=
242.00=begin
243.00=    SEND_OUTPUT_TO_PRINTER;
244.00=    writeln(MESSAGE);
245.00=    RESTORE_OUTPUT_TO_SCREEN;
246.00=end;
247.00=
248.00={*************************************************************************}
249.00={*              CLEAR A LINE OF SPECIFIED LENGTH                         *}
250.00={*            AND RETURN CURSOR TO START OF LINE                         *}
251.00={*************************************************************************}
252.00=
253.00=procedure CLEAR_LINE(NUMBER:integer); entry;
254.00=
255.00=var
256.00=    INC:integer;                    { loop counter                       }
257.00=
258.00=begin
259.00=    INC:=0;
260.00=       repeat
261.00=          write(' '); INC:=INC+1;
262.00=       until  INC=NUMBER;
263.00=
264.00=    INC:=0;
265.00=       repeat
266.00=          write(£08); INC:=INC+1;
267.00=       until  INC=NUMBER;
268.00=end;
269.00=
270.00=modend.
```

```
1.00={** Declarations of procedures and functions which are contained **}
2.00={** in the user library  (1.url.ro)                               **}
3.00=
4.00=
5.00=
6.00=procedure IN232;  external;
7.00=
8.00=procedure INTIO;  external;
9.00=
10.00=procedure WBYTE (address: hex; data: byte);  external;
11.00=
12.00=procedure WWORD (address: hex; data: hex);  external;
13.00=
14.00=procedure FIRQ ( endis : £0..£1 );  external;
15.00=
16.00=procedure IRQ ( endis : £0..£1 );  external;
17.00=
18.00=procedure DELAY (millisec: integer);  external;
19.00=
20.00=
21.00=function INKEY : char;  external;
22.00=
23.00=function RBYTE (address: hex) : byte;  external;
24.00=
25.00=function RWORD (address: hex) : hex;  external;
26.00=
27.00=function SCANKEY : char; external;
28.00=
```

Appendix II

Single Die-Pin Rig Software

Pin Position Calibration

A program to allow the die pin to be pulsed forwards and backwards using the keyboard terminal. The die-pin LVDT reading is displaced at all times.

```
1.00=program SHOWPIN (input,output);
2.00=
3.00=const
4.00=    VIA3 = $EE00;                          { VIA for timer interrupt      }
5.00=    VIA4 = $EE10;                          { VIA for timer interrupt      }
6.00=
7.00=var
8.00=    LVDT1          : byte at $EB00;
9.00=    LVDT2          : byte at $EB01;
10.00=   COUNT,
11.00=   POS            : integer;
12.00=   CHAC           : byte;
13.00=   TIMEOUT_3      : boolean at $E791;       { timer interrupt flag    }
14.00=   TIMEOUT_4      : boolean at $E792;       { timer interrupt flag    }
15.00=   VECTOR         : hex at $DFCE;           { interrupt vector        }
16.00=
17.00={$IUEXTS}
18.00=
19.00={*************************************************************************}
20.00=
21.00=procedure TIMVIA(BASE : hex); external;
22.00=
23.00={*************************************************************************}
24.00=
25.00=procedure SET_TIMER (VIA : 3..4; MILLISEC : integer);
26.00=
27.00=var
28.00=    BASE : hex;
29.00=
30.00=begin
31.00=
32.00=    case VIA of
33.00=      3 : begin
34.00=             TIMEOUT_3:=false;
35.00=             BASE :=VIA3
36.00=          end;
37.00=      4 : begin
38.00=             TIMEOUT_4:=false;
39.00=             BASE :=VIA4
40.00=          end;
41.00=    end; { case }
42.00=
43.00=    TIMVIA(BASE);                  { sets up VIA at BASE as a timer    }
44.00=    IRQ(£1);
45.00=    WBYTE(BASE+$8,chr(MILLISEC mod 256));      { write low byte       }
46.00=    WBYTE(BASE+$9,chr(MILLISEC div 256));      { write high byte      }
47.00=
48.00=end;
49.00=
50.00={*************************************************************************}
51.00={*                        Pulse the Valve                                *}
52.00={*************************************************************************}
53.00=
54.00=procedure PULSE_VALVE(VALVE:hex);
55.00=
56.00=begin
57.00=    COUNT:=0;
58.00=    WBYTE(((VALVE eor $1)*$A)+$1),£0); {ensure unwanted valve off }
59.00=    SET_TIMER(4,5);                    {initialise timer£4 to 5 msecs}
```

```
60.00=
61.00=    while COUNT<2 do                          { loop - pulse duration 10 ms }
62.00=    begin
63.00=       WBYTE(((VALVE*$A)+$1),£1);                          { open valve   }
64.00=
65.00=       if TIMEOUT_4 then                     { pulse timer interrupt  }
66.00=       begin
67.00=          COUNT:=COUNT+1;
68.00=          SET_TIMER(4,5);                                  { reset timer  }
69.00=       end;
70.00=    end;
71.00=
72.00=    WBYTE(((VALVE*$A)+$1),£0);                          { close valve  }
73.00=
74.00=    COUNT:=0;
75.00=    SET_TIMER(4,5);                       {initialise timer£4 to 5 msecs}
76.00=
77.00=    while COUNT<4 do                          { loop - pulse duration 20 ms }
78.00=    begin                                     { to ensure valve has closed  }
79.00=       if TIMEOUT_4 then                        { pulse timer interrupt  }
80.00=       begin
81.00=          COUNT:=COUNT+1;
82.00=          SET_TIMER(4,5);                                  { reset timer  }
83.00=       end;
84.00=    end;
85.00=
86.00=end;
87.00=
88.00={*************************************************************************}
89.00={*                          Main Procedure                                 *}
90.00={*************************************************************************}
91.00=
92.00=begin
93.00=    VECTOR:=$B000;                                { reset interrupt vector }
94.00=    WBYTE($E785,£1);                              { enable keyboard scan   }
95.00=
96.00=    write(£$1A);                                             { clear screen }
97.00=    writeln; writeln; writeln; writeln;
98.00=    writeln('Press F to move pin forward');
99.00=    writeln('Press R to move pin backward');
100.00=   writeln;
101.00=   writeln('Press RETURN to end program');
102.00=   write(£$1E);                                      {  home cursor       }
103.00=
104.00=   WBYTE($0,£1);                                    {  open main air valve   }
105.00=
106.00=   CHAC:=£0;                                                { initialise CHAC   }
107.00=
108.00=   while CHAC<>£$0D do                      { end loop when RETURN pressed}
109.00=   begin
110.00=      POS:=ord(LVDT1);
111.00=      writeln('LVDT1 : ',POS);
112.00=      POS:=ord(LVDT2);
113.00=      write('LVDT2 : ',POS,£$1E);
114.00=
115.00=      CHAC:=SCANKEY;                                {input keyboard character}
116.00=         if CHAC=£$46 then PULSE_VALVE($1);    { pulse forward      }
117.00=         if CHAC=£$52 then PULSE_VALVE($0);    { pulse backwards    }
118.00=   end;
119.00=
```

```
120.00=    WBYTE($0,£0);                          { close main air valve   }
121.00=    write(£$1A);                                    { clear screen }
122.00=
123.00=end.
```

Appendix III

Single Die-Pin Rig Software

Blade Gauging and Positioning

A program to move the die pin from a specified start position until the sample blade is contacted. The blade is then gauged and if necessary repositioned.

```
1.00=program CYCLE(input,output);
2.00=
3.00={*************************************************************}
4.00={*                                                           *}
5.00={* Program to cycle the single die-pin test rig thro' the    *}
6.00={* following sequence:                                       *}
7.00={* drive the pin to the specified start position and wait for *}
8.00={* a fixed time;                                             *}
9.00={* drive the pin towards the specified target position until *}
10.00={* electrical contact is made;                              *}
11.00={* read the two LVDT's and calculate the offset for the datum; *}
12.00={* drive the pin to the offset datum position.             *}
13.00={*                                                          *}
14.00={* This cycle is repeated until the return key is pressed. *}
15.00={*                                                          *}
16.00={*************************************************************}
17.00=
18.00={$IGLOBVARS}                          { include global variables   }
19.00={$ISUBS_EXT}                          { declare procs from PIN_SUBS}
20.00={$IUEXTS}                             { declare procs from library }
21.00=
22.00={*************************************************************}
23.00={*                                                           *}
24.00={*  set-up the screen to facilitate input of the user variables *}
25.00={*                                                           *}
26.00={*************************************************************}
27.00=
28.00=procedure SET_SCREEN;
29.00=
30.00=begin
31.00=    CS:=£$1A; CH:=£$1E; CF:=£$0C; BS:=£08;
32.00=
33.00=    write(CS);
34.00=    POSITION_CURSOR(£0,£0);  write('     DATE: ');
35.00=    DAY:=integer(FETCH_NUMBER);
36.00=    POSITION_CURSOR(£0,£12); write('.');
37.00=    MONTH:=integer(FETCH_NUMBER);
38.00=    POSITION_CURSOR(£0,£15); write('.');
39.00=    YEAR:=integer(FETCH_NUMBER);
40.00=
41.00=    POSITION_CURSOR(£2,£0);  write('TABLE_NO:');
42.00=    POSITION_CURSOR(£2,£11); TABLE_NO:=integer(FETCH_NUMBER);
43.00=
44.00=    write(CS);
45.00=    POSITION_CURSOR(£0,£0);  write('   START PIN POSITION:');
46.00=    POSITION_CURSOR(£2,£0);  write('  TARGET PIN POSITION:');
47.00=
48.00=    POSITION_CURSOR(£6,£0);  write('   POSITION BANDWIDTH:');
49.00=    POSITION_CURSOR(£8,£0);  write('   POSITION TOLERANCE:');
50.00=    POSITION_CURSOR(£10,£0); write('   VALVE  PULSE  TIME:');
51.00=    POSITION_CURSOR(£12,£0); write('   DATA SAMPLING TIME:');
52.00=
53.00=    POSITION_CURSOR(£16,£0); write('DATUM PIN CALIBRATION:');
54.00=    POSITION_CURSOR(£18,£0); write('SUPP. PIN CALIBRATION:');
55.00=
56.00=    POSITION_CURSOR(£22,£0); write(' CURRENT PIN POSITION:');
57.00=
58.00=end;
59.00=
```

```
60.00={*************************************************************}
61.00={*                                                           *}
62.00={* Enter values for start/target positions, position bandwidth *}
63.00={* and pin position tolerance, data storage sampling time and  *}
64.00={* valve pulse time, and the 2 LVDT calibration values.       *}
65.00={*                                                           *}
66.00={*************************************************************}
67.00=
68.00=procedure GET_VARIABLES;
69.00=
70.00=begin
71.00=    POSITION_CURSOR(£0,£24); WRITE_LINE(£$20,5);
72.00=    POSITION_CURSOR(£0,£24);
73.00=    START_POS:=integer(FETCH_NUMBER);
74.00=
75.00=    POSITION_CURSOR(£2,£24); WRITE_LINE(£$20,5);
76.00=    POSITION_CURSOR(£2,£24);
77.00=    TARGET:=integer(FETCH_NUMBER);
78.00=
79.00=    POSITION_CURSOR(£6,£24); WRITE_LINE(£$20,5);
80.00=    POSITION_CURSOR(£6,£24);
81.00=    BAND_WIDTH:=integer(FETCH_NUMBER);
82.00=
83.00=    POSITION_CURSOR(£8,£24); WRITE_LINE(£$20,5);
84.00=    POSITION_CURSOR(£8,£24);
85.00=    TOLERANCE:=integer(FETCH_NUMBER);
86.00=
87.00=    POSITION_CURSOR(£10,£24); WRITE_LINE(£$20,5);
88.00=    POSITION_CURSOR(£10,£24);
89.00=    PULSE_TIME:=integer(FETCH_NUMBER);
90.00=
91.00=    POSITION_CURSOR(£12,£24); WRITE_LINE(£$20,5);
92.00=    POSITION_CURSOR(£12,£24);
93.00=    S_TIME:=integer(FETCH_NUMBER);
94.00=
95.00=    POSITION_CURSOR(£16,£24); WRITE_LINE(£$20,5);
96.00=    POSITION_CURSOR(£16,£24);
97.00=    LVDT1_CAL:=integer(FETCH_NUMBER);
98.00=
99.00=    POSITION_CURSOR(£18,£24); WRITE_LINE(£$20,5);
100.00=   POSITION_CURSOR(£18,£24);
101.00=   LVDT2_CAL:=integer(FETCH_NUMBER);
102.00=
103.00=   POSITION_CURSOR(£23,£0);
104.00=   write('                    ');       { clear previous message   }
105.00=
106.00=end;
107.00=
108.00={*************************************************************}
109.00={*                                                           *}
110.00={* clear the data storage array at $4000 and fill the target  *}
111.00={* array at $4800 and the two band-width arrays at $5000 & $5800 *}
112.00={*                                                           *}
113.00={*************************************************************}
114.00=
115.00=procedure CLEAR_AND_SET_DATA_ARRAYS;
116.00=
117.00=begin
118.00=   COUNTER:=0;                                { array position marker    }
119.00=
```

```
120.00=   repeat
121.00=      POS [COUNTER] :=0;                    { clear pin position array  }
122.00=      FWD_VALVE [COUNTER] :=1;              { clear valve position array}
123.00=      REV_VALVE [COUNTER] :=1;              { clear valve position array}
124.00=
125.00=      CHECK:=integer(COUNTER/20);
126.00=
127.00=      if COUNTER-20*CHECK <>0
128.00=      then
129.00=         TARGET_POS [COUNTER] :=LVDT1_CAL              {only for plot}
130.00=      else
131.00=         TARGET_POS [COUNTER] :=0;      { chain-dot line for plot  }
132.00=
133.00=      COUNTER:=COUNTER+1;
134.00=   until COUNTER=700;                   { data array has 700 elements }
135.00=
136.00=end;
137.00=
138.00={*************************************************************}
139.00={*                                                           *}
140.00={* Check if there is an interrupt for data storage.          *}
141.00={*                                                           *}
142.00={*************************************************************}
143.00=
144.00=procedure CHECK_FOR_STORAGE_INTERRUPT;
145.00=
146.00=begin
147.00=   if TIMEOUT_3 then                 { timer interrupt for data store }
148.00=      begin
149.00=         if R_FLAG=£1 then POS [COUNTER] :=ord(LVDT1);
150.00=         COUNTER:=COUNTER+1;
151.00=         SET_TIMER(3,S_TIME);                { reset timer          }
152.00=      end;
153.00=end;
154.00=
155.00={*************************************************************}
156.00={*                                                           *}
157.00={* If the sign of the ERROR has changed the CHANGE_FLAG is set.  *}
158.00={*                                                           *}
159.00={*************************************************************}
160.00=
161.00=procedure TEST_FOR_DIRECTION_CHANGE;
162.00=
163.00=begin
164.00=   ERROR:=NEW_POS-ord(LVDT1);
165.00=   CHANGE_FLAG:=£0;                    { initialise the change flag  }
166.00=
167.00=   if (ERROR<0) and (DIR_FLAG=£1) then CHANGE_FLAG:=£1;
168.00=   if (ERROR>0) and (DIR_FLAG=£0) then CHANGE_FLAG:=£1;
169.00=
170.00=end;            { DIR_FLAG=£1 if the ERROR was initially positive }
171.00=
172.00={*************************************************************}
173.00={*                                                           *}
174.00={* pulse one valve for the duration of PULSE_LENGTH (msecs)   *}
175.00={*                                                           *}
176.00={*************************************************************}
177.00=
178.00=procedure PULSE_VALVE (VALVE:hex; PULSE_LENGTH:integer; FLAG:byte);
179.00=
```

```
180.00=begin
181.00=    COUNT:=0;
182.00=    WBYTE(((VALVE eor $1)+$4),£0);  { ensure unwanted valve is off }
183.00=    SET_TIMER(4,5);                 { initialise timer, 5  msec int }
184.00=
185.00=    while COUNT < PULSE_LENGTH/5  do   { loop for pulse duration   }
186.00=    begin
187.00=       WBYTE((VALVE+$4),£1);                                { open valve  }
188.00=
189.00=       if R_FLAG=£1 then                                 { record data       }
190.00=       begin
191.00=
192.00=       if VALVE=$0 then begin                            {rev valve selected }
193.00=                   REV_VALVE [COUNTER] :=8;     {store valve open   }
194.00=                   FWD_VALVE [COUNTER] :=1;     {store valve closed }
195.00=                   end
196.00=                   else begin                            {fwd valve selected }
197.00=                   FWD_VALVE [COUNTER] :=8;     {store valve open   }
198.00=                   REV_VALVE [COUNTER] :=1;     {store valve closed }
199.00=                   end;
200.00=       end;
201.00=
202.00=       if TIMEOUT_4 then                          { pulse timer interrupt  }
203.00=          begin
204.00=             COUNT:=COUNT+1;
205.00=             SET_TIMER(4,5);                                  { reset timer }
206.00=          end;
207.00=
208.00=       CHECK_FOR_STORAGE_INTERRUPT;
209.00=
210.00=       if FLAG=£1 then
211.00=       begin
212.00=          TEST_FOR_DIRECTION_CHANGE;
213.00=          if CHANGE_FLAG=£1 then                                { quit loop     }
214.00=                              COUNT:=integer(PULSE_LENGTH/5);
215.00=       end;
216.00=
217.00=    end;                                              { end of pulse loop       }
218.00=
219.00=    WBYTE((VALVE + $4),£0);                           { close valve             }
220.00=
221.00=    COUNT:=0; SET_TIMER(4,5);                  { reset for valve off pulse }
222.00=
223.00=    if (FLAG=£1) and (CHANGE_FLAG=£1) then     { ERROR sign changed }
224.00=       COUNT:=integer((2*PULSE_LENGTH)/5);     { jump around loop   }
225.00=
226.00=    while COUNT < (2*PULSE_LENGTH)/5  do {loop for 2*pulse duration}
227.00=    begin
228.00=       if TIMEOUT_4 then                          { pulse timer interrupt  }
229.00=          begin
230.00=             COUNT:=COUNT+1;
231.00=             SET_TIMER(4,5);                                  { reset timer }
232.00=          end;
233.00=
234.00=       CHECK_FOR_STORAGE_INTERRUPT;
235.00=
236.00=    end;                                              { end of pulse loop       }
237.00=
238.00=end;
239.00=
```

```
240.00={*************************************************************}
                                                                     *}
241.00={*
242.00={*  Stabilise valve by oscillating 4  times. If the pin moves    *}
243.00={*  outside the TOLERANCE during the osc, osc is aborted.        *}
                                                                     *}
244.00={*
245.00={*************************************************************}
246.00=
247.00=procedure STABILISE_VALVE;
248.00=
249.00=begin
250.00=
251.00=    CHECK:=0;
252.00=
253.00=    while CHECK<4  do                          { oscillate the valve }
254.00=       begin
255.00=          PULSE_VALVE((PULSE_FLAG eor $1),10,£0);
256.00=
257.00=          if CHECK=0 then
258.00=             PULSE_VALVE((PULSE_FLAG eor $1),10,£0);
259.00=                      { pulse valve twice in opposite direction  }
260.00=
261.00=          PULSE_VALVE( PULSE_FLAG,10,£0);
262.00=          CHECK:=CHECK+1;
263.00=          ERROR:=NEW_POS-ord(LVDT1);
264.00=
265.00=          if (abs(ERROR)>TOLERANCE) or (COUNTER>699)
266.00=             then CHECK:=4;          { quit loop if out of TOLERANCE }
267.00=       end;
268.00=end;
269.00=
270.00={*************************************************************}
                                                                     *}
271.00={*
272.00={* Pulse the pin to the required position ( NEW_POS ) by pulsing *}
273.00={* the valves for a set time ( PULSE_LENGTH ).  The valve is     *}
274.00={* settled by oscillating using short pulses, once in position.  *}
                                                                     *}
275.00={*
276.00={*************************************************************}
277.00=
278.00=procedure MOVE_PIN_SLOW;
279.00=
280.00=begin
281.00=
282.00=    ERROR:=NEW_POS-ord(LVDT1);
283.00=
284.00=    CHECK_FOR_STORAGE_INTERRUPT;
285.00=
286.00=    if R_FLAG=£1 then                          { record data          }
287.00=    begin
288.00=       FWD_VALVE [COUNTER] :=1;                { store valve is closed}
289.00=       REV_VALVE [COUNTER] :=1;                { store valve is closed}
290.00=    end;
291.00=
292.00=    if ERROR>0 then DIR_FLAG:=£1               { set flag DIR_FLAG for}
293.00=               else DIR_FLAG:=£0;              { dir'n of movement    }
294.00=
295.00=    while abs(ERROR) > 0 do              { pulse the pin until either  }
296.00=    begin                                { the ERROR=0 or the sign of  }
297.00=                                         { the ERROR changes.          }
298.00=       ERROR:=NEW_POS-ord(LVDT1);
299.00=       if COUNTER>699 then ERROR:=0;
```

```
300.00=
301.00=       TEST_FOR_DIRECTION_CHANGE;
302.00=       if CHANGE_FLAG=£1 then ERROR:=0;          { quit loop          }
303.00=
304.00=       if ERROR > 0 then begin                   { move pin forward }
305.00=                         PULSE_VALVE($1,PULSE_TIME,£1);
306.00=                         PULSE_FLAG:=$1;         { valve has pulsed }
307.00=                         end;
308.00=       if ERROR < 0 then begin                   { move pin backward}
309.00=                         PULSE_VALVE($0,PULSE_TIME,£1);
310.00=                         PULSE_FLAG:=$0;         { valve has pulsed }
311.00=                         end;
312.00=
313.00=    end;
314.00=
315.00=    if PULSE_FLAG<>$2 then                  { if valve has been pulsed to }
316.00=    begin                                   { bring the pin into the tol. }
317.00=       STABILISE_VALVE;                     { then oscillate the valve in }
318.00=                                            { order to stabilise the pin. }
319.00=       PULSE_FLAG:=$2;                         { clear pulse flag }
320.00=    end;
321.00=
322.00=end;
323.00=
324.00={*************************************************************************}
325.00={*                                                                       *}
326.00={*  Drive the pin slowly until it contacts the blade                     *}
327.00={*                                                                       *}
328.00={*************************************************************************}
329.00=
330.00=procedure SEEK_CONTACT;
331.00=
332.00=begin
333.00=
334.00=    CHECK_FOR_STORAGE_INTERRUPT;
335.00=
336.00=    if R_FLAG=£1 then                          { record data          }
337.00=    begin
338.00=       FWD_VALVE [COUNTER] :=1;                { store valve is closed}
339.00=       REV_VALVE [COUNTER] :=1;                { store valve is closed}
340.00=    end;
341.00=
342.00=    while (abs(ERROR)>0) and (CONTACT=£1) do
343.00=    begin
344.00=
345.00=       ERROR:=NEW_POS-ord(LVDT1);
346.00=       if COUNTER>699 then ERROR:=0;
347.00=
348.00=       if ERROR > 0 then begin                   { move pin forward }
349.00=                         PULSE_VALVE($1,PULSE_TIME,£1);
350.00=                         PULSE_FLAG:=$1;         { valve has pulsed }
351.00=                         end;
352.00=       if ERROR < 0 then begin                   { move pin backward}
353.00=                         PULSE_VALVE($0,PULSE_TIME,£1);
354.00=                         PULSE_FLAG:=$0;         { valve has pulsed }
355.00=                         end;
356.00=
357.00=    end;
358.00=
359.00=    if PULSE_FLAG<>$2 then                  { if valve has been pulsed to }
```

```
360.00=    begin                                    { bring the pin into the tol. }
361.00=       STABILISE_VALVE;                      { then oscillate the valve in }
362.00=                                             { order to stabilise the pin. }
363.00=       PULSE_FLAG:=$2;                                  { clear pulse flag }
364.00=    end;
365.00=
366.00=end;
367.00={*************************************************************************}
368.00={*                                                                       *}
369.00={*  Hold the pin within the TOL. A one pass loop.                        *}
370.00={*                                                                       *}
371.00={*************************************************************************}
372.00=
373.00=procedure HOLD_PIN;
374.00=
375.00=begin
376.00=
377.00=    ERROR:=NEW_POS-ord(LVDT1);
378.00=
379.00=       if abs(ERROR)>TOLERANCE then MOVE_PIN_SLOW;
380.00=
381.00=       CHECK_FOR_STORAGE_INTERRUPT;
382.00=
383.00=end;
384.00=
385.00={*************************************************************************}
386.00={*                                                                       *}
387.00={* open the valve to move the pin in the direction of                    *}
388.00={* diminishing ERROR until within the BAND-WIDTH. Use                    *}
389.00={* MOVE_PIN_SLOW to pulse the pin to the edge of the TOLERANCE           *}
390.00={* band and stabilise the valve, then MOVE_PIN_SLOW to the target        *}
391.00={*                                                                       *}
392.00={*************************************************************************}
393.00=
394.00=procedure MOVE_PIN_FAST;
395.00=
396.00=begin
397.00=
398.00=    POSITION:=ord(LVDT1);
399.00=    ERROR:=NEW_POS-POSITION;
400.00=    PULSE_FLAG:=$2;                                  { clear pulse flag  }
401.00=
402.00=    while abs(ERROR) > BAND_WIDTH do  {loop until within BAND_WIDTH}
403.00=    begin
404.00=       if ERROR>0 THEN
405.00=          begin
406.00=             WBYTE($4,£0); WBYTE($5,£1);          { drive pin forward }
407.00=
408.00=             if R_FLAG=£1 then                    { record data       }
409.00=             begin
410.00=                FWD_VALVE [COUNTER] :=8;          {store valve is open}
411.00=                REV_VALVE [COUNTER] :=1;       {store valve is closed}
412.00=             end;
413.00=
414.00=          end
415.00=          else begin
416.00=             WBYTE($4,£1); WBYTE($5,£0);          { drive pin backward}
417.00=
418.00=             if R_FLAG=£1 then                    { record data       }
419.00=             begin
```

```
                REV_VALVE [COUNTER] :=8;        {store valve is open}
                FWD_VALVE [COUNTER] :=1;      {store valve is closed}
              end;

         end;

       POSITION:=ord(LVDT1);
       ERROR:=NEW_POS-POSITION;

       CHECK_FOR_STORAGE_INTERRUPT;

       if COUNTER>699 then ERROR:=BAND_WIDTH;            { exit loop }

    end;

end;

{*************************************************************************}
{*    Output a header to the printer                                     *}
{*************************************************************************}

procedure PRINTER_HEADER;

begin
    SEND_OUTPUT_TO_PRINTER;

    write('TABLE: ',TABLE_NO:3,'              SINGLE DIE-PIN TEST RIG');
    writeln('              DATE: ',DAY:2,'.',MONTH:2,'.',YEAR:2);
    writeln;

    write('START: ',START_POS:3,'        TARGET: ',TARGET:3);
    write('        BANDWIDTH: ',BAND_WIDTH:3);
    writeln('        TOLERANCE: ',TOLERANCE:3);

    write('VALVE PULSE TIME: ',PULSE_TIME:3,'            ');
    writeln('                          SAMPLING TIME: ',S_TIME:3);

    write('DATUM PIN CALIBRATION: ',LVDT1_CAL:3,'            ');
    writeln('      SUPPORT PIN CALIBRATION: ',LVDT2_CAL:3);
    writeln;

    write('START    CONTACT    LVDT1    LVDT2    TOL    DATUM');
    writeln('    TARGET    FINISH');
    writeln;

    RESTORE_OUTPUT_TO_SCREEN;
end;

{*************************************************************************}
{*  Output a string to the printer with a specified field-width  *}
{*************************************************************************}

procedure PRINT_VARIABLES;

begin
    SEND_OUTPUT_TO_PRINTER;
    write(P_START:4,P_CONTACT:10,P_LVDT1:10,P_LVDT2:9);
    writeln(P_TOL:8,LVDT1_CAL:8,P_TARGET:10,P_FINISH:10);
    RESTORE_OUTPUT_TO_SCREEN;
end;
```

```
480.00=
481.00={*************************************************************}
482.00={*                    MAIN PROCEDURE                           *}
483.00={*************************************************************}
484.00=
485.00=begin
486.00=    WBYTE($4,£0); WBYTE($5,£0);      { close both hydraulic valves  }
487.00=
488.00=    VECTOR:=$B000;                        { reset interrupt vector  }
489.00=
490.00=    SET_SCREEN;
491.00=    GET_VARIABLES;
492.00=    R_FLAG:=£0;                        { data is not to be recorded  }
493.00=    PRINTER_HEADER;
494.00=
495.00=    CHAC:=£0;                           { initialise termination char }
496.00=
497.00=    while CHAC<>£$0D do                 { loop until RETURN pressed   }
498.00=    begin
499.00=
500.00=       CLEAR_AND_SET_DATA_ARRAYS;
501.00=       WBYTE($8,£1);                      { open main air valve       }
502.00=
503.00=       POSITION:=ord(LVDT1);
504.00=       POSITION_CURSOR(£22,£24);
505.00=       write(POSITION);
506.00=
507.00=       COUNTER:=0;                        { initialise array marker }
508.00=       SET_TIMER(3,S_TIME);               { initialise data timer   }
509.00=
510.00=       NEW_POS:=START_POS;
511.00=       MOVE_PIN_FAST;
512.00=       TEMP:=NEW_POS;                    { store NEW_POS for later use }
513.00=
514.00=          if ERROR>0 then NEW_POS:=NEW_POS-TOLERANCE
515.00=                     else NEW_POS:=NEW_POS+TOLERANCE;
516.00=
517.00=       MOVE_PIN_SLOW;                    { pulse to edge of tol band   }
518.00=       NEW_POS:=TEMP;                    { reload orig value of NEW_POS}
519.00=       ERROR:=ord(LVDT1);
520.00=       MOVE_PIN_SLOW;                    { pulse to START_POS          }
521.00=
522.00=       POSITION_CURSOR(£23,£0);
523.00=       write('HOLDING AT START POSITION');
524.00=
525.00=       POSITION:=ord(LVDT1);
526.00=       POSITION_CURSOR(£22,£24);
527.00=       write(POSITION);
528.00=
529.00=       SET_TIMER(3,50); COUNT:=0;      { initialise timer for delay }
530.00=
531.00=       while COUNT<40 do                 { delay for 2 seconds         }
532.00=          begin
533.00=             if TIMEOUT_3 then
534.00=                begin
535.00=                   COUNT:=COUNT+1;
536.00=                   SET_TIMER(3,50);    { reset timer £3              }
537.00=                end;
538.00=          end;
539.00=
```

```
540.00=       P_START:=ord(LVDT1);                    { start position          }
541.00=
542.00=       COUNTER:=0;                             { reset data store marker }
543.00=       R_FLAG:=£1;                             { - storage NOW required  }
544.00=       SET_TIMER(3,S_TIME);                    { initialise data timer   }
545.00=
546.00=       POSITION_CURSOR(£23,£0);
547.00=       write('DRIVING PIN                ');
548.00=
549.00=       NEW_POS:=TARGET;
550.00=       MOVE_PIN_FAST;
551.00=       SEEK_CONTACT;
552.00=
553.00=       POSITION:=ord(LVDT1);
554.00=       POSITION_CURSOR(£22,£24);
555.00=       write(POSITION);
556.00=
557.00=       P_CONTACT:=POSITION;                    { contact position        }
558.00=
559.00=       POSITION_CURSOR(£23,£0);
560.00=
561.00=          if CONTACT=£0 then write('CONTACT MADE      ')
562.00=                        else write('CONTACT NOT MADE  ');
563.00=
564.00=       NEW_POS:=LVDT1_CAL;
565.00=       MOVE_PIN_SLOW;              { return to nom datum before gauge }
566.00=
567.00=       POSITION_CURSOR(£23,£0);
568.00=       write('BLADE AT DATUM POS ');
569.00=
570.00=       COUNT:=0; CHECK:=0;
571.00=
572.00=    {  while COUNT=0 do
573.00=       begin
574.00=          while COUNT<10 do
575.00=          begin
576.00=             CHECK:=CHECK+ord(LVDT1);
577.00=             COUNT:=COUNT+1;
578.00=          end;
579.00=
580.00=          CHECK:=integer(CHECK/10);
581.00=
582.00=          if (CHECK+1<>ord(LVDT1)) and (CHECK-1<>ord(LVDT1))
583.00=             then begin
584.00=                     COUNT:=0;
585.00=                     STABILISE_VALVE;
586.00=                  end;
587.00=       end; }
588.00=
589.00=       POSITION_CURSOR(£23,£0);
590.00=       write('GAUGING BLADE      ');
591.00=
592.00=       BLADE_TOLERANCE:=ord(LVDT1)-ord(LVDT2);
593.00=       BLADE_TOLERANCE:=BLADE_TOLERANCE-(LVDT1_CAL-LVDT2_CAL);
594.00=
595.00=       P_LVDT1:=ord(LVDT1);
596.00=       P_LVDT2:=ord(LVDT2);;
597.00=       P_TOL:=BLADE_TOLERANCE;
598.00=
599.00=       NEW_POS:=LVDT1_CAL+integer(BLADE_TOLERANCE/2);
```

```
600.00=       P_TARGET:=NEW_POS;                              { target position  }
601.00=
602.00=       MOVE_PIN_SLOW;              { gauge blade and move to datum pos}
603.00=
604.00=       POSITION_CURSOR(£23,£0);
605.00=       write('BLADE POSITIONED  ');
606.00=
607.00=       while COUNTER<700 do                  { data array 700 elements }
608.00=          begin
609.00=             HOLD_PIN;
610.00=          end;
611.00=
612.00=       POSITION_CURSOR(£23,£0);
613.00=       write('TIMED OUT                    ');
614.00=
615.00=       POSITION:=ord(LVDT1);
616.00=       POSITION_CURSOR(£22,£24);
617.00=       write(POSITION);
618.00=
619.00=       P_FINISH:=POSITION;                             { final position   }
620.00=
621.00=       PRINT_VARIABLES;                          { print-out line of text }
622.00=
623.00=       WBYTE($8,£0);                             { close main air valve    }
624.00=
625.00=       WBYTE($E785,£1);                        { enable keyboard scan      }
626.00=       SET_TIMER(3,50); COUNT:=0;      { initialise timer for delay }
627.00=
628.00=       while COUNT<40 do                       { delay for 2 seconds       }
629.00=          begin
630.00=             CHAC:=SCANKEY;                { is RETURN key pressed ?      }
631.00=             if CHAC=£$0D then COUNT:=40;        { quit loop              }
632.00=             if TIMEOUT_3 then
633.00=                begin
634.00=                   COUNT:=COUNT+1;
635.00=                   SET_TIMER(3,50);    { reset timer £3                  }
636.00=                end;
637.00=          end;                                    { main cycling loop      }
638.00=    end;
639.00=end.
```

Appendix IV

Multiple Die Pin Rig Software

Pin Position Calibration

A program to allow each of the ten pins to be pulsed forwards and backwards using the terminal keyboard. Each die pin LVDT reading is displayed at all times. In addition the parameters for each pin associated with the valve pulse-length may be changed if required.

```
1.00=program TESTPINS (input,output);
2.00=
3.00={$IGLOVARS}
4.00={$IUEXTS}
5.00=
6.00={*****************************************************************}
7.00={**                                                               **}
8.00={**  procedure SET_TIMER ( VIA, MILLISEC : integer)               **}
9.00={**  written by M. Taylor of Cirrus Reynolds 9/11/84              **}
10.00={**                                                              **}
11.00={*****************************************************************}
12.00=
13.00=procedure TIMVIA (BASE:hex); external;
14.00=procedure SET_TIMER (VIA, MILLISEC : integer);
15.00=
16.00=var BASE : hex;
17.00=
18.00=begin
19.00=    TIMEOUT_4 := false;
20.00=    BASE := VIA4;
21.00=
22.00=    TIMVIA (BASE);
23.00=    IRQ (£1);
24.00=    WBYTE (BASE+$8, chr (MILLISEC mod 256));
25.00=    WBYTE (BASE+$9, chr (MILLISEC div 256));
26.00=end;
27.00=
28.00=
29.00={*****************************************************************}
30.00={**                                                              **}
31.00={** ensure all valves are closed. allow oil reservoir to        **}
32.00={** re-charge.  open main air valve.                            **}
33.00={**                                                              **}
34.00={*****************************************************************}
35.00=
36.00=procedure HOUSE_KEEPING;
37.00=
38.00=begin
39.00=    VECTOR:=$B000;                          { reset interrupt vector}
40.00=    COUNT:=£1;
41.00=
42.00=    while COUNT < £11 do                    { close all valves      }
43.00=    begin
44.00=       FWD_VALVE [COUNT] :=£0;
45.00=       REV_VALVE [COUNT] :=£0;
46.00=       PULSE_LENGTH_FWD   [COUNT] := 10; {pre-set for valves}
47.00=       PULSE_LENGTH_PAUSE [COUNT] := 20;
48.00=       PULSE_LENGTH_REV   [COUNT] := 10;
49.00=       COUNT:=COUNT + £1;
50.00=    end;
51.00=
52.00=    WBYTE ($0,£0);                          { close main air valve  }
53.00=
54.00=    COUNT:=£0;
55.00=    SET_TIMER (4,100);                      { delay 10 seconds      }
56.00=
57.00=    while COUNT < £100 do
58.00=    begin
59.00=       if TIMEOUT_4 then
```

```
60.00=       begin
61.00=          COUNT:=COUNT + £1;
62.00=          SET_TIMER(4,100);
63.00=       end;
64.00=    end;
65.00=
66.00=    WBYTE ($0,£1);                          { open main air valve    }
67.00=
68.00=end;
69.00=
70.00=
71.00={*****************************************************************}
72.00={**                                                               **}
73.00={** all pins withdrawn from the die cavity for safety            **}
74.00={** all reverse valves opened for 2 seconds followed by          **}
75.00={** a 20 ms settling time after valve closure.                    **}
76.00={**                                                               **}
77.00={*****************************************************************}
78.00=
79.00=
80.00=procedure RETRACT_ALL_PINS;
81.00=
82.00=begin
83.00=    COUNT:=£1;
84.00=
85.00=    while COUNT < £11 do                    { poll all pins          }
86.00=    begin
87.00=       FWD_VALVE [COUNT] :=£0;              { close forward valve    }
88.00=       REV_VALVE [COUNT] :=£1;              { open reverse valve     }
89.00=       COUNT:=COUNT + £1;
90.00=    end;
91.00=
92.00=    SET_TIMER (4,100);
93.00=    COUNTER:=0;
94.00=
95.00=    while COUNTER < 20 do                { 2 second loop            }
96.00=    begin
97.00=       if TIMEOUT_4 then                 { interrupt generated      }
98.00=       begin
99.00=          COUNTER := COUNTER + 1;
100.00=         SET_TIMER (4,100);
101.00=      end;
102.00=   end;
103.00=
104.00=   while COUNT > £0 do                   { close all reverse valves }
105.00=   begin
106.00=      COUNT:=COUNT - £1;
107.00=      REV_VALVE [COUNT] := £0;
108.00=   end;
109.00=
110.00=   SET_TIMER(4,20);                      { reset for 20 ms loop     }
111.00=
112.00=   while TIMEOUT_4 = false do           { allow valves to stabilise }
113.00=   begin
114.00=   end;
115.00=
116.00=   SAFE_TO_LOAD:=true;
117.00=
118.00=end;
119.00=
```

```
120.00=
121.00={*************************************************************}
122.00={**                                                           **}
123.00={**  move one pin forward by opening the forward valve         **}
124.00={**  for a set period (PULSE_LENGTH) - followed by a           **}
125.00={**  settling time of PAUSE_LENGTH                            **}
126.00={**                                                           **}
127.00={*************************************************************}
128.00=
129.00=procedure PULSE_PIN_FWD (PIN,FLAG:byte);
130.00=
131.00=var COUNTER : integer;
132.00=
133.00=begin
134.00=    REV_VALVE [PIN] :=£0;
135.00=
136.00=    COUNTER:=PULSE_LENGTH_FWD [PIN];
137.00=    SET_TIMER(4,COUNTER);
138.00=
139.00=    FWD_VALVE [PIN] :=£1;
140.00=
141.00=    while TIMEOUT_4 = false do;{ open valve for PULSE_LENGTH}
142.00=
143.00=    FWD_VALVE [PIN] :=£0;
144.00=
145.00=    COUNTER := PULSE_LENGTH_PAUSE [PIN];
146.00=    SET_TIMER(4,COUNTER);
147.00=    while TIMEOUT_4 = false do; { close valve for 10 msec      }
148.00=end;
149.00=
150.00=
151.00={*************************************************************}
152.00={**                                                           **}
153.00={**  move one pin in reverse by opening the reverse           **}
154.00={**  valve for a set period (PULSE_LENGTH) - followed         **}
155.00={**  by a settling time of PAUSE_LENGTH                       **}
156.00={**                                                           **}
157.00={*************************************************************}
158.00=
159.00=procedure PULSE_PIN_REV (PIN,FLAG:byte);
160.00=
161.00=var COUNTER : integer;
162.00=
163.00=begin
164.00=    FWD_VALVE [PIN] :=£0;
165.00=
166.00=    COUNTER:=PULSE_LENGTH_REV [PIN];
167.00=    SET_TIMER(4,COUNTER);
168.00=
169.00=    REV_VALVE [PIN] :=£1;
170.00=
171.00=    while TIMEOUT_4 = false do; {open valve for PULSE_LENGTH}
172.00=
173.00=    REV_VALVE [PIN] :=£0;
174.00=
175.00=    COUNTER := PULSE_LENGTH_PAUSE [PIN];
176.00=    SET_TIMER(4,COUNTER);
177.00=    while TIMEOUT_4 = false do;
178.00=end;
179.00=
```

```
180.00=
181.00={*************************************************************}
182.00={**                                                           **}
183.00={**  transfer all bytes from LVDT cards into array             **}
184.00={**                                                           **}
185.00={*************************************************************}
186.00=
187.00=procedure UPDATE_LVDTS;
188.00=
189.00=begin
190.00=   LVDT [£1] := RBYTE($EB00);
191.00=   LVDT [£2] := RBYTE($EB01);
192.00=   LVDT [£3] := RBYTE($EB02);
193.00=   LVDT [£4] := RBYTE($EB03);
194.00=   LVDT [£5] := RBYTE($EB04);
195.00=   LVDT [£6] := RBYTE($EB05);
196.00=   LVDT [£7] := RBYTE($EB06);
197.00=   LVDT [£8] := RBYTE($EB07);
198.00=   LVDT [£9] := RBYTE($EB10);
199.00=   LVDT [£10]:= RBYTE($EB11);
200.00=end;
201.00=
202.00=
203.00=
204.00={*************************************************************}
205.00={**                                                           **}
206.00={**   oscillate solenoids at low level prior to moving        **}
207.00={**                                                           **}
208.00={*************************************************************}
209.00=
210.00=procedure WARMUP (PIN : byte);
211.00=
212.00=var ERROR : integer;
213.00=
214.00=begin
215.00=   COUNTER := PULSE_LENGTH_PAUSE [PIN];
216.00=
217.00=   ERROR := 0;
218.00=   while ERROR < 15 do
219.00=   begin
220.00=      SET_TIMER(4,COUNTER);
221.00=      FWD_VALVE [PIN] := £1;
222.00=      while TIMEOUT_4 = false do;
223.00=      FWD_VALVE [PIN] := £0;
224.00=      SET_TIMER(4,COUNTER);
225.00=      REV_VALVE [PIN] := £1;
226.00=      while TIMEOUT_4 = false do;
227.00=      REV_VALVE [PIN] := £0;
228.00=
229.00=      ERROR := ERROR + 1;
230.00=   end;
231.00=end;
232.00=
233.00=
234.00={*************************************************************}
235.00={**                                                           **}
236.00={**      move pin forward by increasing pulse length          **}
237.00={**                                                           **}
238.00={*************************************************************}
239.00=
```

```
240.00=procedure PIN_FWD (PIN : byte);
241.00=
242.00=var ERROR, START : integer;
243.00=
244.00=begin
245.00=    COUNTER := PULSE_LENGTH_FWD [PIN];
246.00=
247.00=    UPDATE_LVDTS;
248.00=    START := ord(LVDT [PIN]);
249.00=    WARMUP (PIN);
250.00=    ERROR := 0;
251.00=
252.00=    while ERROR < 4 do
253.00=    begin
254.00=       SET_TIMER(4,COUNTER);
255.00=       FWD_VALVE [PIN] := £1;
256.00=       while TIMEOUT_4 = false do;
257.00=       FWD_VALVE [PIN] := £0;
258.00=       SET_TIMER(4,10);
259.00=       REV_VALVE [PIN] := £1;
260.00=       while TIMEOUT_4 = false do;
261.00=       REV_VALVE [PIN] := £0;
262.00=
263.00=       UPDATE_LVDTS;
264.00=       ERROR := ord(LVDT [PIN]) - START;
265.00=    end;
266.00=
267.00=    WARMUP (PIN);
268.00=end;
269.00=
270.00=
271.00={*************************************************************}
272.00={**                                                         **}
273.00={**          recharge oil reservoir                         **}
274.00={**                                                         **}
275.00={*************************************************************}
276.00=
277.00=procedure RECHARGE_RESERVOIR;
278.00=
279.00=begin
280.00=    TOLERANCE := 0;
281.00=    SET_TIMER (4,9000);
282.00=    while TIMEOUT_4 = false do     { reservoir re-fill delay }
283.00=    begin
284.00=       WBYTE($0,£0);                { close main air valve   }
285.00=    end;
286.00=
287.00=    WBYTE($0,£1);                   { re-open main air valve }
288.00=
289.00=end;
290.00=
291.00=
292.00={*************************************************************}
293.00={**                                                         **}
294.00={**  allow a new pulse time to be set for an individual      **}
295.00={**  valve. the forward, reverse & pause times are set       **}
296.00={**  these values are set to a default of 10 when the        **}
297.00={**  program is first run and are therefore transitory       **}
298.00={**                                                         **}
299.00={*************************************************************}
```

```
300.00=
301.00=procedure SET_PULSE_LENGTH;
302.00=
303.00=var TEMP,CHAC : byte;
304.00=
305.00=begin
306.00=    TEMP:=£0;
307.00=
308.00=    while STATUS = false do
309.00=    begin
310.00=    end;
311.00=
312.00=    CHAC:=INKEY;
313.00=    TEMP:=byte(CHAC - £$30);
314.00=
315.00=    while STATUS = false do
316.00=    begin
317.00=    end;
318.00=
319.00=    CHAC:=INKEY;
320.00=
321.00=    TEMP:=(TEMP*£10)+byte(CHAC-£$30);
322.00=
323.00=    PULSE_LENGTH := integer(TEMP);
324.00=
325.00=end;
326.00={*************************************************************}
327.00={**                                                         **}
328.00={**                  main procedure                         **}
329.00={**                                                         **}
330.00={*************************************************************}
331.00=
332.00=
333.00=begin
334.00=    HOUSE_KEEPING;
335.00=    WBYTE($E785,£1);                { enable keyboard scan      }
336.00=
337.00=    write (£$0C);                   { clear screen              }
338.00=    write (£$1A);                   { home cursor               }
339.00=
340.00=    COUNT:=£0;
341.00=
342.00=    while COUNT < £13 do
343.00=    begin
344.00=       writeln;
345.00=       COUNT:=COUNT + £1;
346.00=    end;
347.00=
348.00=    writeln('Enter pin number');
349.00=    writeln('Enter  F  to move selected pin FORWARD');
350.00=    writeln('Enter  R  to move selected pin BACKWARD');
351.00=    writeln('Enter  P  to set valve pulse length (ms)');
352.00=    writeln('Enter  M  to re-charge oil reservoir');
353.00=    writeln('Enter RETURN to quit program');
354.00=
355.00=    if STATUS then CHAC:=INKEY
356.00=              else CHAC:=£$00;
357.00=
358.00=    while CHAC <> £$0D do
359.00=
```

```
360.00=    begin
361.00=       write(£$1A);
362.00=       COUNT:=£1;
363.00=
364.00=       while COUNT < £11 do
365.00=       begin
366.00=          UPDATE_LVDTS;
367.00=          COUNTER:=ord(LVDT [COUNT]);
368.00=          TEMP:=integer(COUNT);
369.00=          write('LVDT [',TEMP,'] : ',COUNTER,'     PULSE LENGTH:'),
370.00=          write(PULSE_LENGTH_FWD [COUNT],PULSE_LENGTH_PAUSE [COUNT
371.00=          writeln(PULSE_LENGTH_REV [COUNT]);
372.00=          COUNT:=COUNT + £1;
373.00=       end;
374.00=
375.00=       if CHAC>£$2F then COUNT:=byte(CHAC-£$30)     { ascii   }
376.00=                    else COUNT:=£1;
377.00=
378.00=       if STATUS then CHAC:=INKEY;
379.00=
380.00=       if (CHAC=£$30) and (COUNT=£1) then COUNT:=£10;
381.00=
382.00=       write(£$1A);
383.00=       COUNTER:=1;
384.00=       CHAC:=£$3A;              { pre-set for following loop    }
385.00=
386.00=       while COUNTER < integer(COUNT) do     { select pin     }
387.00=       begin
388.00=          writeln;
389.00=          COUNTER:=COUNTER + 1;
390.00=       end;
391.00=
392.00=       while CHAC > £$39 do
393.00=       begin
394.00=
395.00=          if STATUS then CHAC:=INKEY;
396.00=
397.00=          if CHAC=£$46 then PULSE_PIN_FWD(COUNT,£0);
398.00=    {     if CHAC=£$46 then PIN_FWD (COUNT);                    }
399.00=                              { alternative pulsing strategy  }
400.00=          if CHAC=£$52 then PULSE_PIN_REV(COUNT,£0);
401.00=          if CHAC=£$4D then RECHARGE_RESERVOIR;
402.00=
403.00=          if CHAC=£$50 then
404.00=          begin
405.00=             while STATUS = false do
406.00=             begin
407.00=             end;
408.00=
409.00=             CHAC:=INKEY;
410.00=
411.00=             if CHAC=£$46 then
412.00=             begin
413.00=                SET_PULSE_LENGTH;
414.00=                PULSE_LENGTH_FWD [COUNT] :=PULSE_LENGTH;
415.00=             end;
416.00=
417.00=             if CHAC=£$52 then
418.00=             begin
419.00=                SET_PULSE_LENGTH;
```

```
420.00=                PULSE_LENGTH_REV [COUNT] :=PULSE_LENGTH;
421.00=             end;
422.00=
423.00=             if CHAC=£$50 then
424.00=             begin
425.00=                SET_PULSE_LENGTH;
426.00=                PULSE_LENGTH_PAUSE [COUNT] :=PULSE_LENGTH;
427.00=             end;
428.00=          end;
429.00=
430.00=          UPDATE_LVDTS;
431.00=          COUNTER:=ord(LVDT [COUNT]);
432.00=          TEMP:=integer(COUNT);
433.00=          write(CR,'LVDT [',TEMP,'] : ',COUNTER,'     PULSE LENGTH:
434.00=          write(PULSE_LENGTH_FWD [COUNT],PULSE_LENGTH_PAUSE [COUNT
435.00=          write(PULSE_LENGTH_REV [COUNT]);
436.00=
437.00=          if (CHAC>£$39) then CHAC:=£$3A; { clear for input }
438.00=
439.00=       end;
440.00=
441.00=    end;
442.00=
443.00=end.
```

```
1.00={**  GLOVARS                                                    **}
2.00=
3.00={*************************************************************}
4.00={**                                                            **}
5.00={**  global variable file for program MULTIPIN                  **}
6.00={**                                                            **}
7.00={*************************************************************}
8.00=
9.00=const              CR = £$0D;          { carriage return       }
10.00=                  LF = £$0A;          { line feed             }
11.00=                  EQ = £$3D;          { equals =              }
12.00=                  ES = £$1B;          { esc                   }
13.00=
14.00=                VIA3 = $EE00;         { for timer interrupt   }
15.00=                VIA4 = $EE10;
16.00=
17.00=var
18.00=
19.00={* inputs                                                     **}
20.00=
21.00=    LVDT                      : array [£1..£10] of byte;
22.00=    PIN_CONTACT               : array [£1..£10] of byte at $0020;
23.00=
24.00=    BLADE_WAITING               : boolean at $002A;
25.00=    BLADE_LOADED                : boolean at $002B;
26.00=    ROBOT_WITHDRAWN             : boolean at $002C;
27.00=    POURING_IN_PROGRESS         : boolean at $002D;
28.00=
29.00={* outputs                                                    **}
30.00=
31.00=    FWD_VALVE                 : array [£1..£10] of byte at $0001;
32.00=    REV_VALVE                 : array [£1..£10] of byte at $000B;
33.00=
34.00=    SAFE_TO_LOAD                : boolean at $0015;
35.00=    ROBOT_MAY_WITHDRAW          : boolean at $0016;
36.00=    BLADE_NOT_CLAMPED           : boolean at $0017;
37.00=
38.00={* internal data                                              **}
39.00=
40.00=    LVDT_CAL                  : array [£1..£10] of integer;
41.00=    LVDT_GAUGE                : array [£1..£10] of integer;
42.00=    NOMINAL                   : array [£1..£10] of integer;
43.00=    PULSE_LENGTH_FWD          : array [£1..£10] of integer;
44.00=    PULSE_LENGTH_PAUSE        : array [£1..£10] of integer;
45.00=    PULSE_LENGTH_REV          : array [£1..£10] of integer;
46.00=    DIR_FLAG                  : array [£1..£10] of byte;
47.00=    PULSE_FLAG                : array [£1..£10] of byte;
48.00=    OSC_COUNT                 : array [£1..£10] of byte;
49.00=
50.00=    CLAMP_POS                 : array [£1..£10] of integer;
51.00=    NOM_POS                   : array [£1..£10] of integer;
52.00=    GAUGE_POS                 : array [£1..£10] of integer;
53.00=    FINAL_POS                 : array [£1..£10] of integer;
54.00=
55.00=    PIN_POS                     : array [1..5000] of integer
56.00=                                                  at $4000;
57.00=
58.00={* system variables                                           *}
59.00=
```

```
60.00=    PRINTER_FLAG          : hex at $CD10;
61.00=    VECTOR                : hex at $DFCE;   { interrupt vector  }
62.00=    TERMINAL              : hex;
63.00=    TIMEOUT_4             : boolean at $E792;
64.00=
65.00=    COUNT                 : byte;           { general counter   }
66.00=    COUNTER               : integer;        { general counter   }
67.00=    ALL_PINS_CONTACTED    : boolean;
68.00=    TOLERANCE             : integer;
69.00=    CHAC                  : byte;
70.00=    TEMP                  : integer;
71.00=    PULSE_LENGTH          : integer;
```

```
1.00={*************************************************************************}
2.00={*************************************************************************}
3.00={**                                                                       **}
4.00={**    Declarations of procedures and functions which are                  **}
5.00={**    available in the user library URL.RO                               **}
6.00={**                                                                       **}
7.00={**    This must be updated along with the .RO file.                      **}
8.00={**                                                                       **}
9.00={**    Version 1.0    -    October 1984                                   **}
10.00={**           1.1     -    10/ 3/86          -    M. Taylor              **}
11.00={**                                                                      **}
12.00={**                                                                      **}
13.00={**                                                                      **}
14.00={**                                                                      **}
15.00={*************************************************************************}
16.00={*************************************************************************}
17.00=
18.00=
19.00=procedure DELAY (millisec: integer);  external;
20.00=procedure FIRQ ( endis : £0..£1 );  external;
21.00=procedure FLEX ( command : string [80] ); external;
22.00=function  INKEY : char;  external;
23.00=procedure IRQ ( endis : £0..£1 );  external;
24.00=procedure JUMP (address: hex);  external;
25.00=function  RBYTE (address: hex) : byte;  external;
26.00=procedure READ_SECTOR (drive:0..1; track:0..79; sector:0..19;
27.00=                       var contents: array [0..255] of byte); external;
28.00=function  RWORD (address: hex) : hex;  external;
29.00=function  STATUS : boolean; external;
30.00=procedure WBYTE (address: hex; data: byte);  external;
31.00=procedure WRITE_SECTOR (drive:0..1; track:0..79; sector:0..19;
32.00=                       var contents: array [0..255] of byte); external;
33.00=procedure WWORD (address: hex; data: hex);  external;
34.00=
```

Appendix V

Multiple Die Pin Rig Software

Blade Gauging and Positioning

A program to simulate robot loading of the die, blade clamping, blade gauging and blade repositioning.

```
1.00=program MULTIPIN (input,output);
2.00=
3.00={$IGLOVARS}
4.00={$IUEXTS}
5.00=
6.00={*************************************************************}
                                                                  **}
7.00={**                                                          **}
8.00={**  procedure SET_TIMER ( VIA, MILLISEC : integer)          **}
9.00={**  written by M. Taylor of Cirrus Reynolds 9/11/84         **}
                                                                  **}
10.00={**
11.00={*************************************************************}
12.00=
13.00=procedure TIMVIA (BASE:hex); external;
14.00=
15.00=procedure SET_TIMER (VIA, MILLISEC :integer);
16.00=
17.00=var BASE : hex;
18.00=
19.00=begin
20.00=    TIMEOUT_4 := false;
21.00=    BASE := VIA4;
22.00=    TIMVIA (BASE);
23.00=    IRQ (£1);
24.00=    WBYTE (BASE+$8, chr (MILLISEC mod 256));
25.00=    WBYTE (BASE+$9, chr (MILLISEC div 256));
26.00=end;
27.00=
28.00=
29.00={*************************************************************}
                                                                  **}
30.00={**                                                         **}
31.00={**   transfer all bytes from LVDT cards into array         **}
                                                                  **}
32.00={**
33.00={*************************************************************}
34.00=
35.00=procedure UPDATE_LVDTS;
36.00=
37.00=begin
38.00=    LVDT  [£1] := RBYTE ($EB00);
39.00=    LVDT  [£2] := RBYTE ($EB01);
40.00=    LVDT  [£3] := RBYTE ($EB02);
41.00=    LVDT  [£4] := RBYTE ($EB03);
42.00=    LVDT  [£5] := RBYTE ($EB04);
43.00=    LVDT  [£6] := RBYTE ($EB05);
44.00=    LVDT  [£7] := RBYTE ($EB06);
45.00=    LVDT  [£8] := RBYTE ($EB07);
46.00=    LVDT  [£9] := RBYTE ($EB10);
47.00=    LVDT [£10] := RBYTE ($EB11);
48.00=end;
49.00=
50.00=
51.00={*************************************************************}
                                                                  **}
52.00={**                                                         **}
53.00={**  allow oil reservoir to recharge by turning air off      **}
                                                                  **}
54.00={**
55.00={*************************************************************}
56.00=
57.00=procedure RECHARGE_RESERVOIR;
58.00=
59.00=begin
```

```
60.00=    WBYTE ($0,£0);                    { close main air valve      }
61.00=
62.00=    SET_TIMER (4,25000);              { 25 second timer           }
63.00=    while TIMEOUT_4 = false do;       { allow oil to re-charge    }
64.00=
65.00=    WBYTE ($0,£1);                    { re-open main air valve    }
66.00=
67.00=    SET_TIMER (4,5000);               { 5 second timer            }
68.00=    while TIMEOUT_4 = false do;       { allow pressure to rise    }
69.00=end;
70.00=
71.00=
72.00={*************************************************************}
73.00={**                                                           **}
74.00={** all pins withdrawn from the die cavity for safety        **}
75.00={** all reverse valves opened for 10 seconds followed by     **}
76.00={** re-charging of the oil reservoir                         **}
77.00={**                                                           **}
78.00={*************************************************************}
79.00=
80.00=procedure RETRACT_ALL_PINS;
81.00=
82.00=begin
83.00=    COUNT:=£1;
84.00=
85.00=    while COUNT < £11 do                  { poll all pins         }
86.00=    begin
87.00=        FWD_VALVE [COUNT] :=£0;           { close forward valve   }
88.00=        REV_VALVE [COUNT] :=£1;           { open reverse valve    }
89.00=        COUNT:=COUNT + £1;
90.00=    end;
91.00=
92.00=    SET_TIMER (4,100);
93.00=    COUNTER:=0;
94.00=
95.00=    while COUNTER < 100 do            { 10 second loop            }
96.00=    begin
97.00=        if TIMEOUT_4 then             { interrupt generated       }
98.00=        begin
99.00=            COUNTER := COUNTER + 1;
100.00=           SET_TIMER (4,100);
101.00=       end;
102.00=   end;
103.00=
104.00=   while COUNT > £0 do               { close all reverse valves  }
105.00=   begin
106.00=       COUNT:=COUNT - £1;
107.00=       REV_VALVE [COUNT] := £0;
108.00=   end;
109.00=
110.00=   SET_TIMER(4,90);                  { reset for 90 ms loop      }
111.00=   while TIMEOUT_4 = false do;
112.00=
113.00=   RECHARGE_RESERVOIR;
114.00=end;
115.00=
116.00=
117.00={*************************************************************}
118.00={**                                                           **}
119.00={**    clear and set data arrays      set screen               **}
```

```
120.00={**                                                              **}
121.00={*************************************************************}
122.00=
123.00=procedure HOUSE_KEEPING;
124.00=
125.00=begin
126.00=    VECTOR:=$B000;                          { reset interrupt vector}
127.00=    write (£$0C); write (£$1A);   { clear screen/home cursor }
128.00=    writeln ('RETRACTING PINS & RE-CHARGING RESERVOIR');
129.00=
130.00=    COUNT:=£1;
131.00=
132.00=    while COUNT < £11 do                    { close all valves      }
133.00=    begin
134.00=       FWD_VALVE           [COUNT] := £0;
135.00=       REV_VALVE           [COUNT] := £0;
136.00=       PULSE_LENGTH_FWD    [COUNT] := 15;
137.00=       PULSE_LENGTH_REV    [COUNT] := 15;
138.00=       LVDT_CAL            [COUNT] := 128; {calibration value}
139.00=       COUNT:=COUNT + £1;
140.00=    end;
141.00=    WBYTE($0,£1);                             { open main air valve }
142.00=
143.00=    SET_TIMER (4,5000);                       { 5 second timer      }
144.00=    while TIMEOUT_4 = false do;     { allow pressure to rise }
145.00=
146.00=    RETRACT_ALL_PINS;
147.00=end;
148.00=
149.00=
150.00={*************************************************************}
                                                                        **}
151.00={**                                                              **}
152.00={**  move one pin forward by opening the forward valve            **}
153.00={**  for the specified forward valve pulse length                 **}
154.00={**  and de-energise for the valve pause length.                  **}
155.00={**  if the flag is set the opposing pin will be moved            **}
156.00={**  in a similar manner only in reverse.                         **}
                                                                        **}
157.00={**                                                              **}
158.00={*************************************************************}
159.00=
160.00=procedure PULSE_PIN_FWD (PIN,FLAG:byte);
161.00=
162.00=var COUNTER : integer;
163.00=
164.00=begin
165.00=    REV_VALVE [PIN] :=£0;
166.00=    if FLAG=£1 then FWD_VALVE [PIN+£1] :=£0;
167.00=
168.00=    FWD_VALVE [PIN] :=£1;
169.00=    if FLAG=£1 then REV_VALVE [PIN+£1] :=£1;
170.00=
171.00=    COUNTER := PULSE_LENGTH_FWD [PIN];
172.00=    SET_TIMER(4,COUNTER);
173.00=    while TIMEOUT_4 = false do;{ open valve for pulselength }
174.00=
175.00=    FWD_VALVE [PIN] :=£0;
176.00=    if FLAG=£1 then REV_VALVE [PIN+£1] :=£0;
177.00=
178.00=    REV_VALVE [PIN] := £1;
179.00=    if FLAG = £1 then FWD_VALVE [PIN + £1] := £1;
```

```
180.00=
181.00=    SET_TIMER(4,5);
182.00=    while TIMEOUT_4 = false do;{ pulse valve closed for 5ms }
183.00=
184.00=    REV_VALVE [PIN] := £0;
185.00=    if FLAG = £1 then FWD_VALVE [PIN + £1] := £0;
186.00=
187.00=    COUNTER := 2 * PULSE_LENGTH_FWD [PIN];
188.00=    SET_TIMER (4,COUNTER);
189.00=    while TIMEOUT_4 = false do; { stabilise 2 x pulselength }
190.00=
191.00=    UPDATE_LVDTS;
192.00=end;
193.00=
194.00=
195.00={*************************************************************}
196.00={**                                                         **}
197.00={**  move one pin in reverse by opening reverse valve        **}
198.00={**  for the specified reverse valve pulse length           **}
199.00={**  and de-energise for the valve pause length.            **}
200.00={**  if the flag is set the opposing pin will be moved      **}
201.00={**  in a similar manner only forwards.                     **}
202.00={**                                                         **}
203.00={*************************************************************}
204.00=
205.00=procedure PULSE_PIN_REV (PIN,FLAG:byte);
206.00=
207.00=var COUNTER : integer;
208.00=
209.00=begin
210.00=    FWD_VALVE [PIN] :=£0;
211.00=    if FLAG=£1 then REV_VALVE [PIN+£1] :=£0;
212.00=
213.00=    REV_VALVE [PIN] :=£1;
214.00=    if FLAG=£1 then FWD_VALVE [PIN+£1] :=£1;
215.00=
216.00=    COUNTER := PULSE_LENGTH_REV [PIN];
217.00=    SET_TIMER(4,COUNTER);
218.00=    while TIMEOUT_4 = false do;{ open valve for pulselength }
219.00=
220.00=    REV_VALVE [PIN] :=£0;
221.00=    if FLAG=£1 then FWD_VALVE [PIN+£1] :=£0;
222.00=
223.00=    FWD_VALVE [PIN] := £1;
224.00=    if FLAG = £1 then REV_VALVE [PIN + £1] := £1;
225.00=
226.00=    SET_TIMER(4,5);
227.00=    while TIMEOUT_4 = false do;{ pulse valve closed for 5ms }
228.00=
229.00=    FWD_VALVE [PIN] := £0;
230.00=    if FLAG = £1 then REV_VALVE [PIN + £1] := £0;
231.00=
232.00=    COUNTER := 2 * PULSE_LENGTH_REV [PIN];
233.00=    SET_TIMER (4,COUNTER);
234.00=    while TIMEOUT_4 = false do; { stabilise 2 x pulselength }
235.00=
236.00=    UPDATE_LVDTS;
237.00=end;
238.00=
239.00=
```

```
240.00={*************************************************************}
241.00={**                                                           **}
242.00={**                                                           **}
243.00={**                                                           **}
244.00={*************************************************************}
245.00=
246.00=procedure STABILISE_VALVE(PIN,FLAG:byte);
247.00=
248.00=begin
249.00=    if OSC_COUNT [PIN] < £4 then
250.00=    begin
251.00=       if DIR_FLAG [PIN] = £1
252.00=             then PULSE_PIN_REV (PIN,FLAG) { pin moving fwd }
253.00=             else PULSE_PIN_FWD (PIN,FLAG);
254.00=
255.00=       if OSC_COUNT [PIN] = £0 then          { if first osc   }
256.00=       begin                                 { pulse valve    }
257.00=          if DIR_FLAG [PIN] = £1             { twice in opp   }
258.00=              then PULSE_PIN_REV (PIN,FLAG){ direction      }
259.00=              else PULSE_PIN_FWD (PIN,FLAG);
260.00=       end;
261.00=
262.00=       if DIR_FLAG [PIN] = £1
263.00=             then PULSE_PIN_FWD (PIN,FLAG) { pin moving fwd }
264.00=             else PULSE_PIN_REV (PIN,FLAG);
265.00=
266.00=       OSC_COUNT [PIN] := OSC_COUNT [PIN] + £1;
267.00=    end;
268.00=end;
269.00=
270.00=
271.00={*************************************************************}
272.00={**                                                           **}
273.00={**  a number of pins in numerical sequence from START_       **}
274.00={**  PIN to END_PIN are driven forward until each pin         **}
275.00={**  contacts the blade.  the valves are then stabilised      **}
276.00={**  the X and Y pins can be clamped independently.           **}
277.00={**                                                           **}
278.00={*************************************************************}
279.00=
280.00=procedure CLAMP_BLADE ( START_PIN,END_PIN : byte);
281.00=
282.00=begin
283.00=    COUNT:=START_PIN;
284.00=
285.00=    while COUNT <= END_PIN do
286.00=    begin
287.00=       REV_VALVE    [COUNT] := £0;
288.00=       PIN_CONTACT  [COUNT] := £0;      { clear contact array }
289.00=       OSC_COUNT    [COUNT] := £0;      { valve stabilisation }
290.00=       DIR_FLAG     [COUNT] := £1;      { all pins moving fwd }
291.00=       COUNT := COUNT + £1;
292.00=    end;
293.00=
294.00=    ALL_PINS_CONTACTED := false;
295.00=
296.00=    while ALL_PINS_CONTACTED = false do
297.00=    begin
298.00=       COUNT:=START_PIN;                 { start at first pin   }
299.00=       ALL_PINS_CONTACTED := true;       { pre-set flag         }
```

```
300.00=       UPDATE_LVDTS;
301.00=
302.00=       while COUNT <= END_PIN do  { loop all specified pins }
303.00=
304.00=       begin
305.00=          if PIN_CONTACT [COUNT]=£0 then
306.00=             begin                              { pin not contacted }
307.00=                ALL_PINS_CONTACTED := false;  { clear flag  }
308.00=                if LVDT [COUNT] > £0 then
309.00=                begin
310.00=                   FWD_VALVE [COUNT] := £0;
311.00=                   PULSE_PIN_FWD (COUNT,£0);
312.00=                   OSC_COUNT [COUNT] := £0;
313.00=                end
314.00=                else FWD_VALVE [COUNT] := £1;  { open valve }
315.00=             end
316.00=             else if OSC_COUNT [COUNT] < £4 then
317.00=             begin                              { pin contacted     }
318.00=                FWD_VALVE [COUNT] := £0;
319.00=                STABILISE_VALVE (COUNT,£0);
320.00=             end;
321.00=
322.00=          COUNT := COUNT + £1;
323.00=       end;
324.00=    end;
325.00=
326.00=    RECHARGE_RESERVOIR;
327.00=end;
328.00=
329.00=     { the main procedure checks if clamping was successful }
330.00=
331.00=
332.00={*************************************************************}
333.00={**                                                           **}
334.00={**  if PAIR_FLAG set the opposing pair of pins will be        **}
335.00={**  moved in unison                                          **}
336.00={**                                                           **}
337.00={*************************************************************}
338.00=
339.00=procedure MOVE_PINS_SLOW (START_PIN,END_PIN,PAIR_FLAG:byte);
340.00=
341.00=var PIN,FLAG : byte;
342.00=     ERROR : integer;
343.00=
344.00=begin
345.00=    write(£$0C); write(£$1A);
346.00=    writeln('procedure MOVE_PINS_SLOW');
347.00=    PIN:=START_PIN;
348.00=    UPDATE_LVDTS;
349.00=
350.00=    while PIN <= END_PIN do
351.00=    begin
352.00=       PULSE_FLAG [PIN]:=£0;
353.00=       OSC_COUNT  [PIN]:=£0;
354.00=       ERROR:=NOMINAL [PIN] - ord(LVDT [PIN]);
355.00=
356.00=       if ERROR > 0 then DIR_FLAG [PIN]:=£1    { moving fwd  }
357.00=                    else DIR_FLAG [PIN]:=£0;   { moving rev  }
358.00=       PIN:=PIN + £1;
359.00=    end;
```

```
360.00=
361.00=    FLAG:=£1;                                { a pin has moved      }
362.00=
363.00=    while FLAG = £1 do
364.00=    begin
365.00=       FLAG:=£0;
366.00=       PIN:=START_PIN;
367.00=       write(£$1A);
368.00=
369.00=       COUNT:=£0;
370.00=       while COUNT < £5 do
371.00=       begin
372.00=          writeln; COUNT := COUNT + £1;
373.00=       end;
374.00=
375.00=       while PIN <= END_PIN do          { loop around pins    }
376.00=       begin
377.00=          UPDATE_LVDTS;
378.00=          COUNTER := ord(LVDT [PIN]);
379.00=          TEMP    := integer(PIN);
380.00=          write('LVDT [',TEMP,'] :',COUNTER);
381.00=          if OSC_COUNT [PIN] < £4 then { pin not stable      }
382.00=          begin
383.00=             ERROR:=NOMINAL [PIN] - ord(LVDT [PIN] );
384.00=
385.00=                                  { pin not changed direction  }
386.00=
387.00=                if (abs(ERROR) > TOLERANCE) and
388.00=                   (OSC_COUNT [PIN] = £0)  then
389.00=                begin
390.00=                   if ERROR > 0 then
391.00=                                  begin
392.00=                                     PULSE_PIN_FWD(PIN,PAIR_FLAG);
393.00=                                     writeln('   FORWARD    ');
394.00=                                  end
395.00=                                  else
396.00=                                  begin
397.00=                                     PULSE_PIN_REV(PIN,PAIR_FLAG);
398.00=                                     writeln('   REVERSE    ');
399.00=                                  end;
400.00=
401.00=                   PULSE_FLAG [PIN] :=£1;   { pin has moved }
402.00=                   FLAG := £1;
403.00=                end
404.00=                else                      { pin is in tolerance   }
405.00=                begin
406.00=                   if PULSE_FLAG [PIN] =£1 then
407.00=                   begin
408.00=                      if OSC_COUNT [PIN] < £4 then
409.00=                                     STABILISE_VALVE (PIN,PAIR_FLAG)
410.00=                      FLAG := £1;
411.00=                      writeln('   STABILISING');
412.00=                   end;
413.00=                end;
414.00=          end;
415.00=          if OSC_COUNT [PIN] = £4 then writeln('                ');
416.00=          if PAIR_FLAG = £1 then PIN := PIN + £2
417.00=                            else PIN := PIN + £1;
418.00=       end;
419.00=    end;
```

```
420.00=end;
421.00=
422.00=
423.00={*************************************************************}
424.00={**                                                           **}
425.00={**  move a range of pins in reverse for a set distance       **}
426.00={**                                                           **}
427.00={*************************************************************}
428.00=
429.00=procedure BACKOFF_PINS (START_PIN,END_PIN:byte; BACKOFF:integer);
430.00=
431.00=var ERROR          : integer;
432.00=    NOMINAL        : array [£1..£10] of integer;
433.00=    PIN,FLAG       : byte;
434.00=
435.00=begin
436.00=   writeln('procedure BACKOFF called');
437.00=   PIN:=START_PIN;
438.00=
439.00=   while PIN <= END_PIN do
440.00=   begin
441.00=      NOMINAL [PIN] :=ord(LVDT [PIN]) - BACKOFF;
442.00=      if NOMINAL [PIN] < 0 then NOMINAL [PIN] := 5;
443.00=                                { prevents pin moving off LVDT }
444.00=      PULSE_FLAG [PIN] := £0;
445.00=      OSC_COUNT  [PIN] := £0;
446.00=      DIR_FLAG   [PIN] := £0;       { pin moving in reverse }
447.00=      PIN := PIN + £1;
448.00=   end;
449.00=
450.00=   FLAG:=£1;                          { a pin has moved       }
451.00=
452.00=   while FLAG = £1 do
453.00=   begin
454.00=      FLAG:=£0;
455.00=      PIN:=START_PIN;
456.00=
457.00=      while PIN <= END_PIN do          { loop around pins    }
458.00=      begin
459.00=         if OSC_COUNT [PIN] < £4 then { pin not stable      }
460.00=         begin
461.00=            UPDATE_LVDTS;
462.00=            ERROR:=NOMINAL [PIN] - ord(LVDT [PIN] );
463.00=
464.00=            if ERROR < 0 then  { pin not changed direction }
465.00=            begin
466.00=               if abs(ERROR) > TOLERANCE then
467.00=               begin
468.00=                  PULSE_PIN_REV(PIN,£0);
469.00=                  PULSE_FLAG [PIN] :=£1;   { pin has moved }
470.00=                  FLAG := £1;
471.00=               end
472.00=
473.00=               else                  { pin is in tolerance      }
474.00=               begin
475.00=                  if PULSE_FLAG [PIN] =£1 then
476.00=                  begin
477.00=                     if OSC_COUNT [PIN] < £4 then
478.00=                                    STABILISE_VALVE (PIN,£0);
479.00=                     FLAG := £1;
```

```
480.00=                    end;
481.00=                 end;
482.00=              end;
483.00=           end;
484.00=           PIN:=PIN + £1;
485.00=        end;
486.00=    end;
487.00=end;
488.00=
489.00=
490.00={*************************************************************}
491.00={**                                                         **}
492.00={**  bring pins to a safe load position                      **}
493.00={**                                                         **}
494.00={*************************************************************}
495.00=
496.00=procedure BRING_PINS_FWD;
497.00=
498.00=var ALL_PINS_IN_RANGE : boolean;
499.00=
500.00=begin
501.00=    COUNT := £1;
502.00=
503.00=    while COUNT < £11 do
504.00=    begin
505.00=       DIR_FLAG [COUNT] := £1;
506.00=       OSC_COUNT [COUNT] := £0;
507.00=       NOMINAL [COUNT] := LVDT_CAL [COUNT];
508.00=       COUNT := COUNT + £1;
509.00=    end;
510.00=
511.00=    ALL_PINS_IN_RANGE := false;
512.00=
513.00=    while ALL_PINS_IN_RANGE = false do
514.00=    begin
515.00=       ALL_PINS_IN_RANGE := true;
516.00=       UPDATE_LVDTS;
517.00=       COUNT:=£1;
518.00=
519.00=       while COUNT < £11 do
520.00=       begin
521.00=          if (LVDT [COUNT]<£11) and (OSC_COUNT [COUNT]=£0) then
522.00=          begin
523.00=             FWD_VALVE [COUNT] := £1;
524.00=             ALL_PINS_IN_RANGE := false;
525.00=          end
526.00=          else
527.00=          begin
528.00=             if OSC_COUNT [COUNT] < £4 then
529.00=             begin
530.00=                FWD_VALVE [COUNT] := £0;
531.00=                STABILISE_VALVE (COUNT,£0);
532.00=                ALL_PINS_IN_RANGE := false;
533.00=             end;
534.00=          end;
535.00=          COUNT := COUNT + £1;
536.00=       end;
537.00=    end;
538.00=
539.00=end;
```

```
540.00=
541.00=
542.00={*****************************************************************}
543.00={**                                                               **}
544.00={**                                                               **}
545.00={**                                                               **}
546.00={*****************************************************************}
547.00=
548.00=procedure POSITION_BLADE;
549.00=
550.00=var ERROR_FLAG, PIN : byte;
551.00=    ERROR          : integer;
552.00=
553.00=begin
554.00=   ERROR_FLAG:=£1;                          { pre-set flag          }
555.00=
556.00=   while ERROR_FLAG <> £0 do
557.00=   begin
558.00=      ERROR_FLAG:=£0;
559.00=      PIN:=£1;
560.00=
561.00=      while PIN < £6 do              { poll x axis datum pins    }
562.00=      begin
563.00=         ERROR:=NOMINAL [PIN] - ord(LVDT [PIN]);
564.00=
565.00=         if abs(ERROR) > 4 then ERROR_FLAG:=£1;
566.00=                     { if any pin is 5 or more LVDT counts  }
567.00=                     { away from the target position, the   }
568.00=                     { flag is set for the pins of the axis }
569.00=                     { to be moved                          }
570.00=
571.00=         PIN:=PIN + £2;
572.00=      end;
573.00=
574.00=      if ERROR_FLAG=£1 then          { move x axis pins      }
575.00=      begin
576.00=         writeln('Y AXIS PINS BACKING OFF');
577.00=         BACKOFF_PINS (£7,£10,32); { y axis pins           }
578.00=         MOVE_PINS_SLOW (£1,£6,£1);{ x axis pins           }
579.00=         CLAMP_BLADE (£7,£10);     { y axis pins           }
580.00=      end;
581.00=
582.00=      PIN:=£7;
583.00=
584.00=      while PIN < £11 do             { poll y axis datum pins    }
585.00=      begin
586.00=
587.00=         ERROR:=NOMINAL [PIN] - ord(LVDT [PIN]);
588.00=
589.00=         if abs(ERROR) > 4 then ERROR_FLAG:=£2;
590.00=
591.00=         PIN:=PIN + £2;
592.00=      end;
593.00=
594.00=      if ERROR_FLAG=£2 then          { move y axis pins      }
595.00=      begin
596.00=         writeln('X AXIS PINS BACKING OFF');
597.00=         BACKOFF_PINS (£1,£6,32);  { x axis pins           }
598.00=         MOVE_PINS_SLOW(£7,£10,£1);{ y axis pins           }
599.00=         CLAMP_BLADE (£1,£6);      { x axis pins           }
```

```
600.00=       end;
601.00=    end;
602.00=end;
603.00=
604.00=
605.00={*****************************************************************}
606.00={**                                                               **}
607.00={**                                                               **}
608.00={**                                                               **}
609.00={*****************************************************************}
610.00=
611.00=procedure GAUGE_BLADE;
612.00=
613.00=var ERROR : integer;
614.00=
615.00=begin
616.00=    COUNT:=£1;
617.00=
618.00=    while COUNT < £10 do
619.00=    begin
620.00=       ERROR:=ord(LVDT [COUNT]) - ord(LVDT [COUNT+£1]);
621.00=       ERROR:=ERROR - (LVDT_CAL [COUNT] - LVDT_CAL [COUNT+£1]);
622.00=       LVDT_GAUGE [COUNT] :=LVDT_CAL [COUNT] -(ERROR div 2);
623.00=
624.00=       if LVDT_GAUGE [COUNT] <  0  then LVDT_GAUGE [COUNT] := 5;
625.00=       if LVDT_GAUGE [COUNT] > 255 then LVDT_GAUGE [COUNT] := 250;
626.00=
627.00=       GAUGE_POS [COUNT] := ord(LVDT [COUNT]);
628.00=       GAUGE_POS [COUNT+£1] := ord(LVDT [COUNT+£1]);
629.00=
630.00=       COUNT:=COUNT + £2;
631.00=    end;
632.00=end;
633.00=
634.00=
635.00={*****************************************************************}
636.00={**                                                               **}
637.00={**                                                               **}
638.00={**                                                               **}
639.00={*****************************************************************}
640.00=
641.00=procedure MONITOR_PIN_POSITIONS;
642.00=
643.00=var ADDRESS : integer;
644.00=
645.00=begin
646.00=    SET_TIMER (4,60);              { set 60 ms interrupts        }
647.00=    COUNTER:=0;
648.00=
649.00=    while COUNTER < 501 do         { take 500 readings           }
650.00=    begin
651.00=
652.00=       if TIMEOUT_4 then           { interrupt - poll all pins   }
653.00=       begin
654.00=          UPDATE_LVDTS;
655.00=          COUNT:=£1;
656.00=
657.00=          while COUNT < £11 do
658.00=          begin
659.00=             ADDRESS:=500 * integer(COUNT - £1) + COUNTER;
```

```
660.00=                                  { calc position in array      }
661.00=
662.00=            PIN_POS [ADDRESS] :=ord(LVDT [COUNT]);
663.00=                                  { store pin pos in array      }
664.00=
665.00=            COUNT:=COUNT + £1;
666.00=          end;
667.00=
668.00=          SET_TIMER (4,60);
669.00=          COUNTER:=COUNTER + 1;
670.00=       end;                       { end interrupt loop          }
671.00=    end;                          { end of 500 readings loop    }
672.00=end;
673.00=
674.00=
675.00={*************************************************************}
676.00={**                                                         **}
677.00={**                                                         **}
678.00={**                                                         **}
679.00={*************************************************************}
680.00=
681.00=procedure PRINT_RESULTS;
682.00=
683.00=begin
684.00=    TERMINAL:=PRINTER_FLAG;
685.00=    PRINTER_FLAG:=$CCE4;          { redirect output to printer  }
686.00=
687.00=    ! JSR $CCCO ;
688.00=
689.00=    writeln('PIN  LVDT   CLAMP   NOM   GAUGE   LVDT    FINAL');
690.00=    writeln('NO.  CAL     POS    POS    POS    GAUGE    POS');
691.00=    writeln;
692.00=
693.00=    COUNT:=£1;
694.00=
695.00=    while COUNT < £11 do
696.00=    begin
697.00=       write(COUNT:3,'   ',LVDT_CAL [COUNT]:4,'    ');
698.00=       write(CLAMP_POS [COUNT]:3,'    ',NOM_POS [COUNT]:3,'    ');
699.00=       write(GAUGE_POS [COUNT]:3,'     ',LVDT_GAUGE [COUNT]:3);
700.00=       writeln('     ',FINAL_POS [COUNT]:3);
701.00=
702.00=       COUNT:=COUNT+£1;
703.00=    end;
704.00=
705.00=    PRINTER_FLAG:=TERMINAL;    { redirect output to screen   }
706.00=end;
707.00=
708.00=
709.00={*************************************************************}
710.00={**                                                         **}
711.00={**                                                         **}
712.00={**                                                         **}
713.00={*************************************************************}
714.00=
715.00=procedure SET_VALVE_PULSE_LENGTHS;
716.00=
717.00=var FLAG                    : byte;
718.00=    TEMPP, START, ERROR : integer;
719.00=
```

```
720.00=begin
721.00=    write(£$0C); write(£$1A);
722.00=    write('procedure SET VALVE PULSE LENGTHS');
723.00=
724.00=    BRING_PINS_FWD;
725.00=
726.00=    writeln(£$0C); write(£$1A); writeln; writeln;
727.00=    writeln('SET PULSE LENGTH FOR FORWARD VALVES'); writeln;
728.00=
729.00=    COUNT := £1;
730.00=    while COUNT < £11 do
731.00=    begin
732.00=       FLAG := £1;
733.00=
734.00=       while FLAG = £1 do
735.00=       begin
736.00=          UPDATE_LVDTS; FLAG := £0;
737.00=
738.00=          if LVDT [COUNT] < £20 then
739.00=          begin
740.00=             while LVDT [COUNT] < £20 do
741.00=             begin
742.00=                FWD_VALVE [COUNT] := £1; UPDATE_LVDTS;
743.00=             end;
744.00=          end;
745.00=          PULSE_PIN_REV (COUNT,£0);
746.00=
747.00=          COUNTER := ord(LVDT [COUNT]);
748.00=          TEMP := integer(COUNT);
749.00=          TEMPP := PULSE_LENGTH_FWD [COUNT];
750.00=          write(CR,'LVDT [',TEMP,'] :',COUNTER,TEMPP);
751.00=
752.00=          START := ord(LVDT [COUNT]); { store initial position }
753.00=          PULSE_PIN_FWD(COUNT,£0);
754.00=
755.00=          SET_TIMER(4,1500);
756.00=          while TIMEOUT_4 = false do;    { wait for pin to move }
757.00=
758.00=          UPDATE_LVDTS;
759.00=
760.00=          ERROR := ord(LVDT [COUNT]) - START;
761.00=
762.00=          if ERROR > 10 then
763.00=          begin
764.00=             PULSE_LENGTH_FWD[COUNT]:=PULSE_LENGTH_FWD [COUNT]-1;
765.00=
766.00=             if PULSE_LENGTH_FWD [COUNT] < 5 then
767.00=                PULSE_LENGTH_FWD [COUNT] := 5;
768.00=
769.00=             FLAG := £1;
770.00=          end;
771.00=
772.00=          if ERROR < 4  then
773.00=          begin
774.00=             PULSE_LENGTH_FWD[COUNT]:=PULSE_LENGTH_FWD [COUNT]+1;
775.00=             FLAG := £1;
776.00=          end;
777.00=
778.00=          write('    ERROR :',ERROR);
779.00=
```

```
780.00=          if LVDT [COUNT] > £200 then
781.00=          begin
782.00=             while LVDT [COUNT] > £50 do
783.00=             begin
784.00=                REV_VALVE [COUNT] := £1; UPDATE_LVDTS;
785.00=             end;
786.00=          end;
787.00=          PULSE_PIN_FWD (COUNT,£0);
788.00=       end;
789.00=       writeln;
790.00=       COUNT := COUNT + £1;
791.00=    end;
792.00=
793.00=    write(£$0C); write(£$1A); writeln; writeln;
794.00=    writeln('SET PULSE LENGTH FOR REVERSE VALVES'); writeln;
795.00=
796.00=    COUNT := £1;
797.00=    while COUNT < £11 do
798.00=    begin
799.00=       FLAG := £1;
800.00=
801.00=       while FLAG = £1 do
802.00=       begin
803.00=          UPDATE_LVDTS; FLAG := £0;
804.00=
805.00=          if LVDT [COUNT] > £230 then
806.00=          begin
807.00=             while LVDT [COUNT] > £230 do
808.00=             begin
809.00=                REV_VALVE [COUNT] := £1; UPDATE_LVDTS;
810.00=             end;
811.00=          end;
812.00=          PULSE_PIN_FWD (COUNT,£0);
813.00=
814.00=          COUNTER := ord(LVDT [COUNT]);
815.00=          TEMP := integer(COUNT);
816.00=          TEMPP := PULSE_LENGTH_REV [COUNT];
817.00=          write(CR,'LVDT [',TEMP,'] :',COUNTER,TEMPP);
818.00=
819.00=          START := ord(LVDT [COUNT]); { store initial position }
820.00=          PULSE_PIN_REV(COUNT,£0);
821.00=
822.00=          SET_TIMER(4,1500);
823.00=          while TIMEOUT_4 = false do;    { wait for pin to move }
824.00=
825.00=          UPDATE_LVDTS;
826.00=
827.00=          ERROR := START - ord(LVDT [COUNT]);
828.00=
829.00=          if ERROR > 10 then
830.00=          begin
831.00=             PULSE_LENGTH_REV[COUNT]:=PULSE_LENGTH_REV[COUNT]-1;
832.00=
833.00=             if PULSE_LENGTH_REV [COUNT] < 5 then
834.00=                PULSE_LENGTH_REV [COUNT] := 5;
835.00=
836.00=             FLAG := £1;
837.00=          end;
838.00=
839.00=          if ERROR < 4  then
```

```
840.00=          begin
841.00=             PULSE_LENGTH_REV[COUNT]:=PULSE_LENGTH_REV[COUNT]+1;
842.00=             FLAG := £1;
843.00=          end;
844.00=
845.00=          write('   ERROR :',ERROR);
846.00=
847.00=          if LVDT [COUNT] < £20 then
848.00=          begin
849.00=             while LVDT [COUNT] < £170 do
850.00=             begin
851.00=                FWD_VALVE [COUNT] := £1; UPDATE_LVDTS;
852.00=             end;
853.00=          end;
854.00=          PULSE_PIN_REV (COUNT,£0);
855.00=       end;
856.00=       writeln;
857.00=       COUNT := COUNT + £1;
858.00=    end;
859.00=
860.00=    TERMINAL:=PRINTER_FLAG;
861.00=    PRINTER_FLAG:=$CCE4;       { redirect output to printer  }
862.00=
863.00=    ! JSR $CCCO ;
864.00=
865.00=    writeln;
866.00=    writeln('FORWARD & REVERSE VALVE PULSE LENGTHS');
867.00=
868.00=    COUNT := £1;
869.00=    while COUNT < £11 do
870.00=    begin
871.00=       COUNTER := integer(COUNT);
872.00=       write('PIN :',COUNTER,'   FORWARD :');
873.00=       write(PULSE_LENGTH_FWD [COUNT],'   REVERSE :');
874.00=       writeln(PULSE_LENGTH_REV [COUNT]);
875.00=       COUNT := COUNT + £1;
876.00=    end;
877.00=
878.00=    PRINTER_FLAG := TERMINAL;
879.00=end;
880.00=
881.00=
882.00={*************************************************************}
883.00={**                                                         **}
884.00={**                  main procedure                         **}
885.00={**                                                         **}
886.00={*************************************************************}
887.00=
888.00=begin
889.00=    WBYTE($E785,£1);                { enable keyboard scan      }
890.00=    HOUSE_KEEPING;
891.00=    SET_VALVE_PULSE_LENGTHS;
892.00=    RETRACT_ALL_PINS;
893.00=    TOLERANCE := 5;
894.00=    CHAC := £0;
895.00=
896.00=    while CHAC <> £$0D do          { continue until RETURN    }
897.00=    begin
898.00=       write (£$0C); write (£$1A);  { clear screen / cursor }
899.00=       writeln('CYCLE STARTED : WAITING FOR BLADE');
```

```
900.00=
901.00=       while BLADE_WAITING=false do;      { wait for input    }
902.00=       writeln('Blade waiting to load');
903.00=
904.00=       BRING_PINS_FWD;
905.00=       MOVE_PINS_SLOW (£1,£10,£0);  { all pins individually }
906.00=       BACKOFF_PINS (£1,£10,64);
907.00=       writeln('SAFE TO LOAD BLADE');
908.00=
909.00=       while BLADE_LOADED=false do;       { wait for input    }
910.00=       writeln('Blade loaded');
911.00=
912.00=       CLAMP_BLADE (£1,£10);              { all pins          }
913.00=
914.00=       UPDATE_LVDTS;
915.00=       COUNT:=£1;
916.00=
917.00=       while COUNT < £11 do     { store initial positions   }
918.00=       begin
919.00=          CLAMP_POS [COUNT] := ord(LVDT [COUNT]);
920.00=          COUNT:=COUNT + £1;
921.00=       end;
922.00=
923.00=       if ALL_PINS_CONTACTED = true then
924.00=       begin
925.00=          writeln; writeln('BLADE CLAMPED : ROBOT MAY WITHDRAW');
926.00=          ROBOT_MAY_WITHDRAW := true;
927.00=
928.00=          while ROBOT_WITHDRAWN = false do;{ wait for input }
929.00=          writeln('Robot withdrawn');
930.00=
931.00=          COUNT:=£1;
932.00=
933.00=          while COUNT < £11 do     { set target values as    }
934.00=                                   { lvdt calibration values }
935.00=          begin
936.00=             NOMINAL [COUNT] := LVDT_CAL [COUNT];
937.00=             COUNT:=COUNT + £1;
938.00=          end;
939.00=
940.00=          POSITION_BLADE;
941.00=          writeln('BLADE AT NOMINAL POSITION');
942.00=
943.00=          if ALL_PINS_CONTACTED=true then
944.00=          begin
945.00=             UPDATE_LVDTS;
946.00=             COUNT:=£1;
947.00=
948.00=             while COUNT < £11 do  { store pin positions at }
949.00=                                   { nominal blade position }
950.00=             begin
951.00=                NOM_POS [COUNT] := ord(LVDT [COUNT]);
952.00=                COUNT:=COUNT+£1;
953.00=             end;
954.00=
955.00=             GAUGE_BLADE;
956.00=             writeln('BLADE GAUGED');
957.00=
958.00=             COUNT:=£1;
959.00=
```

```
960.00=              while COUNT < £11 do     { set target values as }
961.00=                                       { gauged blade values  }
962.00=              begin
963.00=                 NOMINAL [COUNT] := LVDT_GAUGE [COUNT];
964.00=                 COUNT:=COUNT + £1;
965.00=              end;
966.00=
967.00=              POSITION_BLADE;            { to gauged position   }
968.00=              writeln('BLADE AT OFFSET POSITION');
969.00=
970.00=              if ALL_PINS_CONTACTED=true then
971.00=              begin
972.00=                 UPDATE_LVDTS;
973.00=                 COUNT:=£1;
974.00=
975.00=                 while COUNT < £11 do  { store pin positions }
976.00=                                       {at final blade position }
977.00=                 begin
978.00=                    FINAL_POS [COUNT] := ord(LVDT [COUNT]);
979.00=                    COUNT:=COUNT+£1;
980.00=                 end;
981.00=
982.00=                 while POURING_IN_PROGRESS=false do;
983.00=
984.00=                 MONITOR_PIN_POSITIONS;
985.00=
986.00=                 RETRACT_ALL_PINS;
987.00=                 writeln('SAFE TO UNLOAD DIE');
988.00=
989.00=                 PRINT_RESULTS;
990.00=
991.00=              end;
992.00=           end;
993.00=        end;
994.00=
995.00=        if ALL_PINS_CONTACTED=false then { blade not clamped }
996.00=        begin
997.00=           RETRACT_ALL_PINS;
998.00=           BLADE_NOT_CLAMPED := true;
999.00=           writeln('MIS-LOAD : CYCLE ABANDONED - PRESS RETURN');
1000.00=          CHAC := £0;
1001.00=          while CHAC <> £$0D do
1002.00=          if STATUS then CHAC:=INKEY;
1003.00=          CHAC := £0;                { clear for main loop    }
1004.00=       end;
1005.00=
1006.00=    end;                             { loop to start again    }
1007.00=
1008.00=end.
```

REFERENCES

[1] Rolls-Royce Limited "The Jet Engine" 1969

[2] WELLER J.A., "Diecasting Beats Blade Machining Problem" Canadian Machinery and Metalworking Feb. 1971

[3] EDIGER D., "Broach Fixturing by Encapsulation" Am. Machinist pg.130 Nov. 1975

[4] GRAHAM E., " A New Way to Hold Turbine Blades During Machining" Manufacturing Engineering pp.89-90 Feb. 1982

[5] HAYWARD H.T., " An Investigation of the Encapsulation Process for Machining Turbine Blades" PhD Thesis Lanchester Polytechnic pp.7-27, 1983

[6] "Grinding Turbine Blades in Automated Cells", Manufacturing Engineering pp.63-64 Dec. 1983

[7] ASTROP A., "Seven-Cell Creep Feed Grinding Line" Mach. and Prod. Engineering pg.29 Jan. 1983

[8] "Creep Feed Grinding Automated" CME pg.3 Jan. 1983

[9] ABRAHAM R.G., STEWART R.J.S., SHUM L.Y.,"State-of-the-art in Adaptable-Programmable Assembly Systems" pp. 6-8,87-96 IFS Ltd. 1972

[10] WATSON P.C., DRAKE S.H.,"Pedestal and Wrist Force Sensors for Automatic Assembly" Proc. 5th Int. Symp. on Ind. Robots pp.501-511 SME USA 1975

[11] NEVINS J.L., WHITNEY D.E.,"The Force Vector Assembler Concept" First CISM-IFToMM Symp. on Theory and Practice of Robots and Manipulators Vol.II pp. 273-288 Springer-Verlag 1973

[12] SIMUNOVIC S.,"Force Information in Assembly Processes" 5th Int. Symp on Ind. Robots pp. 415-431 SME USA 1975

[13] VAN BRUSSEL H., SIMONS J.,"Automatic Assembly by Active Force Feedback Accomodation" Proc. 4th Int. Conf. on Ind. Robot Technology Vol I pp. 181-193 IFS Ltd. 1978

[14] GOTO T., INOYAMA.T. TAKEYASU K.,"Precise Insert Operation by Tactile Controlled Robot" Proc. 2nd Conf. on Ind. Robot Technology pp. 209-218 IFS Ltd. 1974

[15] DRAKE S., WATSON P., SIMUNOVIC S., "High Speed Robot Assembly of Precision Parts using Compliance instead of Sensory Feedback" Proc. 7th Int. Symp. on Ind. Robots pp. 87-98 JIRA 1977

[16] VAN BRUSSEL H., SIMONS J.,"The Adaptable Compliance Concept and its use for Automatic Assembly by Active Force Feedback Accommodation" Proc. 9th Int. Symp. on Ind. Robots pp. 167-181 SME USA 1979

[17] VAN BRUSSEL H., SIMONS J.,"A Self-Learning Robot for Automatic Assembly" Proc. 1st. Int. Conf. on Assembly Automation pp. 295-309 IFS Ltd. 1980

[18] VAN BRUSSEL H., SIMONS J.,"Adaptive Assembly" 4th BRA Annual Conf. pp. 95-106 IFS Ltd. 1981

[19] WHITNEY D.E., NEVINS J.L., "What is the Remote Centre Compliance (RCC)and what can it do ?" Proc. 9th Int. Symp. on Ind. Robots pp. 135-152 SME USA 1979

[20] STOYANOV B., IVANOV R.,"Experimental Research and Analysis of RCC Device" Proc. 4th Int. Conf. on Assembly Automation pp.282-291 IFS Ltd. 1983

[21] HIRZINGER G., BRUNET U.,"Fast and Self-Improving Compliance using Digital Force-Torque-Control" Proc. 4th Int. Conf. on Assembly Automation pp. 268-281 IFS Ltd. 1983

[22] SIMUNOVIC S.N., "Parts Mating Theory for Robot Assembly" Proc. 9th Int. Symp. on Ind. Robots pp. 183-193 SME 1979

[23] BURCKHARDT C.W., HELMS D., " Some General Rules for Building Robots" Proc. 3rd. Conf. on Ind. Robot Tech., pp.49-58 IFS 1976

[24] DEMAUREX M.O., GERELLE E.G.R., "Can I Build this Robot ?" Proc. 9th Int. Symp. on Ind. Robots pp. 621-639 SME 1979

[25] BURCKHARDT C.W., GERELLE E.G,R., "Dynamic Design Parameters for Robot Arms: Experimental Results" Proc 10th Int. Symp. on Ind. Robots pp. 321-329 1980

[26] BOULDEN L.L., "Hydraulics, Pneumatics and Dollars" Machine Design 43 Feb 18 pp.104-110 1971

[27] METZGER J., "Which Fluid Power System - Hydraulic or Pneumatic" Machine Design 41 June 26 pp. 126-129 1969

[28] ARCHER J.R., BLENKINSOP P.T., "Actuation for Industrial Robots" Proc. Conf. "Electric vs. Hydraulic Drives" pp. 85-89 IMechE 1983

[29] SALMON M., "Consideration of the Design of the Olivetti Sigma: An Industrial Robot for the Manufacturing Industries" Proc. 2nd. CISM-IFTOMM Symp. on Ind. Robots pp.93-109 Elsevier Scientific Publ. Co. 1976

[30] INAGAKI.S., "A Discussion on Positioning Accuracy on Industrial Robots" Proc. 9th Int. Symp. on Ind. Robots pp. 679-690 SME 1979

[31] HO C.S., "Error Analysis for Robot Assembly Systems" Proc. 4th. Int Conf. on Assembly Automation pp. 243-254 IFS 1983

[32] MORGAN C., "The Rationalisation of Robot Testing" Proc. 10th. Int. Symp. on Ind. Robots pp. 399-406 1980

[33] McENTIRE R.H., "Three Dimensional Accuracy Measurement Methods for Robots" The Industrial Robot pp.105-111 Sept. 1976

[34] GEORGE R.F., "Evaluating Bearings for Robots", Machine Design April 7 pp. 79-83 1983

[35] ICI Victrex Polyethersulphone Technical Service Note VX 16.82, ICI Petrochemicals and Plastics Division, Wilton

[36] SKF Ltd. Dry Sliding Bearings Type A

[37] TESCHLER L., " Advances in Sensor Technology" Machine Design May 20 1982

[38] ORMISTON P.T., "Measuring Displacement with LVDT Transducers" Electronic Engineering pp. 69 - 71 Vol.50 No. 608 June 1978

[39] HAYWARD H.T., "An Investigation of the Encapsulation Process for Machining Turbine Blades" PhD Thesis Lanchester Polytechnic pp. 106 - 109 1983

BIBLIOGRAPHY

McCALLUM R., "Casting Critical Components" Metallurgia pp. 524-533 Oct 1982

FOSTER C.W., "Precision Casting for Gas Turbine Engines" Foundry Trade Journal pp. 175-189 July 1982

ELECTRO-CRAFT CORPORATION, "DC Motors, Speed Control and Servo-Systems" 1980

NEVINS J.L., WHITNEY D.E., "Adaptable-Programmable Assembly Systems: an Information and Control Problem" Proc. 5th Int. Symp. on Ind. Robots pp. 387-406 SME 1975

DRAZAN P.J., JEFFREY M.F., "Some Aspects of an Electro-Pneumatic Industrial Manipulator" Proc. 4th Int. Conf. on Ind. Robot Tech. Vol I pp. 396-405 IFS 1978

DRAZAN P.J., JEFFREY M.F., ZAREK J.M., "Recent Developments in the Design of a Novel Electro-Pneumatic Robot" Proc. 2nd. Int. CISM-IFTOMM Symp. pp. 143-150 Elsevier Scientific Publ. Co. Sept 1976