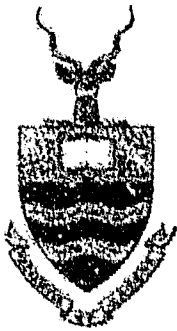


DEVELOPMENT OF A TOOL CHANGING ROBOT FOR A COMPUTER NUMERICALLY CONTROLLED MILLING MACHINE



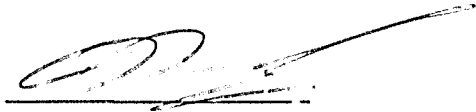
Craig David Read

**A research report submitted to the Faculty of Engineering, University of the
Witwatersrand, in partial fulfillment of the degree of Master of Science in
Engineering**

Johannesburg, 1999

DECLARATION

I declare that this report is my own, unaided work. It is being submitted for the degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.



(Signature of Candidate)

12th day of October (year) 1999

ABSTRACT

The Industrial Engineering department at the University of the Witwatersrand is currently developing a Flexible Manufacturing Cell (FMC) for local application. One of the components of this FMC is a Computer Numerically Controlled (CNC) Milling Machine, for which a tool change robot was designed and built in 1991. The project was never completed and over the years the condition of the robot deteriorated and parts of the control system were damaged or lost.

This report documents the redevelopment of the tool change robot. The control system has been modified and suitable replacement components incorporated into the design. An external interface with the CNC Milling Machine has been developed, and the software has been completely rewritten in a modular manner for increased functionality. In its current state, the Robot is capable performing an automated tool change.

CONTENTS

Page

TITLE PAGE	I
DECLARATION	II
ABSTRACT	III
LIST OF FIGURES	VI
LIST OF TABLES	VII
NOMENCLATURE	VIII
1 INTRODUCTION	1
1.1 INTRODUCTION TO ROBOTICS	1
1.2 THE MECHANICAL STRUCTURE OF ROBOTS	2
1.3 ROBOTIC CONTROL SYSTEMS	2
1.3.1 Choice of Controller	4
1.4 SENSORS AND ACTUATORS	5
1.4.1 Stepping Motors	6
2 HISTORY OF THE TOOL CHANGE ROBOT	8
3 STATEMENT OF THE PROBLEM	10
3.1 STATEMENT OF THE SUB-PROBLEMS	10
3.1.1 Mechanical Design	10
3.1.2 Control System	10
3.1.3 Interface to The Milling Machine	11
3.2 DELIMITATION'S	11
4 EXISTING EQUIPMENT	12
4.1 PC CONTROLLER	12
4.2 STEPPER MOTOR AND CONTROLLER	12
4.3 RELAY CONTROL CARD	12
4.4 MECHANICAL STRUCTURE	13
4.5 INSTRUMENTATION	13
5 THE ROBOT CONTROL SYSTEM	14
5.1 THE ORIGINAL CONTROL SYSTEM	14
5.2 CHOICE OF CONTROLLER	15
5.3 OVERVIEW OF CONTROL SYSTEM	16

5.4	INTERFACE WITH THE CNC MILL	23
5.5	REPLACEMENT OF SENSORS	24
5.6	WIRING OF THE MAIN CONTROL UNIT	25
6	THE SOFTWARE	30
6.1	OVERVIEW OF THE SOFTWARE	30
6.2	OPERATING THE SOFTWARE	32
6.3	DETAILED DESCRIPTION OF SOFTWARE PROCEDURES	32
7	MECHANICAL MODIFICATIONS	39
8	CONCLUSION	43
9	FURTHER WORK REQUIRED	46
10	REFERENCES	47
APPENDIX A		49
APPENDIX B		83
APPENDIX C		92

LIST OF FIGURES

FIGURE		PAGE
1.1	Open Loop Control	3
1.2	Closed Loop Control	3
1.3 (a)	Rotor of a Stepping Motor	6
1.3 (b)	Energizing Sequence of a Four-Pole Stepping Motor	7
5.1	Overview of the Original Control System	14
5.2	Overview of the Implemented Control System	17
5.3	PCL-838 37 Pin Assignment	18
5.4	Pin Assignment for Main Controller 37-Pin Connector	20
5.5 (a)	Top View of the Main Control Unit	21
5.5 (b)	Main Control Unit	22
5.6	Wiring of the Stepping Motor Driver	26
5.7	PC-38 Computer Relay Adapter Card	27
5.8	Wiring Diagram for 24V Relay Card	28
5.9	Wiring Diagram for (9-Pin) Mill Interface	29
5.10	Wiring Diagram for Solenoid Actuators	29
5.11	Wiring Diagram for Sensor Connector	29
6.1	Flow Diagram for Main Software Sequence	38
7.1	Overview of Robot showing new Support Frame	40
7.2	New Tool Holder Support Frame	41
7.3	New Robot Support Stand	42
8.1	Overview of Robot (photograph)	45
8.2	Robot Arm Extended to Retrieve Tool from Mill	45

FIGURE		PAGE
B1	Overview of Robotic Arm with Original Frame	84
B2	Robotic Arm Gripper Assembly	85
B3	Protractor & Indicator Details	86
B4	Tool Holder	87
B5	Tool Locator Details	88
B6	Main Support Frame	89
B7	Vertical & Horizontal Guides	90
B8	Vertical Guide Arm	91

LIST OF TABLES

TABLE		PAGE
5.1	Pin Assignments for PCL-838 Card	19
6.1	Robot Positions Corresponding to Binary Outputs	37

NOMENCLATURE

AC	Alternating Current
CN	Computer Numerical Control
DC	Direct Current
DCS	Distributed Control System
FMC	Flexible Manufacturing Cell
I / O	Input / Output
OOP	Object Orientated Programming
PC	Personal Computer
PLC	Programmable Logic Controller
TSR	Terminate and Stay Resident software

1 INTRODUCTION

1.1 Introduction to Robotics

From the moment people started doing work, they began to find methods of automating the work. However, the term *robot* and *robotics* are of recent origin, and were first used by a Czechoslovakian dramatist, Karel Capek in his 1921 play (Fuller 1991). These robots were designed to perform perfect and tireless manual labour.

The industrial robot has been developed to assist people with manual work. Yet the term robot is commonly misunderstood, and often confused with terms such as automation. Even within the industrial world, there is no single definition for a robot. The Japanese have defined a robot as an all purpose device equipped with a memory device and a terminal, and capable of rotation and of replacing human labour by automatic performance of movement (Fuller 1991). However, this definition could include numerous automated devices such as bottling machines, or even a modern washing machine.

The Robotic Association of America defines a robot as *'a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialised devices, through variable programmable motions, for the purpose of a variety of tasks'*. Modern industrial robots are commonly used for repetitive, unskilled or semiskilled, monotonous and burdensome tasks, as well as jobs that would be hazardous for a person to perform. They are distinguished from *hard automation* due to them not performing a single specific task.

While industrial robots and automated machines are usually treated as separate topics, most industrial robots work in co-operation with other

automated machines. Therefore any robot must interface with the environment in order to obtain information regarding the task to be performed.

The study of robotics covers many disciplines: mechanical, electrical electronic, computing and control engineering. There are severe constraints on any robot design, most notably that the robot must be capable of performing useful tasks in real time, at an affordable cost.

1.2 The Mechanical Structure of Robots

A robot must have a mechanical structure to move an object around, and also an end effector to hold a tool or grip a part. Typically the mechanical structure is constructed from a series of rigid links, connected by joints. The particular type of joint defines how one link can move relative to the other. The two most common type of joints are *prismatic* and *revolute*. A prismatic joint, also known as a sliding joint, allows one link to move in a straight line relative to another. A revolute joint takes the form of a hinge between one joint and another. In practice there are a vast number of configurations used.

A major consideration in the design of a robot is to keep the links as light as possible to reduce the response time and the force required to move them, while keeping them as rigid as possible.

1.3 Robotic Control Systems

For any specific mechanical robotic structure, it will be necessary to control the movements of each part of the mechanism in order that the end effector can be moved to a required position and orientation in space. Broadly

speaking, two types of control can be used, namely Open Loop control and Closed loop control.

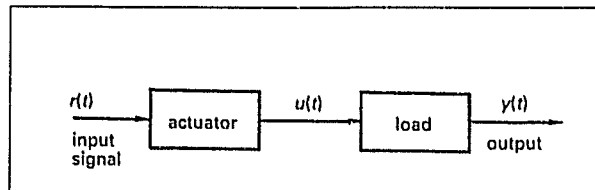


FIGURE 1.1 OPEN LOOP CONTROL

Open control is shown schematically in Figure 1.1. No feedback or measure of the output position is taken, so this control relies wholly on good calibration and the reliable behaviour of the actuator system.

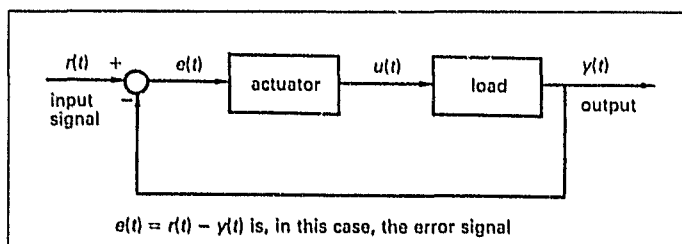


FIGURE 1.2 CLOSED LOOP CONTROL

Closed Loop control is depicted in Figure 1.2. As shown, there is a feedback signal indicating the current position, which is used as a difference error to continuously correct the driving signal to achieve accurate control.

1.3.1 Choice of Controller

The choice of the correct controller for automation applications has long been the subject of debate. Traditionally, a PLC was considered the only option for plant automation due to their rugged construction, dedicated control functionality and lower cost. A PC was only considered suitable for office applications. However in more recent years, the PC has gained much ground in this field. This is largely the result of high power PC's becoming extremely economical in the highly competitive environment, and the relative simplicity and power of modern programming interfaces dedicated to control software.

"Why PC's won't kill PLC's" (Hohmann, 1987) reviews the situation, and concludes that for the majority of traditional industrial applications, the PLC is still better suited and more economical for automation and control. This is largely due to the dedicated architecture of a PLC which is specifically designed for both analogue and digital I/O, while PC's require special control interfaces.

However, provided that the environment is not particularly harsh, the PC still has the advantage of greater flexibility and programmability, and therefore functionality. These devices can be programmed in a number of standard commercial programming languages such as C, C++, Pascal or Forth. Furthermore, 'Soft' PLC software allows a PC to be programmed and function like a PLC, giving the benefit of both systems.

"Take advantage of control options" (Barwise, Moodley 1998) also reviews various control options, including PLC's, PC's and DCS'. The choice of controller is ultimately dependent on the particular application, and the specific requirements such as level of desired discrete or process function, operational speed, electrical power requirements, the environment, flexibility for future changes, and the budget. One important consideration

raised' is the rapid technological advancement in the PC industry, which renders a PC redundant within a few years of purchase. This problem does not exist in the relatively stable PLC industry.

As already discussed, there have been tremendous advances in application software for industrial automation. The current trend in software development is Object Orientated Programming (OOP). The goal of this type of programming is to save time through developing objects that are re-usable, thus allowing software developers to create libraries of functions and procedures. These objects differ from traditional functions and procedures in one fundamental manner, they contain data in addition to functions that operate on that data. Most common modern programming languages such as Turbo Pascal 5.5 or higher, and C++ support object orientated programming.

1.4 Sensors and Actuators

In order to move the mechanical structure of a robot, actuators are required. There are a number of alternate ways of actuating a mechanical link, which broadly speaking can be divided into pneumatic, hydraulic and electric. All of these can give either rotary or linear motion.

Typically pneumatic actuators comprise a cylinder and valves which control the flow of fluid to it from a high pressure supply. There is an extensive range of standard actuators available to select from, depending on the nature of the movement, the distance to be moved, and the force required, which is a function of the pressure and diameter of the cylinder.

Electric actuators are typically electric motors, operating using either AC and DC supplies. Numerous specialised types are available including *servo drives* and *stepper motors*.

1.4.1 Stepping Motors

The main feature of a stepping motor is that for a given sequences of pulses or steps, the output shaft of the motor will rotate an angle directly proportional to the number of pulses. The rotor comprises a permanent magnet with its axis lying along the axis of rotation of the rotor. Each end of the magnet is shaped like a toothed wheel, with the North-pole teeth radially displaced one-half tooth pitch from the South-pole teeth, as is shown in Figure 1.3 (a).

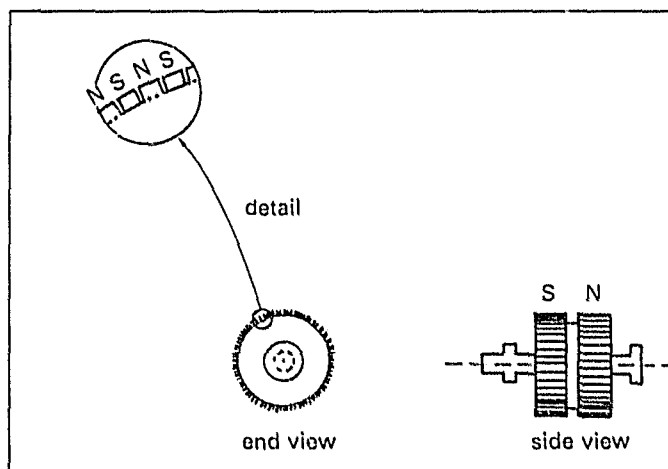


FIGURE 1.3(a) ROTOR OF A STEPPING MOTOR

Multiple stator windings are distributed around the periphery. There are energised electrically. An energising sequence for a four-phase stepping motor is shown in Figure 1.3 (b – e).

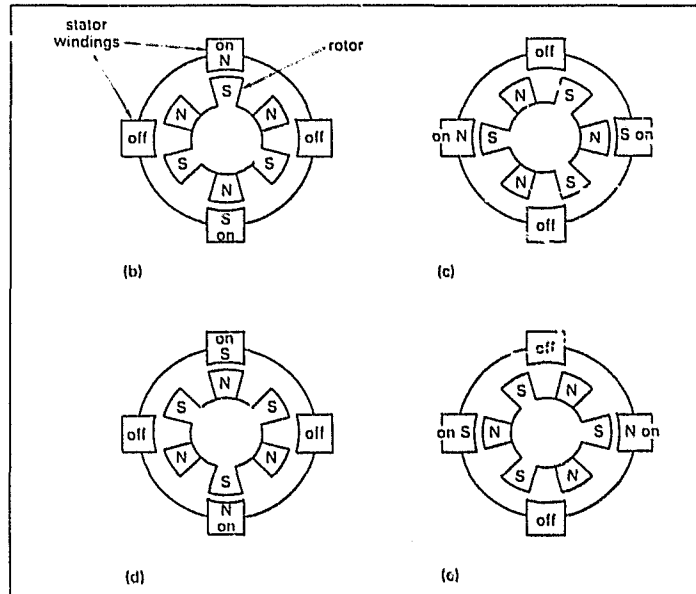


FIGURE 1.3(b-e) ENERGISING SEQUENCE OF FOUR-POLE STEPPING MOTOR

As the windings are sequentially energised, the rotor will be displaced a half-tooth radially in order to reach a stable state. For each successive positive or negative step the motor will rotate a half-tooth radially.

A useful feature is the static holding torque in each stable state, which will tend to hold the rotor in the correct position even when an external torque is applied. The number of teeth and stator windings determines the resolution of the motor, or step size.

If an accurate pulse count is kept, then the angular position of the rotor is known with respect to it's position at the start of the count, and the stepping motor can be operated open loop, so positional feedback sensors (such as rotary encoders) are not required. For this reason stepping motors are popular in applications where cost is an important constraint.

2 HISTORY OF THE TOOL CHANGE ROBOT

The Industrial Engineering Department at the University of the Witwatersrand is in the process of developing a Flexible Manufacturing Cell (FMC) in the Mechanical Engineering Laboratory. One component required for this FMC is an automatic tool-changing device to change the tools on a CNC controlled Maho 700 Milling Machine.

A.J. Speedie first undertook this project in 1988 as an MSc project. At this time a Dainichi-Skyes PT600 industrial robot was selected for the application. In 1991, it was decided that this robot was no longer suitable as it was outdated and spares were no longer readily available.

Hence the project was taken up again by M. F. Smargiasso (Smargiasso 1991) as fourth year research and design projects. Smargiasso further developed one of the designs which had been proposed by A. J. Speedie. This design used a PC based control system, and incorporated a stepper motor for rotation of the gripper and pneumatic cylinders for vertical and horizontal movement.

Smargiasso completed the design of the robot and built and demonstrated it. Subsequently it was decided to modify this design, and this task was given to third year groups each year from 1992 to 1994 for completion. This included interfacing the robot to the milling machine, building a permanent base, and correcting problems with the software. However none of these groups were able to meet the requirements.

During this period, numerous electronic components of the robot were damaged or lost. Damage of unknown origin to the stepper motor driver occurred which has been diagnosed by the supplier to have resulted from incorrect prior repairs to the drive unit, causing circuit damage. It is unclear when this occurred, as the last documented work undertaken on the robot was by a third year group in 1994. In

this report, photographs show the driver complete with an original motherboard. However at the commencement of this project, the original motherboard had been replaced with a simple home-made unit. It is also unclear if the current card is the original used for the robot. Smargiasso (1991) documents the drive to be type SD-5, while the damaged drive is type SD-2. The original relay control card was also damaged in 1994. The wiring of the robot was also damaged and there was no tagging of wires or circuit diagrams. In this condition, the robot was completely inoperable.

3 STATEMENT OF THE PROBLEM

The objective of this research was to redevelop the existing tool changing robot in the Industrial Engineering Laboratory in order for it to be capable of executing a completely automated tool change on the CNC Milling machine.

Specific areas of development were to design an interface with the existing CNC Mill, to improve the control software to make it more accurate and flexible, and it design a suitable mounting for the robot to position it correctly relative to the Mill.

3.1 Statement of the Sub-problems

3.1.1 *Mechanical Design*

Mount the robot in a suitable position to achieve a tool change on the milling machine. The existing frame has not been designed to accommodate this, and modifications to the frame and possibly the robot may be necessary.

Review the complete mechanical design of the robot and tool holder to ensure it is suitable for the purpose.

3.1.2 *Control System*

The complete control system for the robot, including both hardware and software, must be redesigned and rebuilt for the following reasons:

- The original design has never met the design specification in terms of accuracy and control.

- Damage to the original hardware requires it be replaced, not necessarily with the same parts, requiring different control.
- The existing software has been developed in Turbo Pascal, using a simple command interpreter, 'ADVICE.SYS' to control the stepper motor. This is an inefficient programming technique.
- The design of the software should be as modular and flexible as possible.
- The existing control system has no external interface, which is required for the robot to become a completely automated component of the FMC.

3.1.3 Interface to The Milling Machine

Develop an interface within the control system, which will interface with the CNC Mill, in order to fully automate the tool change.

3.2 Delimitation's

As far as is practically possible, components from the existing robot and control system should be reused to remain as cost effective as possible.

4 EXISTING EQUIPMENT

Since being built in 1991, the condition of the robot deteriorated to the extent that it was completely unusable due to numerous components being either damaged or lost.

4.1 PC Controller

There is an existing 286 AT PC used for control of the robot. It is fully functional other than losing BIOS settings as the internal battery no longer functions. This PC is suitable for simple control in a DOS environment using either Turbo Pascal or C to develop the software.

4.2 Stepper Motor and Controller

Rotational actuation of the robot arm is via a stepper motor (type HY3450-350-A8). The control of this motor is by a PCL-838 dedicated PC control card, which generates one digital pulse for each step of the motor. A terminate and stay resident (TSR) software driver, 'PCL-838.EXE' which is independent of the programming language, can be used to program low level instructions via common programming languages, giving simple but precise control of the stepper motor. The PCL-838 card also has three 8-Bit input and output ports which can be used for other control purposes.

The stepper motor driver, type SD-12, has been damaged to the extent that it no longer provided accurate control, and therefore required replacement.

4.3 Relay Control Card

In 1994, the original PCLD-785 PC Relay control board was damaged.

4.4 Mechanical Structure

The mechanical components of the arm were complete. While standing unused it had become dirty with minor corrosion of the guide shafts.

4.5 Instrumentation

All solenoid valves are still functioning, but the wiring is damaged and untidy and required replacement. None of the position sensors originally used were still in existence, and required replacement.

5 THE ROBOT CONTROL SYSTEM

5.1 The Original Control System

The original control system was PC based, and incorporated two separate PC Cards, as depicted in Figure 5.1. The Stepper Motor was controlled via the PCL-838 Programmable Stepper Motor Driver, while the Solenoid Valves to actuate the pneumatic cylinders were controlled using a PCLD-785 Relay Control card which had 16 digital relay outputs. Two feedback position sensors (reed switches) were also input using the PCLD-785 card which incorporated 16 optically isolated digital inputs.

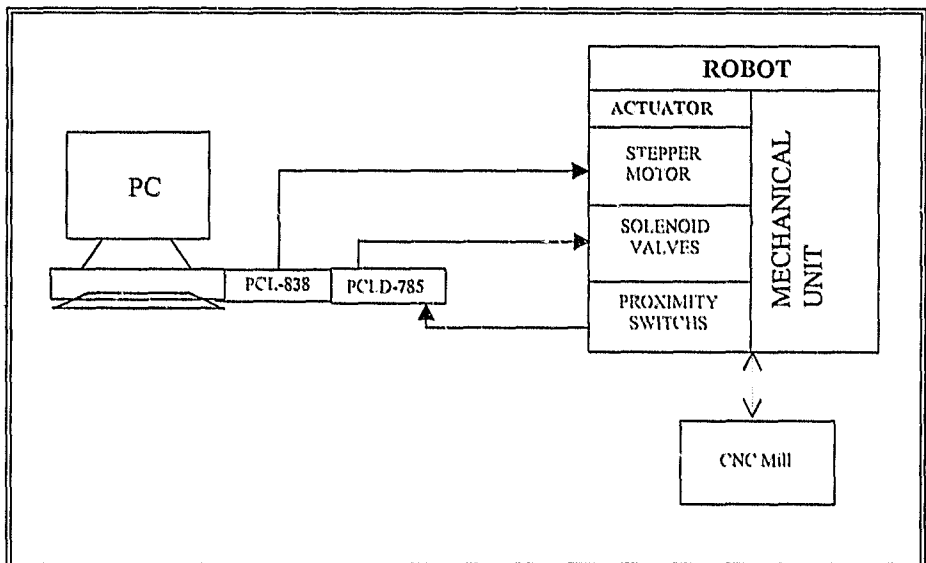


FIGURE 5.1 OVERVIEW OF ORIGINAL CONTROL SYSTEM

As is depicted, there is no control interface with either the environment or with the milling machine, which is an essential requirement for an automated tool change.

5.2 Choice of Controller

As the original control system was no longer operable, with damage to the PCLD-785 relay control card, wiring, and reed switches, the choice of controllers for the system was reviewed, and the option of using a PLC based control system investigated. A number of medium priced units are available for below R2000.

However, these low end PLC's are not designed for Stepper Motor applications, which require a pulse to be generated for each rotational step of the motor. While PLC's are capable of generating a series of pulses, they cannot easily control or vary the number and frequency of the pulses to achieve the accurate control, which is possible with the PC based system. This allows precise control of the displacement, velocity and acceleration of the motor, and hence the robot, in an open loop configuration.

The *SD12 Stepper Motor Driver*, which is required for either PC or PLC based control, has an internal pulse generator which could be switched by a PLC to drive the motor. However this would require the control system to be modified to closed-loop, incorporating a positional sensor at each radial position at which the arm may stop.

The PCL-838 stepper motor drive card also includes three 8-bit digital input ports and three 8-bit digital output ports. There are a number of adapter relay cards available which can be driven by the PCL-838's digital outputs, and these are significantly cheaper than dedicated PC-based cards such as the original PCLD-785.

For the above reasons, it was decided to remain with the PC based control, and replace the damaged PCLD-785 card with an electro-mechanical relay adapter card which would be driven by the PCL-838 stepper motor card. For this application, the PC-38 relay adapter unit was chosen.

5.3 Overview of Control System

An overview of the implemented control system is shown in Figure 5.2 overleaf. As is shown, the only connection to the computer is via the PCL-838's 37-Pin Female D connector, directly to another 37-Pin D connector on the Main Control Unit. This 37-Pin interface includes the wiring for the stepper motor, the solenoid valves, the interface with the Milling Machine and the positional sensors. Wiring diagrams for the 37-Pin connectors are shown in Figures 5.3 and 5.4 respectively.

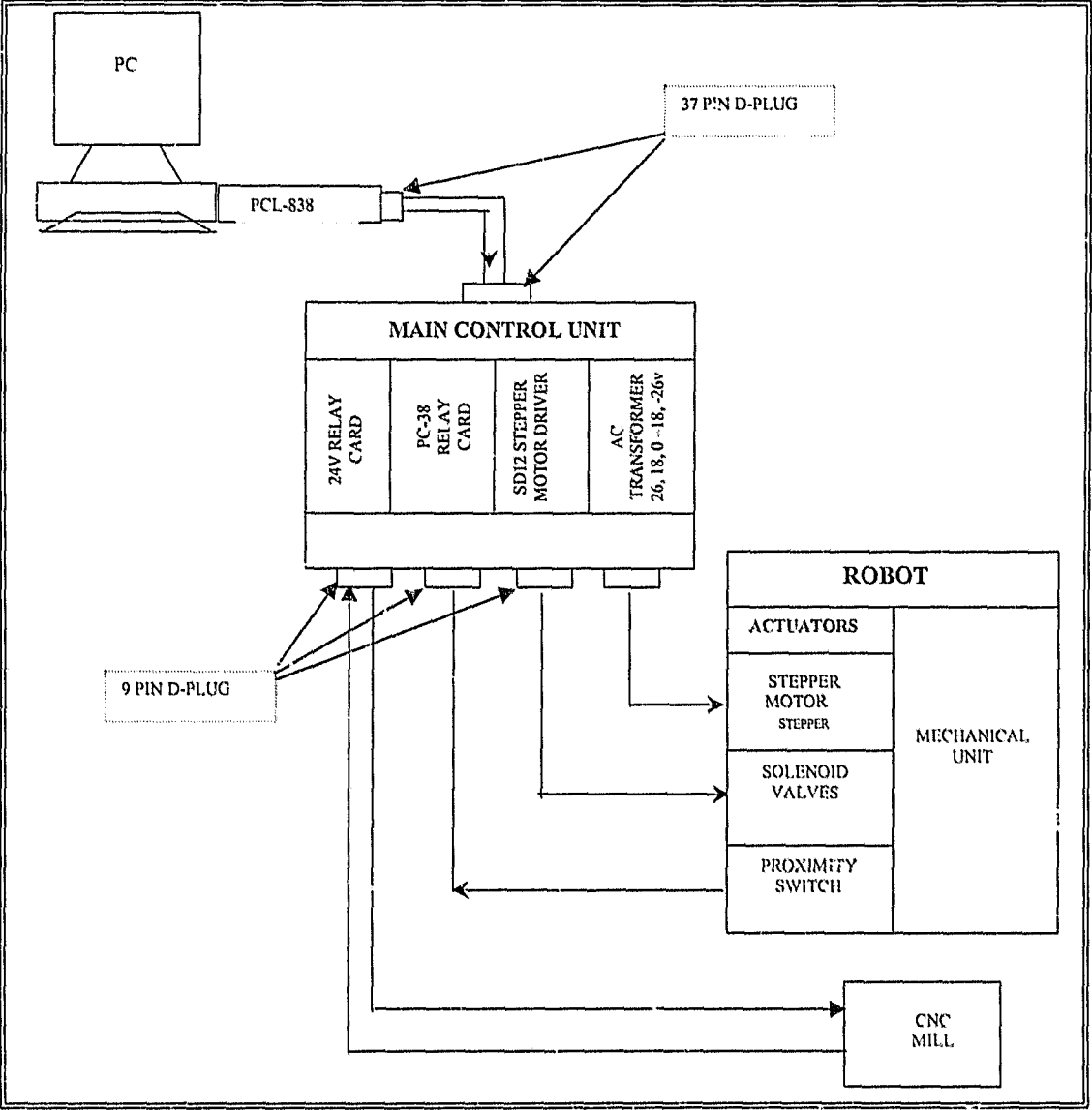


FIGURE 5.2 OVERVIEW OF THE IMPLEMENTED CONTROL SYSTEM

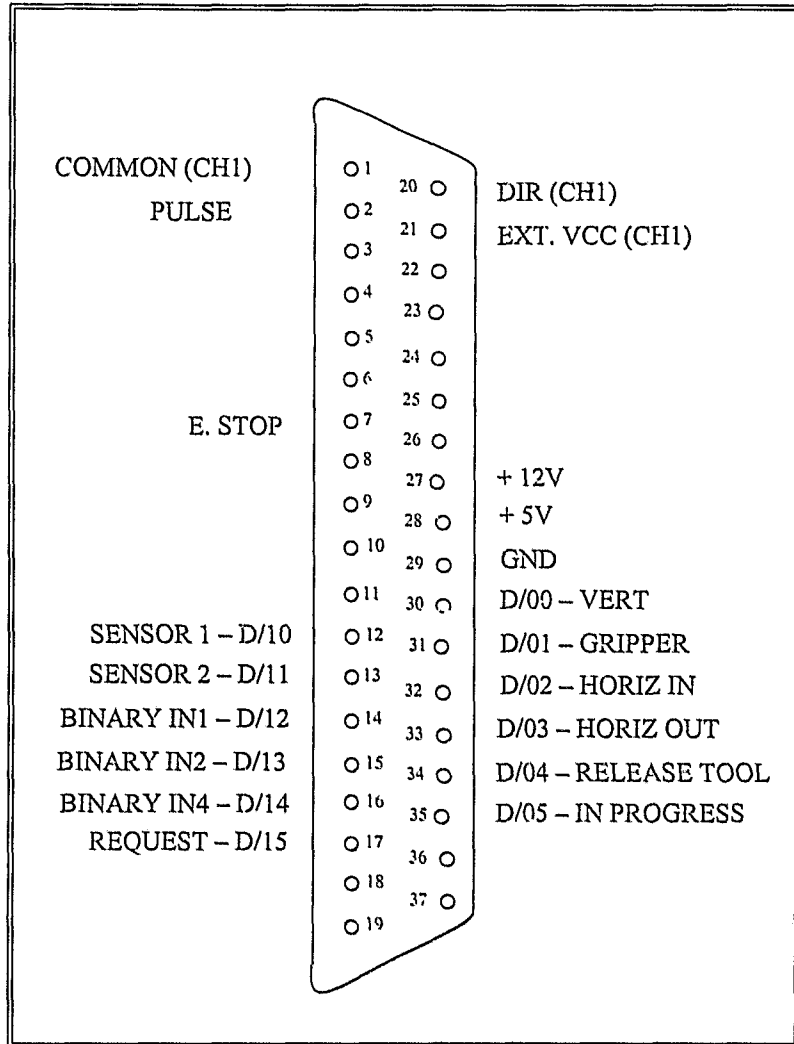


FIGURE 5.3 PCL-838 37 PIN ASSIGNMENT

TABLE 5.1 PIN ASSIGNMENTS FOR PCL-838 CARD

TAG	PIN No	DESCRIPTION
COMMON	1	Isolated Common for Stepper Motor
PULSE	2	Stepping Pulses for Stepper Motor
DIRN	20	Direction Signal for Stepper Motor
Ext Vcc	21	24 V Power supply from external source (SD-12 Stepper motor Driver)
E. STOP	7	Emergency Stop for Stepper Motor.
+12 V	27	12 V Output from Computer
+ 5 V	28	5 V Output from Computer
GND	29	PC Ground
SENSOR 1 D/I 0	12	Digital Input 0: Position Sensor for Horizontal Cylinder
SENSOR 2 D/I 1	13	Digital Input 1: Micro Switch to Check Arm Down
BINARY 1 D/I 2	14	Digital Input 2: Mill Interface Binary Tool Number Digit 1
BINARY 2 D/I 3	15	Digital Input 3: Mill Interface Binary Tool Number Digit 2
BINARY 4 D/I 4	16	Digital Input 4: Mill Interface Binary Tool Number Digit 4
REQUEST	17	Digital Input 5: Mill Interface Tool Request
D/O 0 VERT	30	Digital Output 0: Control of Vertical Cylinder Solenoid Valve
D/O 1 GRIPPER	31	Digital Output 1: Control of Gripper Solenoid Valve
D/O 2 HORIZ IN	32	Digital Output 2: Control of Horizontal In Solenoid Valve
D/O 3 HORIZ OUT	33	Digital Output 3: Control of Horizontal Out Solenoid Valve
D/O 4 RELEASE	34	Digital Output 4: Output to Mill to Release / Clamp Tool Chuck
D/O 5 IN PROGRESS	35	Digital Output 5: Output to Mill to Indicate Tool Change in Progress

TABLE 5.1 PIN ASSIGNMENTS FOR PCL-838 CARD

TAG	PIN No	DESCRIPTION
COMMON	1	Isolated Common for Stepper Motor
PULSE	2	Stepping Pulses for Stepper Motor
DIRN	20	Direction Signal for Stepper Motor
Ext Vcc	21	24 V Power supply from external source (SD-12 Stepper motor Driver)
E. STOP	7	Emergency Stop for Stepper Motor.
+12 V	27	12 V Output from Computer
+ 5 V	28	5 V Output from Computer
GND	29	PC Ground
SENSOR 1 D/I 0	12	Digital Input 0: Position Sensor for Horizontal Cylinder
SENSOR 2 D/I 1	13	Digital Input 1: Micro Switch to Check Arm Down
BINARY 1 D/I 2	14	Digital Input 2: Mill Interface Binary Tool Number Digit 1
BINARY 2 D/I 3	15	Digital Input 3: Mill Interface Binary Tool Number Digit 2
BINARY 4 D/I 4	16	Digital Input 4: Mill Interface Binary Tool Number Digit 4
REQUEST	17	Digital Input 5: Mill Interface Tool Request
D/O 0 VERT	30	Digital Output 0: Control of Vertical Cylinder Solenoid Valve
D/O 1 GRIPPER	31	Digital Output 1: Control of Gripper Solenoid Valve
D/O 2 HORIZ IN	32	Digital Output 2: Control of Horizontal In Solenoid Valve
D/O 3 HORIZ OUT	33	Digital Output 3: Control of Horizontal Out Solenoid Valve
D/O 4 RELEASE	34	Digital Output 4: Output to Mill to Release / Clamp Tool Chuck
D/O 5 IN PROGRESS	35	Digital Output 5: Output to Mill to Indicate Tool Change in Progress

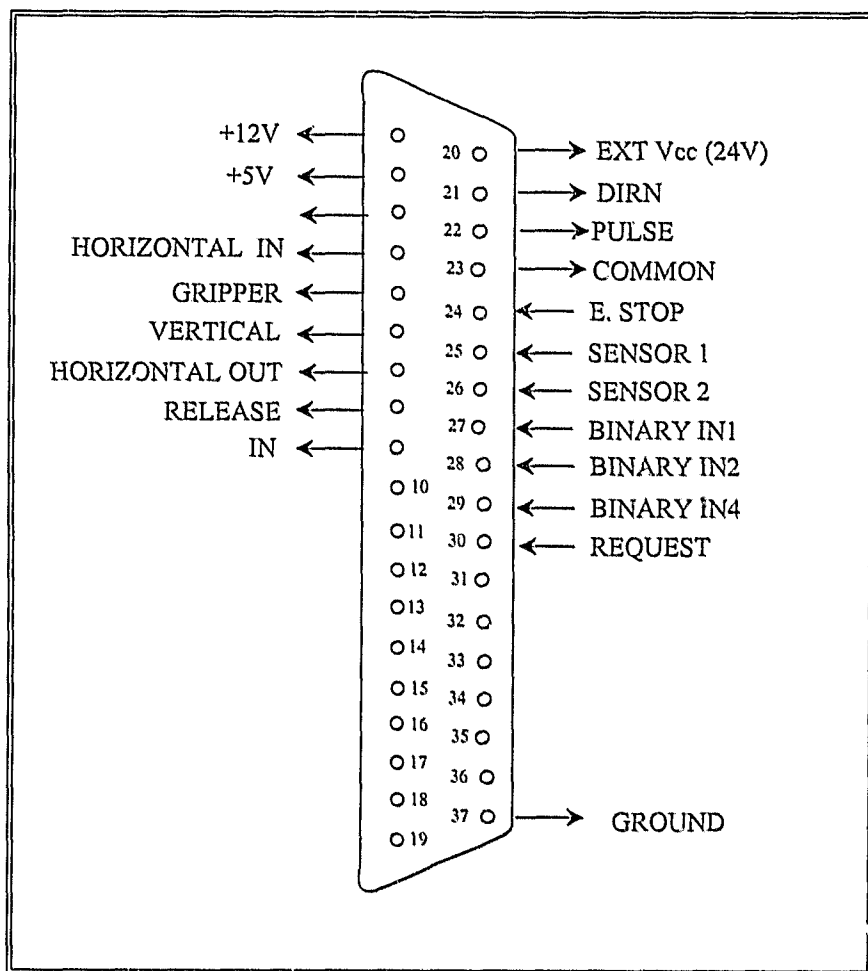


FIGURE 5.4 PIN ASSIGNMENT FOR MAIN CONTROLLER 37-PIN CONNECTOR

The Main Control Unit case contains the following items:

- An AC transformer to provide power to the stepper motor driver. This supplies 26V, 18V, 0, -18V and -26V AC outputs.
- The PC-38 Relay adapter card. The relays are switched from the PCL-838 digital outputs, and control the solenoid valves as well as a interface signal to the Milling machine.
- A 24V Relay card, which is used to isolate the 24V output signals from the Milling machine interface. These are then input into the PCL-838 card.
- The SD-12 Stepper Motor Driver.

Photographs of the Main Control Unit is shown below in Figure 5.5.

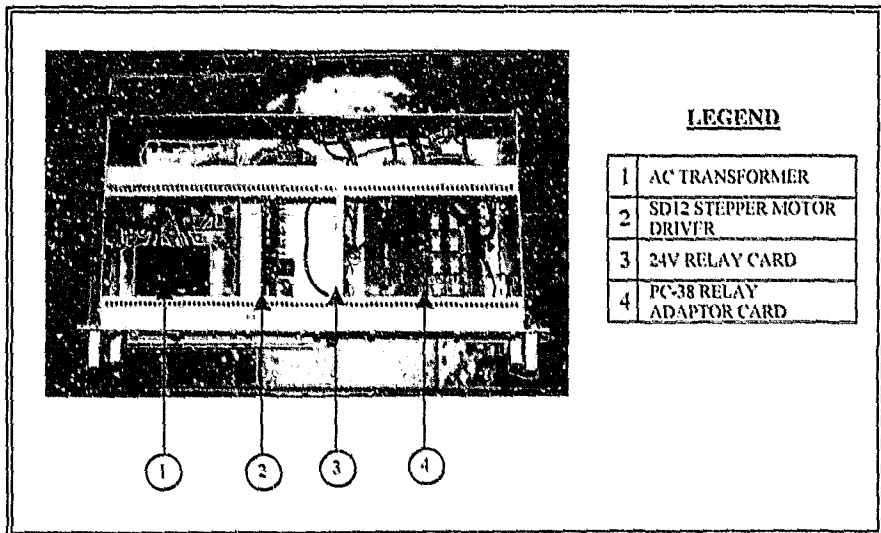


FIGURE 5.5(a) TOP VIEW OF MAIN CONTROL UNIT

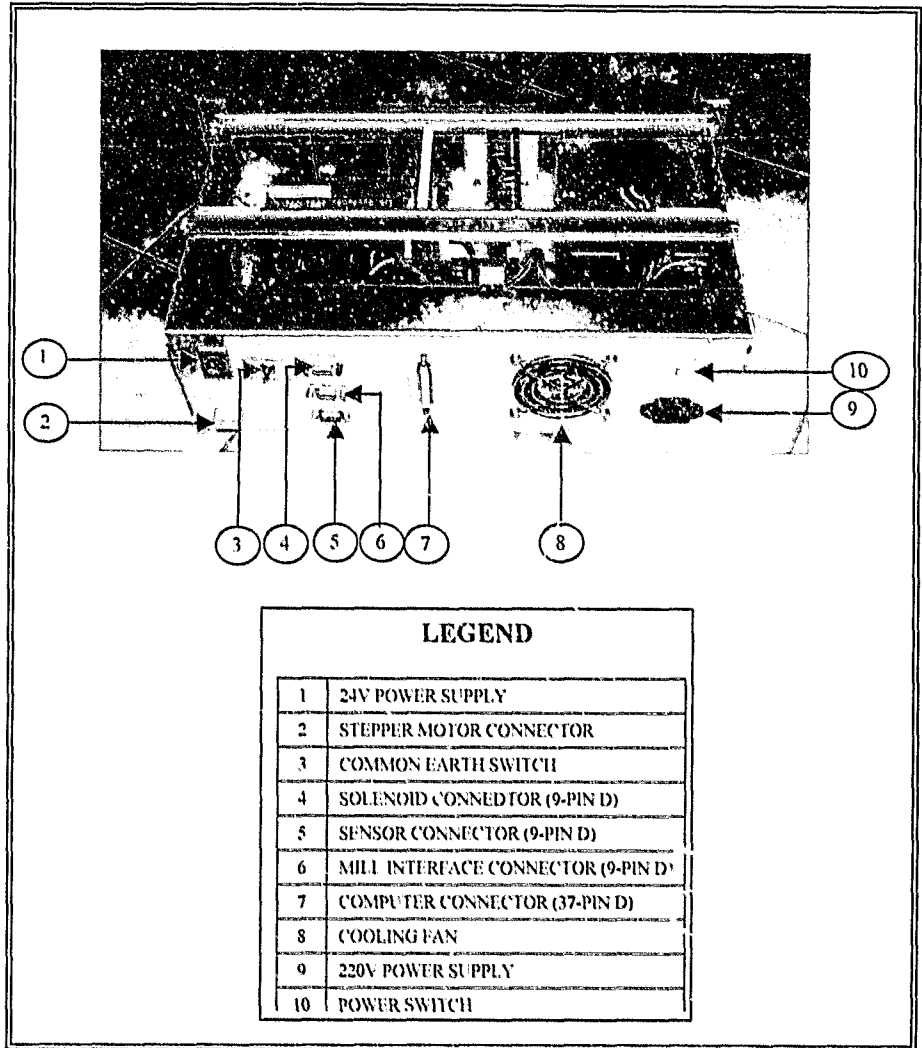


FIGURE 5.5 (b) MAIN CONTROL UNIT

As is shown in Figure 5.5 (b), the Main Control Unit provides plug in connections for the stepper motor, the position sensors, the solenoid valves and the milling machine interface. The box is enclosed with clear acrylic to prevent dirt ingress. A cooling fan has also been included to remove the heat generated by the transformer.

5.4 Interface with the CNC Mill

Following consultation with Dana Maas (1998) of Unimation CC, the agents for the CNC Milling Machine, a standard for the interface was developed, using a 9-Pin RS connector. The signal voltage is 24V.

When a tool change is required, the 'Tool Request' output from the CNC becomes live, and a 3 bit binary number representing the tool number is sent. At this stage, the Robot will make the 'Toolchange Done' output 0V, indicating the change is in progress.

When the tool must be clamped or unclamped, a one second pulse is sent on the 'Tool Unclamp' Output from the Robot, which will activate the relay on the CNC Mill tool chuck. When the tool change is complete, the 'Toolchange Done' Output becomes live again, and the robot will proceed. The pin assignment is as follows:

CNC to Robot:

Pin 1	Binary Bit 1
Pin 2	Binary Bit 2
Pin 3	Binary Bit 4
Pin 4	Tool Request

Robot to CNC:

Pin 5:	Tool change in progress
Pin 6:	Tool Unclamp
Pin 7:	0 V

A quotation for the modifications (a software upgrade and the wiring up of the 9-Pin RS connector) has been obtained, and is included in Appendix C. However, due to time constraints, this has not been implemented.

In order to proceed with testing of the Robot, a manual control box has been built to simulate the CNC Mill. The test box includes six buttons, and when pressed, sends a binary signal to the Robot to indicate the requested tool number. The test box also has two lights, which represent the 'Tool change complete' and 'Tool Unclamp' outputs from the Robot.

5.5 Replacement of Sensors

In order to facilitate greater operational flexibility, two positional sensors have been incorporated in the new design. Both are optional, and can be disabled in the software settings. This is discussed in the following chapter.

The first sensor (connected to the *SENSOR 1* input on the PCL-838 card) is a reed switch connected to the horizontal cylinder on the arm. This is used to position the arm in a mid position by switching the air supply to the cylinder off when the input becomes positive. The position of the sensor can be adjusted along the length of the horizontal cylinder, allowing the gripper extension at this state to be adjusted. This is useful to allow the position of the robot relative to the Mill to be adjusted.

The second sensor (connected to the *SENSOR 2* input on the PCL-838 card) is a micro switch, which is used to check the arm had returned to a down position after the tool has been removed from the Mill chuck. This is necessary as the tool tends to get stuck in the chuck. Should this occur, the limit switch will not be made, and the robot will not proceed with the remainder of the sequence, but will give an audible beep from the computer speaker, which repeats every couple of seconds until the problem is resolved.

5.6 Wiring of the Main Control Unit

The Main Control Unit is hard wired to allow the Mill, Solenoids and Sensors to plug in using standard RS 9-Pin D plugs. All wiring internal to the unit is fully tagged for quick identification. Wiring Diagrams are presented below for the following:

- The SD-12 Stepper Motor Driver (Figure 5.6)
- The PC-38 Relay card (Figure 5.7)
- The 24V Relay Card (Figure 5.8)
- 9-Pin D Connector to Mill Interface (Figure 5.9)
- 9-Pin D Connector to Solenoid Valves (Figure 5.10)
- 9-Pin D Connector to Position Sensors (Figure 5.11)

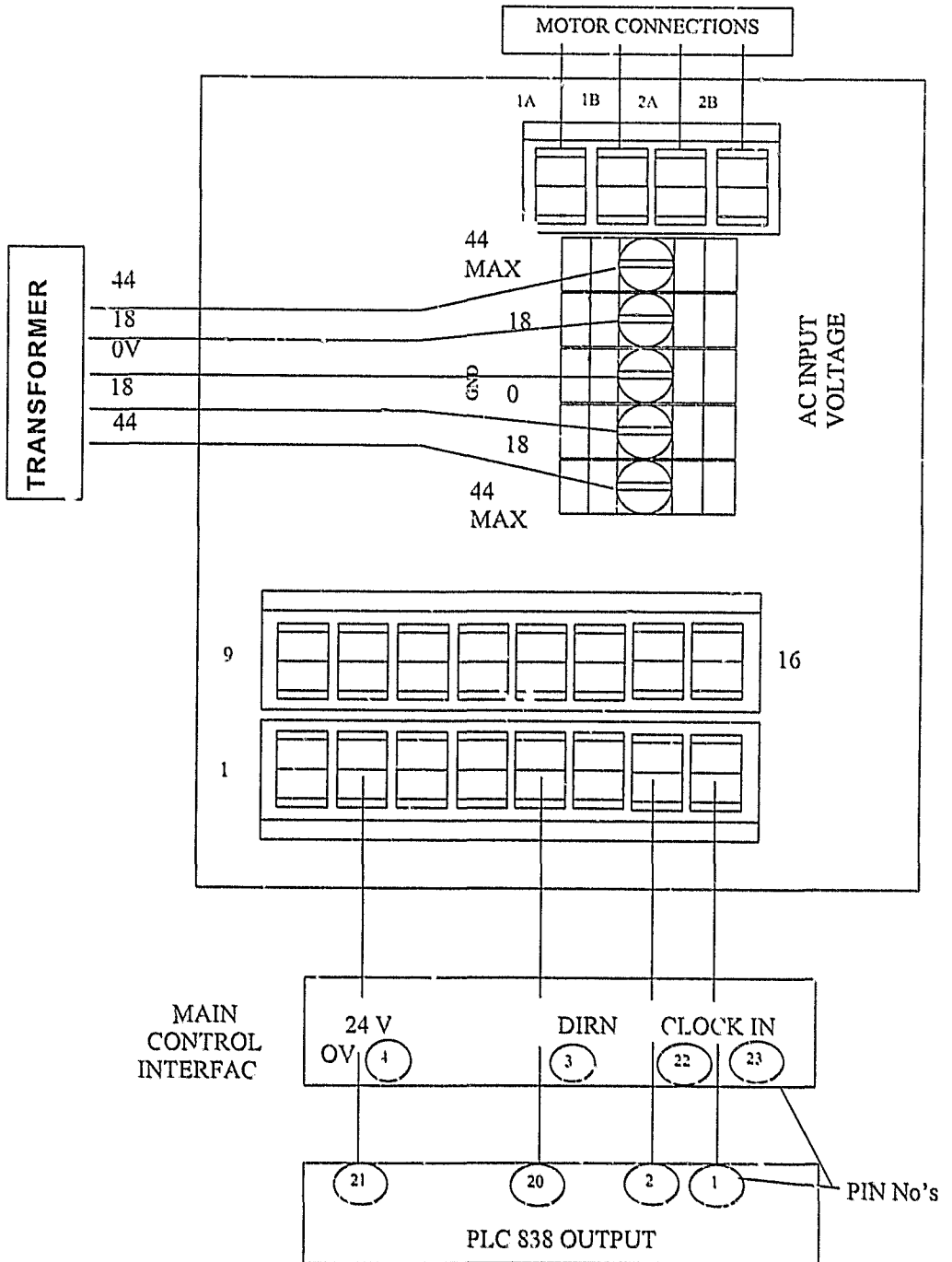


FIGURE 5.6 WIRING OF STEPPER MOTOR DRIVER

FROM PCL-838 37 PIN D

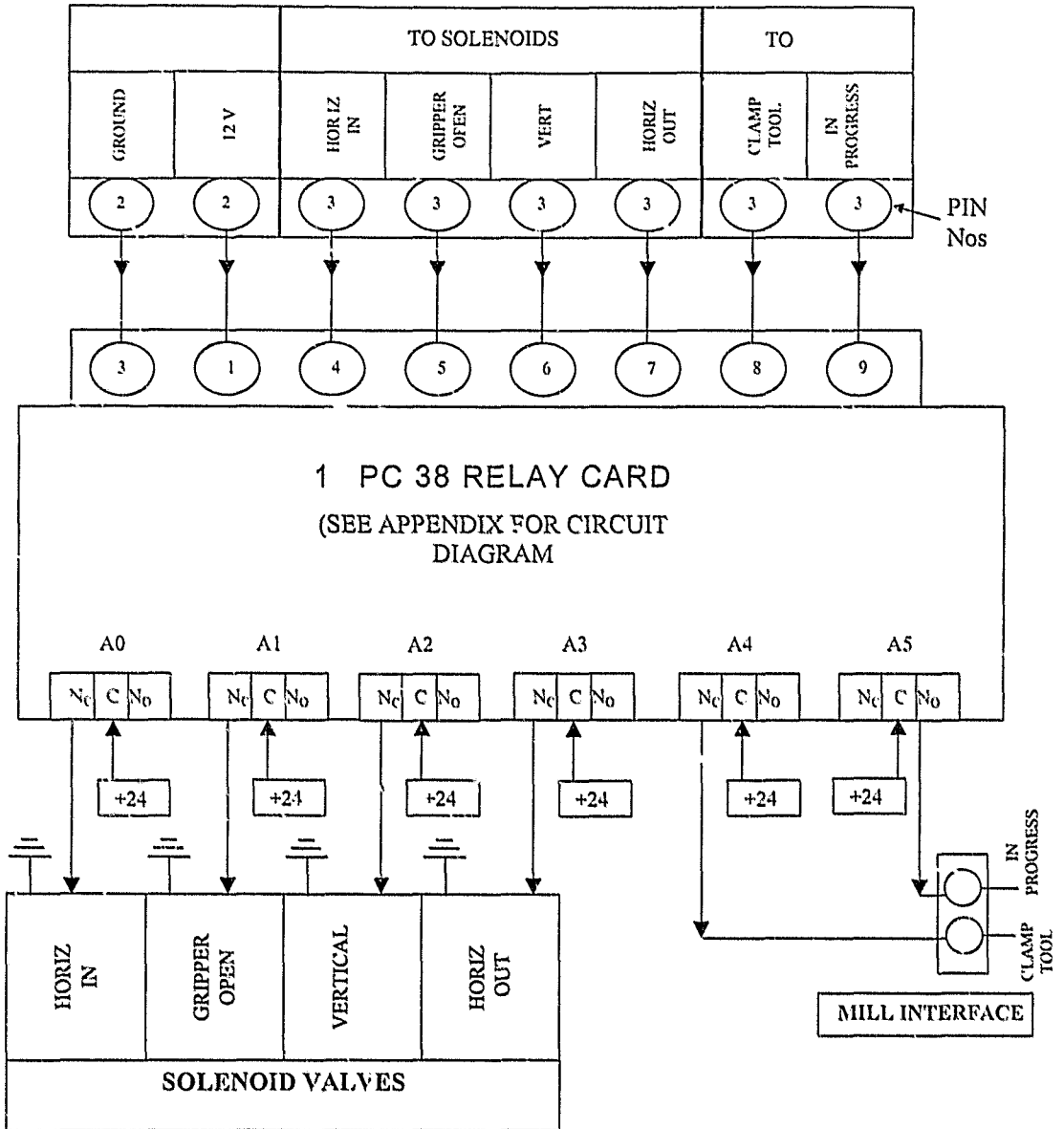


FIGURE 5.7 PC-38 COMPUTER RELAY ADAPTER CARD

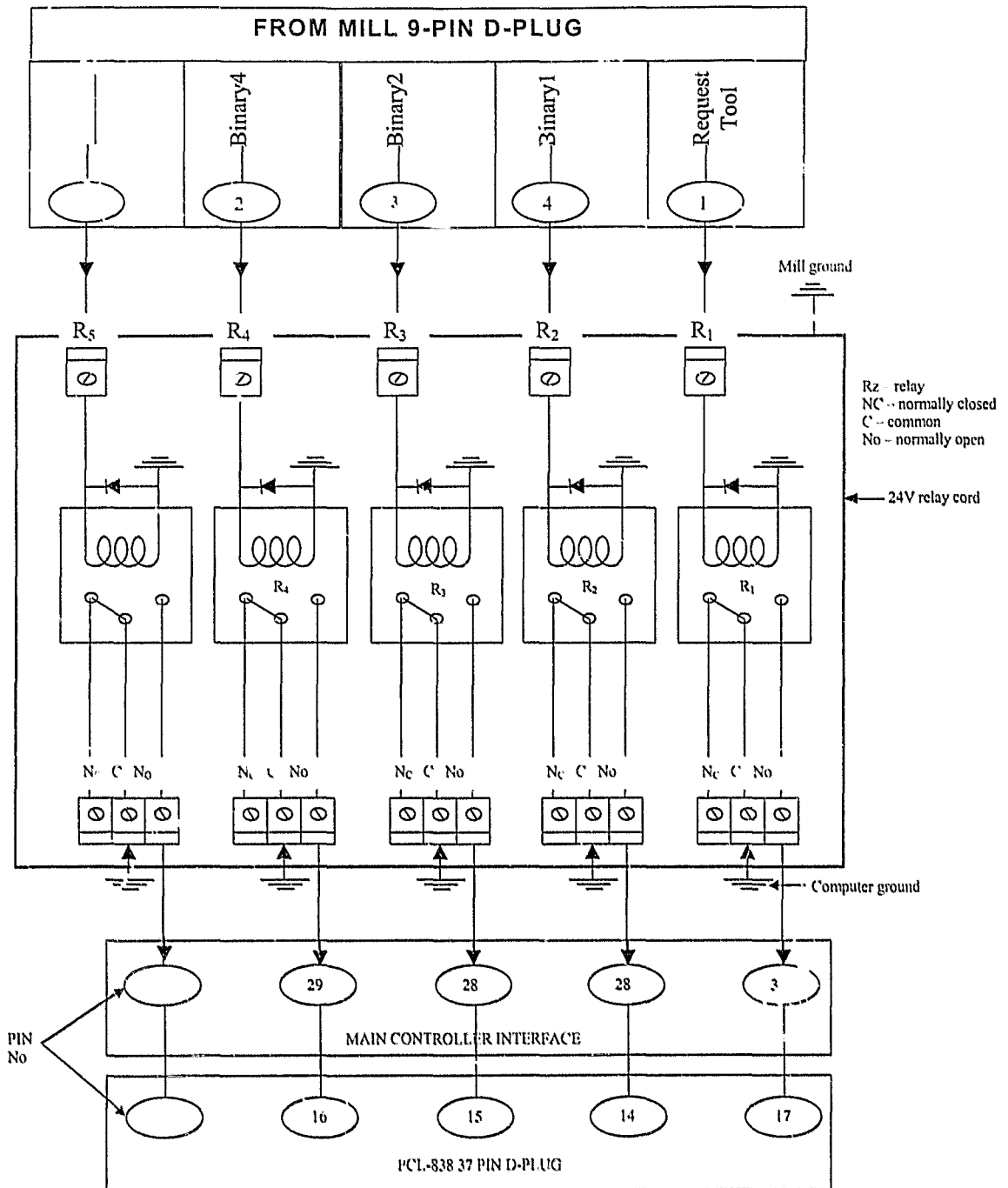


FIGURE 5.8 WIRING DIAGRAM FOR 24 V RELAY CARD

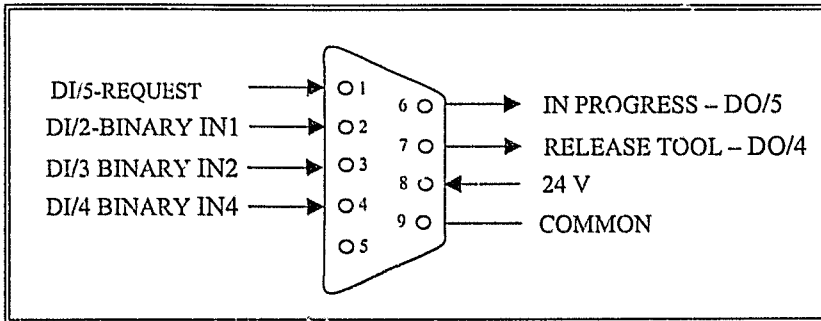


FIGURE 5.9 WIRING DIAGRAM FOR (9 PIN) MILL INTERFACE

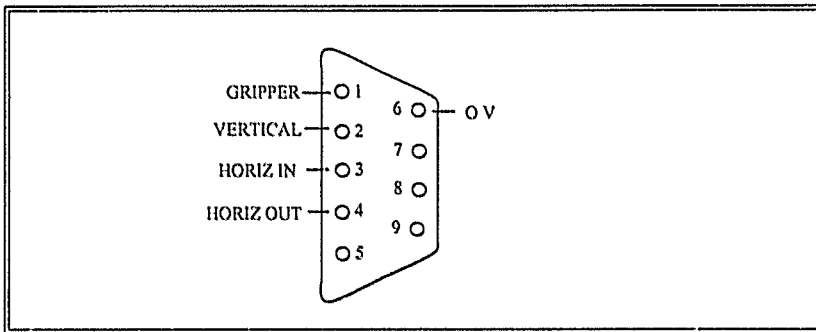


FIGURE 5.10 WIRING DIAGRAM FOR SOLENOID ACTUATORS (9 PIN)

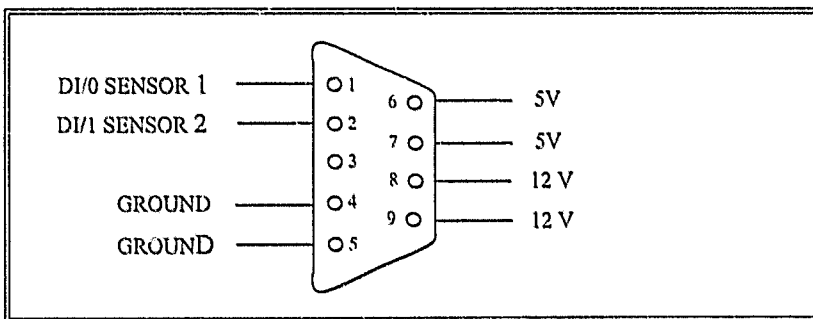


FIGURE 5.11 WIRING DIAGRAM FOR SENSOR CONNECTOR (9 PIN)

6 THE SOFTWARE

The original software was designed specifically for the original hardware, and included control of the solenoid valves via the PCLD-785 relay card, which is no longer used. The software was written in Turbo Pascal and used a simple command interpreter, "ADVICE.SYS" to control the stepper motor. This is an inefficient programming technique.

The PCL-838 comes with a language independent Terminate and Stay Resident (TSR) software driver, 'PCL-838.EXE', which once loaded into the computer's memory, allows the Stepping Motor to be programmed in any programming language. The programming method is documented in the PCL-838 Stepper Motor Control Card users manual: Chapter 5.

For the reasons discussed, it was decided to completely rewrite the software, again using Turbo Pascal. The complete listing of the software is given in Appendix A.

6.1 Overview of the Software

The control software has been developed to be as modular and flexible as possible. Procedures for movement of the arm are defined to provide a statement list, with each position defined by a statement, which is called to move the arm to that position, for example, to raise the arm with it fully extended, and the gripper open, the procedure '*Up_Close_Out*' is called. For rotation of the arm, the procedure '*Rotate*' is called, with the current position and new position (in step co ordinates) specified.

The digital output is an 8-bit port, and the individual outputs are switched by sending a number which is then represented as a binary digit on tha.

port. By way of illustration, the binary code for the number 9 is 00001001, so the first and fourth output would be switched on (last binary digit represents the first digital output, and each subsequent digit the next output). Each solenoid valve is controlled via a single digital output. Therefore, the value of the output designated which solenoid valves will be switched. Similarly, for the digital inputs, the computer is reading a number which when represented as an 8-bit binary number, specifies which inputs are on and which are not.

The software has also been designed to allow any control settings to be adjusted in a SETTINGS menu. From this the menu, the home position of each of the tools can be adjusted (Positions are specified in Degrees), as well as the angular position of the Mill. Other variables that can be adjusted include the rotational speed and acceleration of the arm, the pause time delay between movements to allow a movement to be completed.

The 'SETTINGS' menu also provides the option of enabling or disabling the two sensors. The first sensor allows the extension of the arm to be limited to a mid position which is adjustable by positioning the reed switch at the appropriate position on the horizontal cylinder. This is useful should the robot be positioned such that Mill Chuck is not located at the correct distance for full extension of the gripper.

The second sensor checks that the arm had moved down after retrieving a tool to ensure that the tool has not become stuck in the milling machine chuck, which is a common problem. If this function is enabled from the software and the tool becomes stuck, the robots motion is halted and an audible beep is repeatedly sounded until the problem is resolved.

6.2 Operating the Software

Operation of the software is extremely simple. The program is initiated by typing 'ROBOTX' (if necessary the computer drive assignment and / or root directory where the software is located may be needed). The software takes a few seconds to initiate. From there, on screen prompts specify the options which are available. Validity and range checking is provided for all inputs to ensure stability of the software and prevent the system crashing due to a user error.

6.3 Detailed Description of Software Procedures

A description of each procedure in the software is described below:

<i>Welcome</i>	This procedure initiates the welcome screen when the software is run. The user is given the option of varying the software settings (see procedure <i>Get_Settings</i> below). The user is also offered the option of running the manual test program (see <i>Test</i> below).
<i>Base_Active</i>	This procedure initiated control of the PCL-838 driver card by loading the software driver into the computer memory.
<i>Check1</i>	Ensures that the software driver 'PCL-838.EXE' is correctly loaded into the computer memory

Get_Settings

This procedure reads the data stored in the file 'SETTINGS.DAT' into the computer memory. This file contains data which is used to adjust the control and functions of the robot. It includes the following:

- variables which describe the radial position of the mill chuck, each tool in the tool holder and the 'home' position of the arm (in degrees). Any of these can be adjusted, to cater for mechanical or positional adjustments of the robot and tool holder.
- the acceleration rate and maximum speed of the stepper motor.
- variables which allow the user to select if either of the two position sensors are to be used (as detailed in section 5.5).
- a variable which allows the user to specify if the robot is going to be controlled automatically from the mill CNC interface, or via a manual user input using the PC keyboard.

If the software cannot find the file 'SETTINGS.DAT', which would be the case the first time the software is run on a different computer, a new file is created.

Change_Settings

This procedure is called by the software if the user selects the option of changing the settings from the *Welcome* menu. When called, it allows any of the variables stored in the file 'SETTINGS.DAT' to be changed, and the new values are re-written into the file SETTINGS.EXE. Range checking is performed at each stage to ensure only valid values for each parameter are entered. Should a value outside the

specified range be entered, the software requires the user to re-enter the value until a valid value is entered. The software also allows either lower- or upper case character inputs.

.

Test This procedure allows the user to test simple individual movements of the arm from a menu on the screen, using keyboard inputs.

.

Waiting This sends an output to the mill, which indicates that the robot is idle. When a tool change is in progress, this output is low (off), and becomes high again once the tool change is complete, indicating the mill can continue with its routine.

.

Clamp_Unclamp Sends an output to the mill for one (1) second to activate the relay which releases the hydraulic tool chuck. This is required to both release and clamp the tool.

.

Check_Arm_Down This procedure is only called if the vertical position sensor is turned on from the *Settings* menu. It checks that the arm has returned to the down position after a tool is retrieved from the mill chuck. If this does not happen, the robot sequence is delayed and an audible beep is sounded repeatedly until the problem is resolved.

.

Tool_Number This procedure reads in the new tool number from either the mill interface, or from a manual input into the PC keyboard, depending on the configuration of the software in the *Settings* menu.

Full range checking is performed to ensure that only a valid tool number is input.

Delay

Creates a pause in the robot's motion, the length of which is specified in milliseconds.

Rotate

This procedure controls the stepper motor in order to rotate the robot arm from it's current position to a new position, as specified by the variable '*new_pos*'. The arm is accelerated from rest to its maximum speed over a specified time, which is stored the 'SETTINGS.DAT' file and can be adjusted. The arm then rotates at constant speed until close to its new position before decelerating at a rate equal to the initial acceleration. This is to avoid unnecessary stress on the motor. The radial position of the arm at all times is stored in the variable '*current_pos*'.

Down_Close_In; Down_Close_Out; Down_Open_In; Down_Open_Out;
Up_Close_In; Up_Close_Out; Up_Open_In; Up_Open_Out

These procedures form a statement list, which is used later in the software. Each describes a position of the arm. By calling the procedure '*Up_Open_In*', the arms position would move such that it was in an up position with the gripper retracted and open. Each procedure sends a binary output to the solenoid valves which control the cylinders, which corresponds to a position of the arm as described in Table 6.1.

Down_Close_Half; Up_Close_Half; Down_Open_half; Up_Open_Half

As above, these procedures form a statement list, each describing a position of the arm, which is used by the rest of the software. However these procedures are only called if the option to use the reed switch on the horizontal cylinder is specified in the *Settings* menu. If this is the case, the horizontal extension of the arm is controlled to a position determined by the position of the reed switch on the cylinder. This can be adjusted to allow any amount of horizontal extension when putting a tool in the mill chuck.

Get_Current_Tool This procedure uses the above described procedures to retrieve whichever tool is currently in use in the mill (as stored in the variable '*current_tool*') and return it to it's correct location in the tool holder once a valid tool change has been requested.

Put_New_Tool This procedure uses the above described procedures to retrieve whichever new tool is requested during a tool change from the tool holder, and place it in the mill chuck before returning the robot arm to the home position.

Table 6.1 Robot Positions Corresponding to Binary Outputs

ROBOT ARM POSITION	BINARY VALUE	DECIMAL VALUE
Down_Close_In	0100	4
Up_Close_In	0101	5
Down_Open_In	0110	6
Up_Open_In	0111	7
Down_Close_Out	1000	8
Up_Close_Out	1001	9
Down_Open_Out	1010	10
Up_Open_Out	1011	11

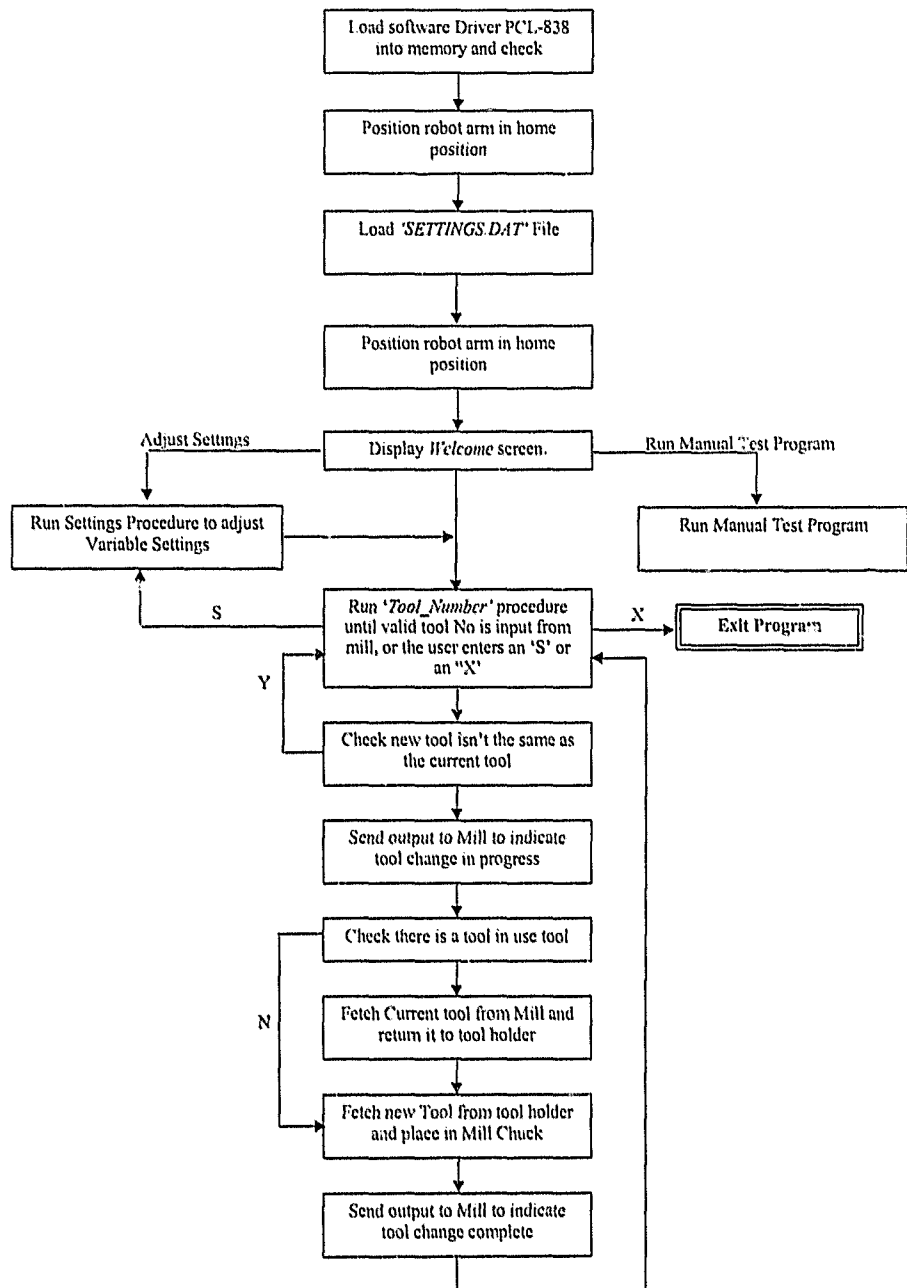


Figure 6.1 Flow Diagram for Main Software Sequence

7 MECHANICAL MODIFICATIONS

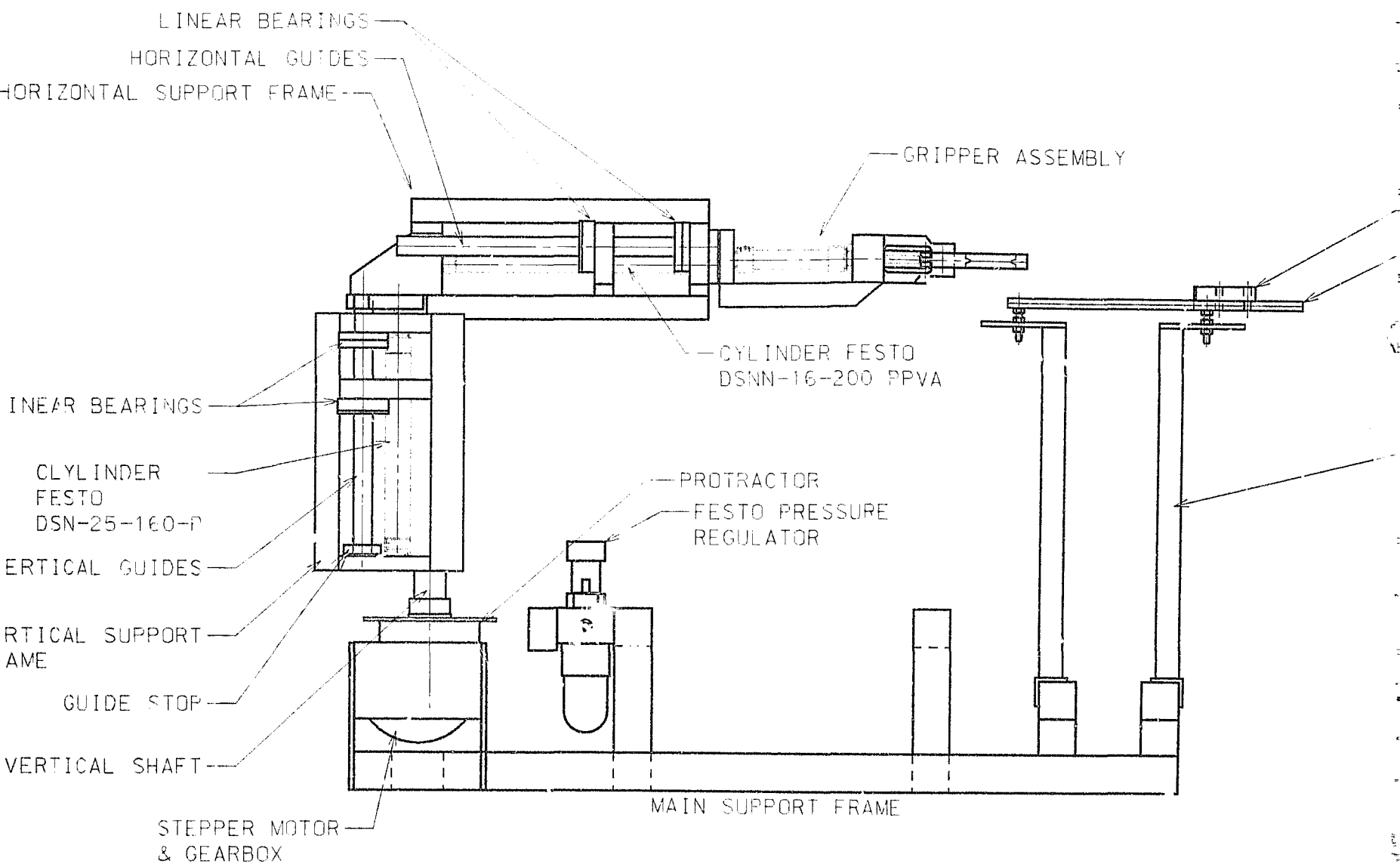
Mechanically, the robot was complete and fully functional. For completeness of this report, a set of assembly and component drawings has been produced and is presented in Appendix B.

However, there was no stand to mount the robot in position to perform a tool change on the Milling machine. A problem was also experienced with the position and alignment of the tool holder, as the original stand was not well constructed and offered little adjustment for the tool holder.

A new stand for the robot was built which consists of a simple rectangular frame, made from 38x38x2 mild steel tube, mounted on the floor, which positions the robot at the correct height to perform a tool change. This is shown in Figure 7.1.

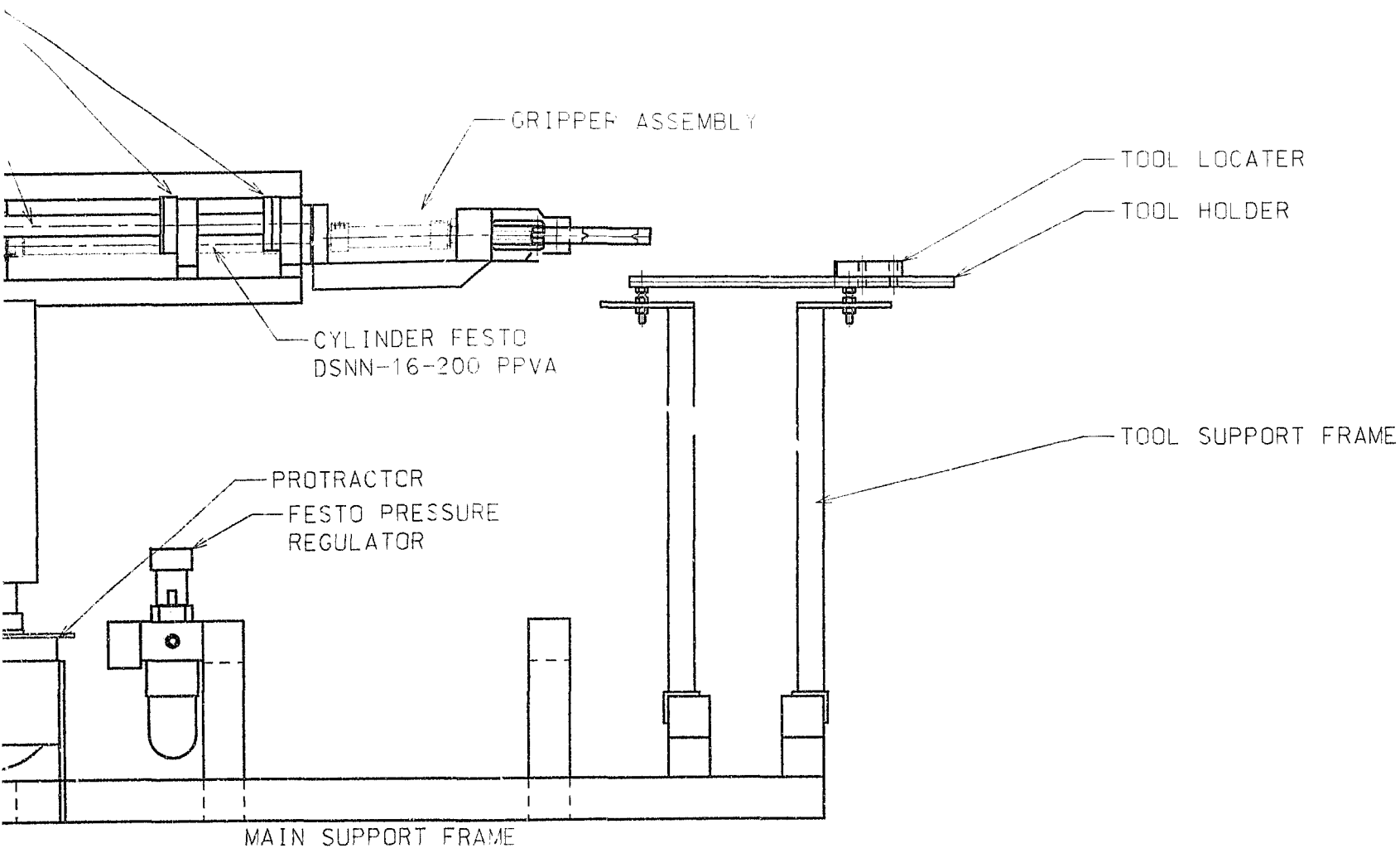
The tool stand support frame was also re-built to allow for adjustability and greater stability, while still being a far simpler construction. This is shown in Figure 7.3

A limiting collar was also placed on each of the two 20mm horizontal guide shafts to restrict the maximum reach of the arm and prevent it from hitting the tool holder when extended. Boakes et al (1994) documented problems with the arm shooting out too fast and proposed a number of methods to overcome this, concluding that the most feasible option would be to pulse the air supply to the cylinder by switching the solenoid valve. However this problem was easily overcome by using the air flow restrictor, thereby slowing the flow of air into the cylinder, and hence reducing the velocity of the gripper.



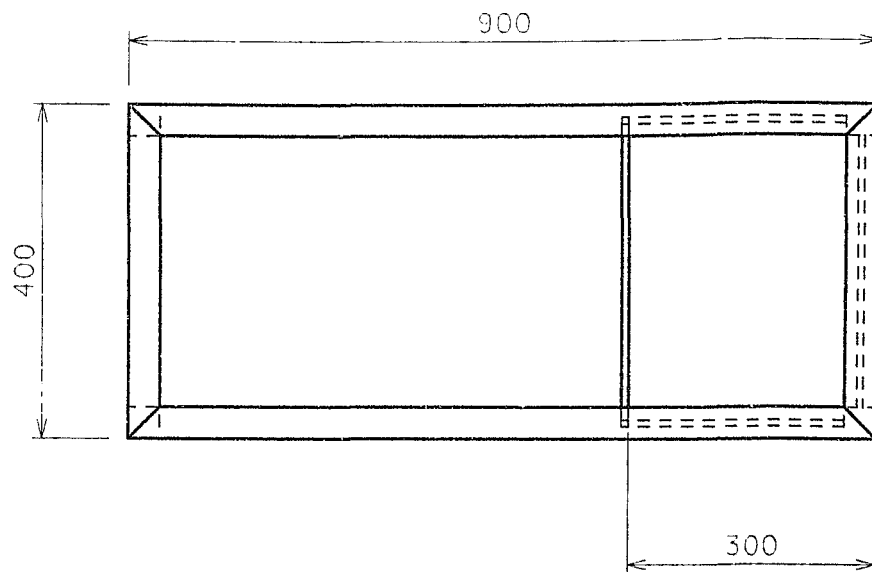
ELEVATION ON ROBOTIC ARM TOOL SELECTOR

TITLE:		P. 10
DRAWN		
SCALE		

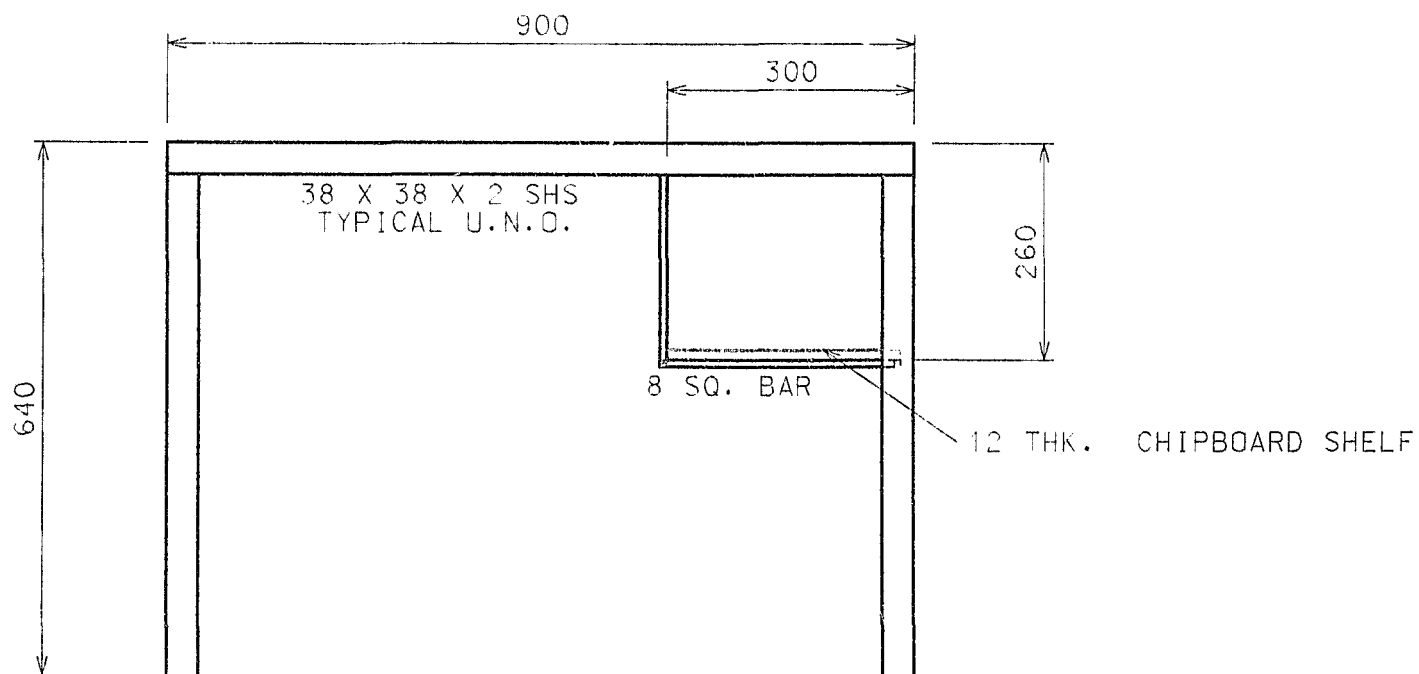


ATION ON ROBOTIC ARM TOOL SELECTOR

TITLE:		ROBOTIC ARM TOOL SELECTOR	
DRAWN		DRG. NO.	FIGURE 7.1
SCALE	1:5	REV	



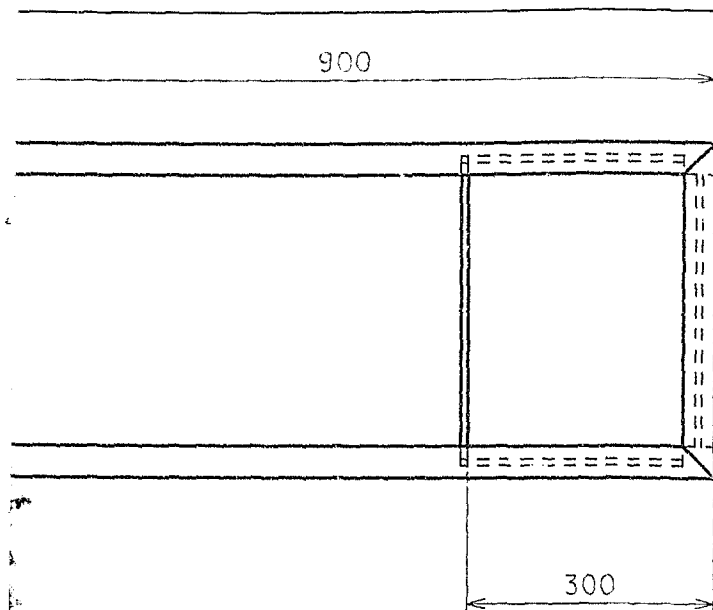
PLAN



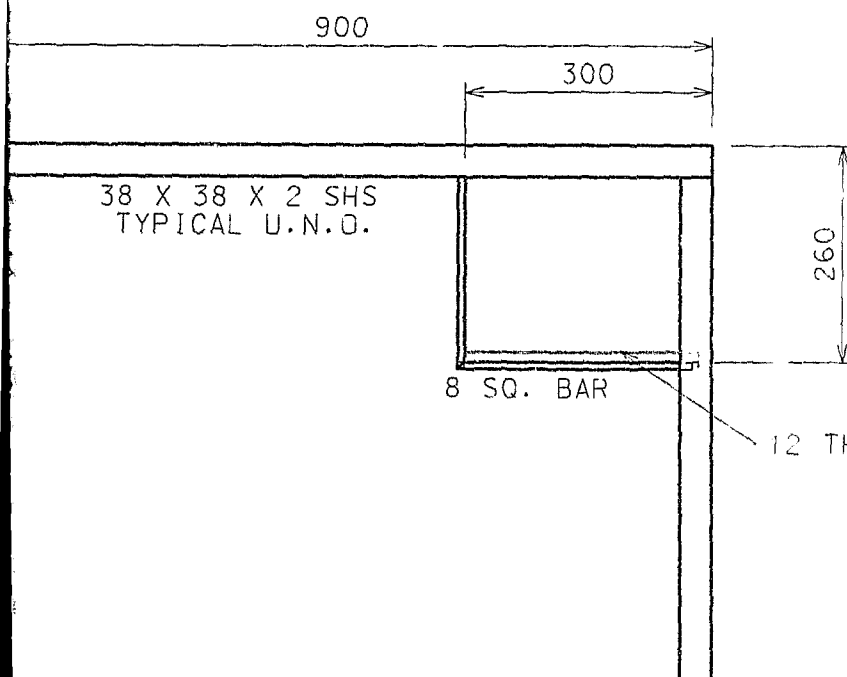
ELEVATION ROBOT SUPPORT FRAME

TOLERANCE 0.5 UNO

TITLE: ROBC ROBOT SU		
DRAWN		
SCALE	1:7.5	



PLAN

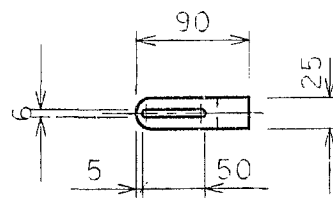


12 THK. CHIPBOARD SHELF

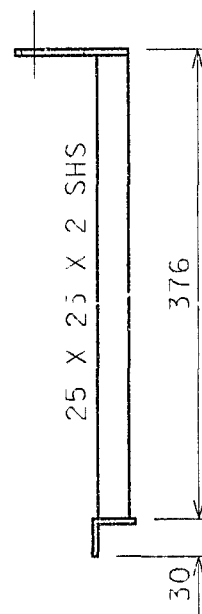
ELEVATION ROBOT SUPPORT FRAME

TOLERANCE 0.5 UNO

TITLE: ROBOTIC ARM ROBOT SUPPORT FRAME			
DRAWN		ORG. NO.	REV
SCALE	1:7.5	FIGURE 7.2	

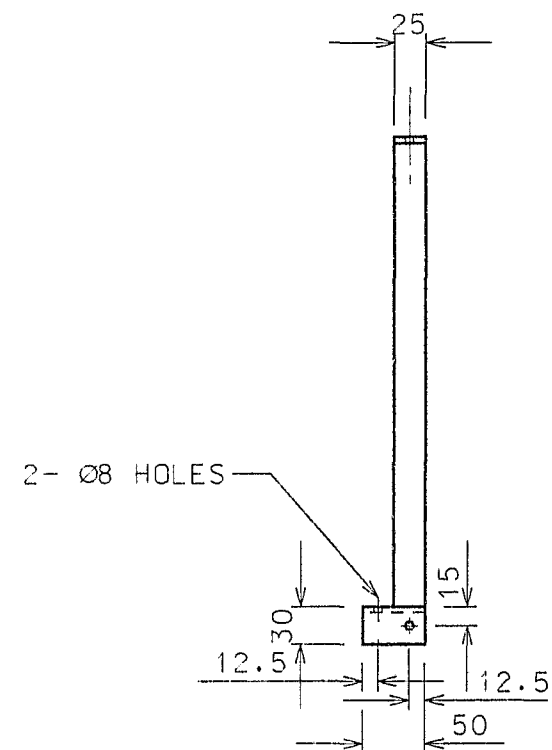


PLAN



ELEVATION ON TOOL HOLDER SUPPORT

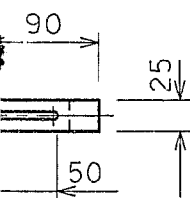
3- REQ'D AS DRAWN



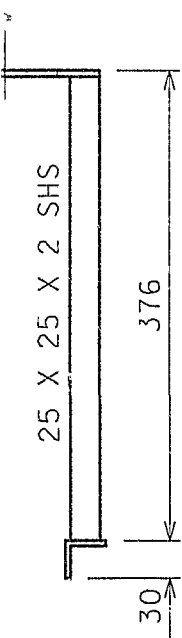
END VIEW

ALL DIMENSIONS ± 0.2

TITLE:		RO
		TO
		SUP
DRAWN		
SCALE		1:

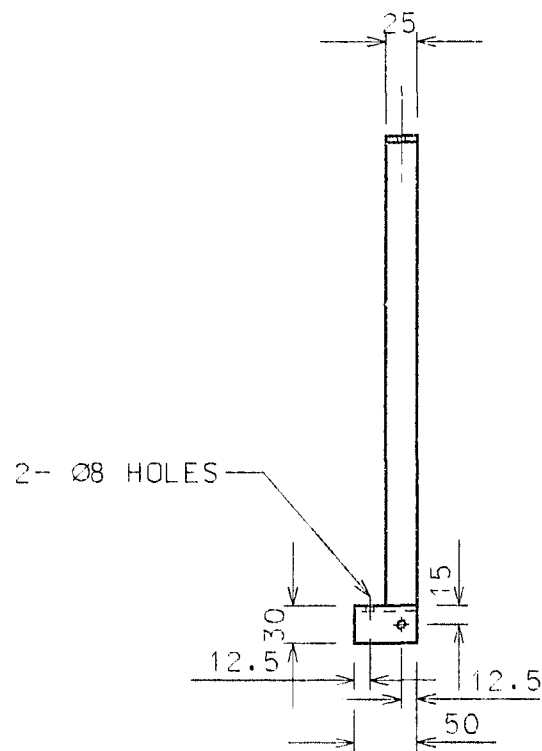


PLAN



TOOL HOLDER SUPPORT

REQ'D AS DRAWN



END VIEW

ALL DIMENSIONS ± 0.2

TITLE: ROBOTIC ARM TOOL HOLDER SUPPORT FRAME			
DRAWN		DRG. NO.	REV
SCALE	1:5	FIGURE 7.3	

8 CONCLUSION

The Tool Change Robot has been modified substantially, particularly the control system, to make it operational again and to incorporate an interface with the environment.

The damaged PCLD-785 Relay Control Card has been replaced with a PC-38 Relay Adapter card, which is switched by the digital outputs from the PCL-838 Stepper Motor Driver. All Electronic hardware external to the computer has been built into the Main Control Unit, which is connected to the PC via a single cable with 37-Pin D-type connectors. The main Control Unit has been hard wired to provide plugs for all auxiliary equipment to connect easily to it.

All software has been re-developed to improve the functionality. Procedures are used to define a statement list of all positions for the robot arm which can be simply called. The software also makes provision for many control variables, such as the position of each tool, and the maximum speed of rotation, to be adjusted and stored permanently in a separate file. The software also allows for either the CNC Mill interface to be used, or the user to select the new tool from the PC Keyboard.

The software has also been developed to be stable, and provides range checking of all inputs, and requires the user to re enter any value which is incorrectly entered. It also incorporates a simple test option which allows individual movement procedures, such as rotation of the arm to a specific position, to be tested individually using only keyboard inputs.

The new control system also includes two optional sensors, which can be switched off from the 'SETTINGS' menu in the main program. The first is used to control the extension of the arm and limit it to a mid position if

required, and the second provides a check that the arm has lowered after retrieving a tool from the mill, to ensure no damage occurs if the tool becomes stuck in the mill chuck.

An interface to the CNC has been developed with the agents for the CNC Mill and this has been incorporated into the Robot. However, modifications are still required on the CNC Mill for the interface. However, a manual control unit has been built to simulate the CNC interface for testing purposes. This manual control unit plugs into the 9 pin mill interface plug (Figure 5.9)

A stand had also been built for the Robot to position it correctly to perform a tool change on the Mill. The Tool Holder support has also been re-built more simply but to allow better adjustment of the Tool Holder and increase rigidity.

The robot has been completely tested and all components are completely operational. This was demonstrated to Mr E Fielding, the project supervisor from the Department of Industrial Engineering. During the demonstration, the robot performed a number of complete tool changes on the milling machine while using the manual control unit to simulate the Mill interface.

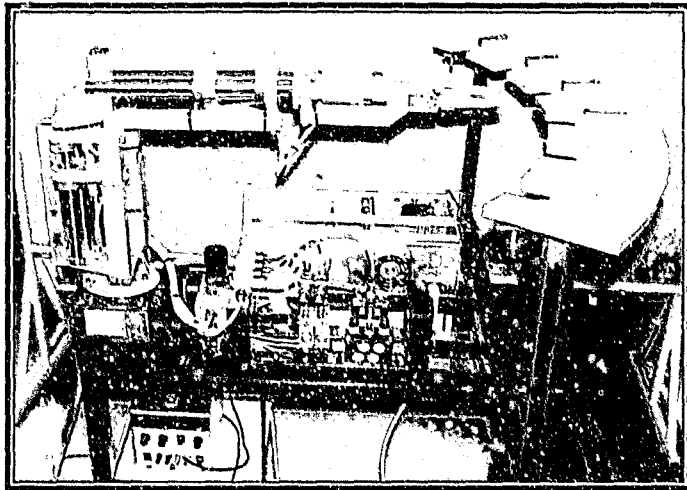


FIGURE 8.1 OVERVIEW OF ROBOT

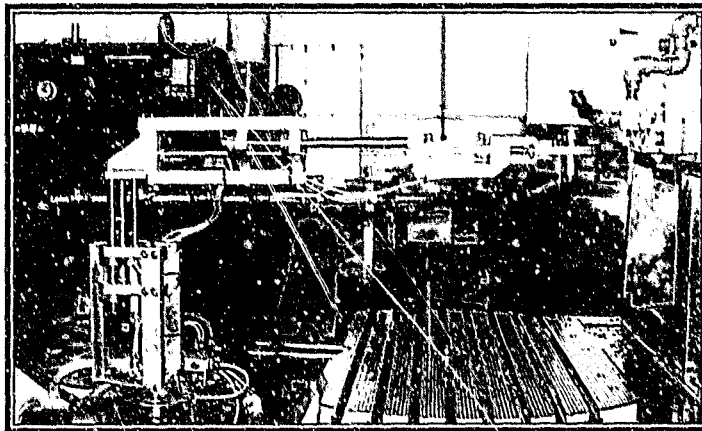


FIGURE 8.2 ROBOT ARM EXTENDED TO RETRIEVE TOOL FROM MILL.

9 FURTHER WORK REQUIRED

The robot meets all the requirements and is capable of performing a completely automated tool change. However the following two items should still be addressed:

1. The modifications to the CNC Mill should be implemented. A quote is included in Appendix C, and includes the scope of work required.
2. There is currently no way of rotating the Milling Machine chuck to a standard position during the tool change. This is necessary, as when a tool must be put into the chuck, the driving lugs on the chuck must be aligned with the key ways on the tool for it to fit. This problem is discussed by Chang et al (1993) and again by Boake et al (1994). They conclude that this can be addressed by placing a proximity switch on the clutch spindle, and writing a routine in the CNC software which would rotate the spindle at the lowest RPM, 40, until the sensor is made and the spindle would be stopped. To implement this, the agents for the CNC would be required to implement the change to wire the sensor input into the CNC controller.

10 REFERENCES

1. Smargiasso M.F 'The Design of a Tool Change Device for a Numerically Controlled Milling Machine' , Final Year Design Report, Faculty of Mechanical Engineering, University of the Witwatersrand, 1991.
2. Bird V, Comodakes C, Hayt G, 'Tool Changing Robot for an NC Milling Machine' Third Year Laboratory Report, Faculty of Mechanical Engineering, University of the Witwatersrand, 1992.
3. Chang C, Corsaro L.F, Svoboda V, 'Tool-Changing Robot', Faculty of Mechanical Engineering, University of the Witwatersrand, 1993.
4. Boake T.S, Buldeli S.M, Henning D, McCaffrey D, Smith S.A, 'Tool Changing Robotic Arm' Faculty of Mechanical Engineering, University of the Witwatersrand, 1994.
5. Fuller, James L 'ROBOTICS. Introduction, Programming, and Projects' Macmillan Publishing Company 1991.
6. Taylor, P.M 'Robotic Control', Macmillan Educational, 1990.
7. Hohmann T, 'Why PC's won't kill PLC's', S.A Instrumentation and Control, February 1997 : 72 – 75
8. Barwise C, Moodley J, 'Take advantage of control options' S.A Instrumentation and Control, March 1998 : 2 – 4.

9. Maas, D. Unimation CC (tel 011- 845-3278). Verbal Consultation concerning interface to CNC Milling Machine.
10. '*PCL-838 Stepping Motor Control Card Users Manual*', Advantech Co, 1992

APPENDIX A

PRINTOUT OF THE SOFTWARE

{ \$m 8000,0,0 }

PROGRAM ROBOT;

{CONTROL SOFTWARE FOR THE CNC MILL TOOL CHANGE
ROBOT}

{THIS PROGRAM USES THE TSR SOFTWARE DRIVER, 'PCL-838.EXE'}

USES

dos,crt;

CONST

{SETTING UP A PARAMETER TABLE FOR REFERENCE BY PCL-838 DRIVER}

_ch1 = \$0001;

_cw1 = \$0000;

_ccw1 = \$0100;

CMD_SetTime = 1;

CMD_SetStep = 2;

CMD_SetVelocity = 3;

CMD_Active = 10;

CMD_Transfer = 11;

CMD_Run = 12;

CMD_Pause = 13;

CMD_Continue = 14;

CMD_Remain = 15;

CMD_Free = 16;

CMD_Hold = 17;

CMD_Loop = 18;

CMD_Stop = 19;

CMD_LoopNo = 20;

```
CMD_Output = 30;  
CMD_Input = 31;  
CMD_Delay = 42;  
CMD_Wright = 50;  
CMD_Read = 51;  
CMD_Base = 53;  
CMD_Reset = 54;  
CMD_SetTimer = 55;  
CMD_WaitTimer = 56;  
CMD_RemainLp = 58;  
CMD_Clear = 59;  
CMD_CwStep = 60;  
CMD_CcwStep = 61;  
CMD_StepSpd = 62;  
CMD_SlowTime = 63;  
CMD_SlowStep = 64;  
CMD_SlowVelocity = 65;  
CMD_SlowDown = 66;  
CMD_CwSpeed = 67;  
CMD_CcwSpeed = 68;  
CMD_SetStart = 69;  
CMD_GetIndex = 70;  
CMD_IncIndex = 71;  
CMD_Busy = 72;
```

DEVELOPMENT OF A TOOL CHANGING ROBOT

```
FileName = 'b:settings.dat';           {NAME OF DATA SETTINGS FILE}
Steps_per_Degree : INTEGER = 70;       {NO OF STEPS OF MOTOR PER}
                                         {DEGREE}
```

TYPE

{DECLARATION OF TABLE FOR CONTROL OF REGISTERS FOR PCL-838 DRIVER}

```
FunTable = RECORD
    funNo : INTEGER;
    Channel : INTEGER;
    buffer : ^BYTE;
    vi,vf :INTEGER;
    time,step :LONGINT;
    data : INTEGER;
    returnValue : LONGINT
END;                                     {OF RECORD DECLARATION}
```

{DECLARATION OF RECORD WHICH WILL CONTAIN CONTROL SETTINGS FOR ROBOT}

```
Settings = RECORD
    Tool1_pos,
    Tool2_pos,
    Tool3_pos,

    Tool4_pos,
    Tool5_pos,
    Tool6_pos : INTEGER;               {POSITION OF TOOLS IN DEGREES}

    Base, Mill : INTEGER;              {POSITION OF BASE&MILL IN DEGREES}

    Acc_Steps,
    Max_Speed, Pause_Time : INTEGER;  {SPEED CONTROL SETTINGS}
    Input_Method : INTEGER;            {KEYBOARD OR MILL INTERFACE INPUT}
    Mill_Ext : INTEGER;                {MILL AT FULL OR HALF EXTENSION}
    Vert_Sensor : CHAR;                {FOR A SENSOR ON VERTICAL CYLINDER}
END;
```

VAR

{DECLARATION OF GLOBAL VARIABLES FOR SOFTWARE}

```
regs : Registers;
table :FunTable;
Sett : Settings;
Current_pos : INTEGER;                {CUTTENT ROTATIONAL POSITION OF ARM}
Current_Tool : INTEGER;               {NUMBER OF TOOL IN USE}
```

```
New_Tool : INTEGER;           {NUMBER OF NEW TOOL}
Tool_Pos : ARRAY[1..6] of INTEGER; {ARRAY STORING THE RADIAL
                                  LOCATION OF TOOLS}

Base_Pos,
Mill_Pos : INTEGER; {RADIAL LOCATION OF BASE AND MILL IN STEPS}

Current_Output_0 : INTEGER; {CURRENT OUTPUT VALUE OF PORT 0}

Half_Out_Test : BOOLEAN;      {TEST IF ARM IN MID POSITION}
SettingFile : FILE OF Settings; {FILE CONTAINING ALL}
                                   {ADJUSTABLE SETTINGS}

Exit_Var : CHAR; {TERMINATE PROGRAM IF 'X' IS}
                                   {ENTERED}
```

PROCEDURE Get_Settings;

{THIS PROCEDURE READS THE VARIABLE CONTROL SETTINGS FROM THE
FILE} {'SETTINGS.DAT'}
{IF THE FILE DOESN'T EXIST, IT CREATES IT WITH STANDARD SETTINGS}

BEGIN

```
ASSIGN(SettingFile, FileName);
{$i-}                                {TURN OFF ERROR CHECHING}
  Reset(SettingFile);
{$i+}
IF IoResult <> 0 THEN {CHECKS FOR AN ERROR IN OPENING FILE,}
                    {WHICH WILL OCCUR IF THE FILE DOES NOT}
                    {EXIST. IF THERE IS AN ERROR, }
                    {A NEW FILE IS CREATED}
```

BEGIN

```
Sett.Tool1_pos := -30;
Sett.Tool2_pos := -18;
Sett.Tool3_pos := -6;
Sett.Tool4_pos := 6;
Sett.Tool5_pos := 18;
Sett.Tool6_pos := 30;

Sett.Mill := 90;
Sett.Base := 0;
Sett.Acc_Steps := 200;
Sett.Max_Speed := 1000;
Sett.Pause_Time := 3000;
Sett.Input_Method := 1;
Sett.Mill_Ext := 1;
Sett.Vert_Sensor := 'N';
```

```
REWRITE(SettingFile);
WRITE(SettingFile,Sett);
END
```

```

ELSE
  READ(SettingFile,Sett);          {READING DATA IN SETTINGS
FILE}

  CLOSE(SettingFile);
END;

PROCEDURE Change_Settings;

  {THIS PROCEDURE IS USED TO CHANGE THE DATA IN THE FILE
  'SETTINGS.DAT'}

VAR

  Test1      : BOOLEAN; {TESTS INPUT FOR 'X' TO EXIT LOOP}
  Temp1      : STRING;  {TEMPORARY STORAGE FOR NEW VARIABLE INPUT}
  Temp2,err  : INTEGER; {USED IN CHECKING VALIDITY OF DATA}

BEGIN

  REPEAT                                {CONTINUOUS LOOP UNTIL AN 'X' IS ENTERED}
    Test1 := False;
    ClrScr;

    WRITELN('                                CHANGE OF ROBOT VARIABLE
SETTINGS');
    WRITELN;WRITELN;
    WRITELN('CURRENT SETTINGS');
    WRITELN;
    WRITELN('Enter Number To Change Setting');
    WRITELN;
    WRITELN('A. Home Position:
',Sett.Base);
    WRITELN('B. Position of Tool1:
',Sett.Tool1_pos);
    WRITELN('C. Position of Tool2:
',Sett.Tool2_pos);
    WRITELN('D. Position of Tool3:
',Sett.Tool3_pos);
    WRITELN('E. Position of Tool4:
',Sett.Tool4_pos);
    WRITELN('F. Position of Tool5:
',Sett.Tool5_pos);
    WRITELN('G. Position of Tool6:
',Sett.Tool6_pos);
    WRITELN('H. Position of Mill Chuck:
',Sett.Mill);
    WRITELN('I. Maximum Rotational Speed:
',Sett.Max_Speed);
    WRITELN('J. Acceleration Rate:
',Sett.Acc_Steps);

```

```

    Writeln('K. Pause Time Delay:
',Sett.Pause_Time)
    Writeln('L. Input Method:
',Sett.Input_Method);
    Writeln('    1 = Mill Interface      2 = Keyboard');
    Writeln('M. Mill Horizontal Position
',Sett.Mill_Ext);
    Writeln('    1 = Full Extension      2 = Part Extension');
    Writeln;
    Writeln('N. SENSOR ON VERTICAL CYLINDER:
',Sett.Vert_Sensor);
    Writeln('PRESS "X" to Exit');

CASE ReadKey OF           {READS INPUT AND TAKES APPROPRIATE ACTION}

  'A','a': BEGIN
    Write('Enter new Home Position : ');
    Readln(Temp1);
    Val(Temp1,Temp2,err);
    IF (err = 0)           {CHECKS INPUT IS AN INTEGER}
      THEN Sett.Base := Temp2
      ELSE Writeln('ERROR IN INPUT');
    END;

  'B','b': BEGIN
    Write('Enter Tool 1 Position : ');
    Readln(Temp1);
    Val(Temp1,Temp2,err);
    IF err = 0           {CHECKS INPUT IS AN INTEGER}
      THEN Sett.Tool1_pos := Temp2
      ELSE Writeln('ERROR IN INPUT');
    END;

  'C','c': BEGIN
    Write('Enter Tool 2 Position : ');
    Readln(Temp1);
    Val(Temp1,Temp2,err);
    IF err = 0           {CHECKS INPUT IS AN INTEGER}
      THEN Sett.Tool2_pos := Temp2
      ELSE Writeln('ERROR IN INPUT');
    END;

  'D','d': BEGIN
    Write('Enter Tool 3 Position : ');
    Readln(Temp1);

    Val(Temp1,Temp2,err);

    IF err = 0           {CHECKS INPUT IS AN INTEGER}

      THEN Sett.Tool3_pos := Temp2

      ELSE Writeln('ERROR IN INPUT');

    END;

```

```
'E','e': BEGIN

    WRITE('Enter Tool 4 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0                      {CHECKS INPUT IS AN INTEGER}
    THEN Sett.Tool4_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'F','f': BEGIN
    WRITE('Enter Tool 5 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0
    THEN Sett.Tool5_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'G','g': BEGIN
    WRITE('Enter Tool 6 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0
    THEN Sett.Tool6_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'H','h': BEGIN
    WRITE('Enter Mill Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0                      {CHECKS INPUT IS AN INTEGER}
    THEN Sett.Mill := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'I','i': BEGIN
    WRITE('Enter Maximum Rotation Speed : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF (err = 0) AND (Temp2 < 3000)
        {CHECKS INPUT IS AN INTEGER}

        {WITHIN REASONABLE RANGES}

    THEN Sett.Max_Speed := Temp2

    ELSE WRITELN('ERROR IN INPUT');

END;

'J','j': BEGIN

    WRITE('Enter Acceleration Rate : ');
    READLN(Temp1);
```

```

'E','e': BEGIN

    WRITE('Enter Tool 4 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0 {CHECKS INPUT IS AN INTEGER}
    THEN Sett.Tool4_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'F','f': BEGIN
    WRITE('Enter Tool 5 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0
    THEN Sett.Tool5_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'G','g': BEGIN
    WRITE('Enter Tool 6 Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0
    THEN Sett.Tool6_pos := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'H','h': BEGIN
    WRITE('Enter Mill Position : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF err = 0 {CHECKS INPUT IS AN INTEGER}
    THEN Sett.Mill := Temp2
    ELSE WRITELN('ERROR IN INPUT');
END;

'I','i': BEGIN
    WRITE('Enter Maximum Rotation Speed : ');
    READLN(Temp1);
    VAL(Temp1,Temp2,err);
    IF (err = 0) AND (Temp2 < 3000)
    {CHECKS INPUT IS AN INTEGER}

    {WITHIN REASONABLE RANGES}

    THEN Sett.Max_Speed := Temp2

    ELSE WRITELN('ERROR IN INPUT');

END;

'J','j': BEGIN

    WRITE('Enter Acceleration Rate : ');
    READLN(Temp1);

```

```

        VAL(Temp1,Temp2,err);

        IF (err = 0) AND          {CHECKS INPUT IS AN INTEGER}
        (Temp2 < 480) AND (Temp2 > 20) {WITHIN REASONABLE RANGES}

        THEN Sett.Acc_Steps := Temp2

        ELSE WRITELN('ERROR IN INPUT');
        END;

    'K','k': BEGIN
        WRITE('Enter Pause Time Delay (ms) : ');
        READLN(Temp1);
        VAL(Temp1,Temp2,err);
        IF (err = 0) AND          {CHECKS INPUT IS AN INTEGER}
        (Temp2 > 1000) AND (Temp2 < 8000) {WITHIN REASONABLE RANGES}
        THEN Sett.Pause_Time := Temp2
        ELSE WRITELN('ERROR IN INPUT');
        END;

    'L','l': BEGIN

        WRITE('Enter Input Method : ');

        READLN(Temp1);
        VAL(Temp1,Temp2,err);
        IF (err = 0) AND          {CHECKS INPUT IS AN INTEGER}
        (Temp2 < 3) AND (Temp2 > 0)    {WITHIN REASONABLE RANGES}
        THEN Sett.Input_Method := Temp2
        ELSE WRITELN('ERROR IN INPUT');
        END;

    'M','m': BEGIN
        WRITE('Enter Horizontal Extension Parameter : ');
        READLN(Temp1);
        VAL(Temp1,Temp2,err);
        IF (err = 0) AND          {CHECKS INPUT IS AN INTEGER}
        (Temp2 < 3) AND (Temp2 > 0)    {WITHIN REASONABLE RANGES}
        THEN Sett.Milli_Ext := Temp2
        ELSE WRITELN('ERROR IN INPUT');
        END;

    'N','n': BEGIN
        WRITE('Is there a sensor on the Vertical Cylinder (Y
/ N) : ');
        IF UPCASE(ReadKey) = 'Y' THEN
            Sett.Vert_Sensor := 'Y';
        IF UPCASE(ReadKey) = 'N' THEN
            Sett.Vert_Sensor := 'N';
        END;

    'X','x' : Test1 := True

```

END

```
UNTIL TEST1;                                {UNTIL 'X' ENTERED}

{WRITING DATA TO FILE}
  REWRITE(SettingFile);
  WRITE(SettingFile, Sett);
  CLOSE(SettingFile);

END;
```

PROCEDURE Base_Active;

{INITIATES CONTROL FOR THE PCL-838 CARD}

BEGIN

```

table.funNo := CMD_Base;
table.data := $02e0;
regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);

table.funNo := CMD_Active;
table.channel := _CH1;
regs.ox := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);

```

END;

PROCEDURE Check1;

{PROCEDURE TO CHECK SOFTWARE DRIVER 'PCL-838' IS INSTALLED}

VAR

installStatus : INTEGER;

BEGIN

```

regs.ax := $d200;
intr($2f,regs);
InstallStatus := regs.ax;
WRITELN(installStatus);READLN;

```

```

IF installStatus <> 255
THEN
BEGIN

```

```

    Writeln('You must install the software driver "PCL-
838.EXE"');

```

```

    HALT;

```

```

END;

```

END;

PROCEDURE Set_Posn_Steps;

{CALCULATES POSITIONS OF STOPS IN STEP CO ORDINATES}

BEGIN

```
Base_pos := TRUNC(Seit.Base * Steps_per_Degree);
Mill_Pos := TRUNC(Seit.Mill * Steps_per_Degree);
Tool_Pos[1] := TRUNC(Seit.Tool1_Pos*Steps_per_Degree);
Tool_Pos[2] := TRUNC(Seit.Tool2_Pos*Steps_per_Degree);
Tool_Pos[3] := TRUNC(Seit.Tool3_Pos*Steps_per_Degree);
Tool_Pos[4] := TRUNC(Seit.Tool4_Pos*Steps_per_Degree);
Tool_Pos[5] := TRUNC(Seit.Tool5_Pos*Steps_per_Degree);
Tool_Pos[6] := TRUNC(Seit.Tool6_Pos*Steps_per_Degree);
```

END;

PROCEDURE Delay(Time : INTEGER);

{TO CREATE A PAUSE IN THE ROBOT SEQUENCE}

BEGIN

```
table.FunNo :=CMD_Delay;
table.data := Time;

regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);
```

END;

BEGIN

```
Base_pos := TRUNC(Sett.Base * Steps_per_Degree);
Mill_Pos := TRUNC(Sett.Mill * Steps_per_Degree);
Tool_Pos[1] := TRUNC(Sett.Tool1_Pos*Steps_per_Degree);
Tool_Pos[2] := TRUNC(Sett.Tool2_Pos*Steps_per_Degree);
Tool_Pos[3] := TRUNC(Sett.Tool3_Pos*Steps_per_Degree);
Tool_Pos[4] := TRUNC(Sett.Tool4_Pos*Steps_per_Degree);
Tool_Pos[5] := TRUNC(Sett.Tool5_Pos*Steps_per_Degree);
Tool_Pos[6] := TRUNC(Sett.Tool6_Pos*Steps_per_Degree);
```

END;

PROCEDURE Delay(Time : INTEGER);

{TO CREATE A PAUSE IN THE ROBOT SEQUENCE}

BEGIN

```
table.FunNo :=CMD_Delay;
table.data := Time;

regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);
```

END;

{STATEMENT LIST PROCEDURES FOR ARM CONTROL}

{EACH PROCEDURE IS A POSITION OF THE ROBOT ARM}

PROCEDURE Down_Close_In;

BEGIN

```

    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 4;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
    Current_Output_0 := table.data;
    Delay(Sett.Pause_Time);

```

END;

PROCEDURE Down_Close_Out;

BEGIN

```

    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 8;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
    Current_Output_0 := table.data;
    Delay(Sett.Pause_Time);

```

END;

PROCEDURE Down_Open_In;

BEGIN

```

    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 6;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
    Current_Output_0 := table.data;
    Delay(Sett.Pause_Time);

```

END;

```
PROCEDURE Down_Open_Out;
```

```
BEGIN
```

```
    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 10;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
```

```
    Current_Output_0 := table.data;
    Delay(Seet.Pause_Time);
END;
```

```
PROCEDURE Up_Close_In;
```

```
BEGIN
```

```
    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 5;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
```

```
    intr($2f,regs);
    Current_Output_0 := table.data;
    Delay(Seet.Pause_Time);
END;
```

```
PROCEDURE Up_Close_Out;
```

```
BEGIN
```

```
    table.funNo := CMD_Output;
    table.channel := 0;
    table.data := 9;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
```

```
    Current_Output_0 := table.data;
    Delay(Seet.Pause_Time);
END;
```

```
PROCEDURE Up_Open_In;
```

```
BEGIN
```

```
    table.funNo := CMD_Output;
    table.channel := 0;
```

```
table.data := 7;  
regs.ax := $d202;  
regs.bx := ofs(table);  
regs.cx := seg(table);  
intr($2f,regs);  
Current_Output_0 := table.data;  
Delay(Seet.Pause_Time);  
END;
```

```
PROCEDURE Up_Open_Out;
```

```
BEGIN
```

```
table.funNo := CMD_Output;  
table.channel := 0;  
table.data := 11;  
regs.ax := $d202;  
regs.bx := ofs(table);  
regs.cx := seg(table);  
intr($2f,regs);  
Current_Output_0 := table.data;  
Delay(Seet.Pause_Time);  
END;
```

{PROCEDURES TO CONTROL HORIZONTAL POSITION TO HALF OUT}

```

PROCEDURE Down_Close_Half;

VAR

Out_0_Temp  : INTEGER;           {TEMP OUTPUT TO PORT 0}

BEGIN

    IF Current_Output_0 > 3 THEN    {CHECKS IF ARM IS IN MID}
                                    {POSITION}

BEGIN

    IF Current_Output_0 > 7 THEN    {IF ARM IS OUT}
        Out_0_Temp := Current_Output_0 - 4
    ELSE
        Out_0_Temp := Current_Output_0 + 4;    {IF ARM IS IN}

    table.funNo := CMD_Output;      {OUTPUT TO MOVE ARM}
    table.channel := 0;
    table.data := Out_0_Temp;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);

REPEAT                            {LOOP UNTIL MID PROXIMITY SWITCH IS MADE}

    table.funNo := CMD_Input;    {READING VALUES OF INPUT PORT}
    table.channel := 0;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);
    UNTIL NOT ODD(table.data)
END;

    table.funNo := CMD_Output;    {OUTPUT TO HOLD ARM IN MID}
                                    {POSITION}

    table.channel := 0;
    table.data := 0;
    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);

```

```

Current_Output_0 := table.data;

Delay(Se tt.Pause_Time);

END;

PROCEDURE Up_Close_Half;

VAR
    Out_0_Temp  : INTEGER;      {Temp Output to Port 0}

BEGIN

    IF Current_Output_0 > 3 THEN

        BEGIN

            IF Current_Output_0 > 7 THEN
                Out_0_Temp := Current_Output_0 - 4
            ELSE
                Out_0_Temp := Current_Output_0 + 4;

            table.funNo := CMD_Output;
            table.channel := 0;
            table.data := Out_0_Temp;
            regs.ax := $d202;
            regs.bx := ofs(table);
            regs.cx := seg(table);
            intr($2f,regs);

            REPEAT
                table.funNo := CMD_Input;
                table.channel := 0;
                regs.ax := $d202;
                regs.bx := ofs(table);
                regs.cx := seg(table);
                intr($2f,regs);
            UNTIL NOT ODD(table.data)
        END;

        table.funNo := CMD_Output;
        table.channel := 0;
        table.data := 1;
        regs.ax := $d202;
        regs.bx := ofs(table);
        regs.cx := seg(table);
        intr($2f,regs);

        Current_Output_0 := table.data;

        Delay(Se tt.Pause_Time);

    END;

```

```
PROCEDURE Down_Open_Half;

VAR

    Out_0_Temp : INTEGER;      {Temp Output to Port 0}

BEGIN

    IF Current_Output_0 > 3 THEN

        BEGIN

            IF Current_Output_0 > 7 THEN
                Out_0_Temp := Current_Output_0 - 4
            ELSE
                Out_0_Temp := Current_Output_0 + 4;

            table.funNo := CMD_Output;
            table.channel := 0;
            table.data := Out_0_Temp;
            regs.ax := $d202;
            regs.bx := ofs(table);
            regs.cx := seg(table);
            intr($2f,regs);

            REPEAT
                table.funNo := CMD_Input;
                table.channel := 0;
                regs.ax := $d202;
                regs.bx := ofs(table);
                regs.cx := seg(table);
                intr($2f,regs);
            UNTIL NOT ODD(table.data)

        END;

        table.funNo := CMD_Output;
        table.channel := 0;
        table.data := 2;
        regs.ax := $d202;
        regs.bx := ofs(table);
        regs.cx := seg(table);
        intr($2f,regs);

        Current_Output_0 := table.data;

        Delay(Seet.Pause_Time);

    END;
```

```
PROCEDURE Up_Open_Half;

VAR

    Out_0_Temp  : INTEGER;      {Temp Output to Port 0}

BEGIN

    IF Current_Output_0 > 3 THEN

        BEGIN

            IF Current_Output_0 > 7 THEN
                Out_0_Temp := Current_Output_0 - 4
            ELSE
                Out_0_Temp := Current_Output_0 + 4;

            table.funNo := CMD_Output;
            table.channel := 0;
            table.data := Out_0_Temp;
            regs.ax := $d202;
            regs.bx := ofs(table);
            regs.cx := seg(table);
            intr($2f,regs);

            REPEAT
                table.funNo := CMD_Input;
                table.channel := 0;
                regs.ax := $d202;
                regs.bx := ofs(table);
                regs.cx := seg(table);
                intr($2f,regs);
            UNTIL NOT ODD(table.data)
            END;

            table.funNo := CMD_Output;
            table.channel := 0;
            table.data := 3;
            regs.ax := $d202;
            regs.bx := ofs(table);
            regs.cx := seg(table);
            intr($2f,regs);

            Current_Output_0 := table.data;

            Delay(Sett.Pause_Time);

        END;

    END;
```

```

PROCEDURE Rotate(VAR Current_Pos,New_Pos : INTEGER);
{ROTATES THE ARM TO THE NEW POSITION, FROM CURRENT_POS TO NEW_POS}

VAR

    Steps_to_rotate :INTEGER;
    Direction       :INTEGER;

BEGIN

    {INITIAL CALCS AND DIRN CHECKS}

    Steps_to_Rotate := New_Pos - Current_pos;

    IF Steps_to_Rotate > 0
        THEN Direction := _CW1

        ELSE Direction := _CCW1;

    Steps_to_Rotate := ABS(Steps_to_Rotate);

    {CLEAR ANY PREVIOUS CURVES}

    table.funNo := CMD_Clear;
    table.channel := _CH1;

    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);

    intr($2f,regs);

```

```

{CHECK NEW POS <> OLD POS}

IF Steps_To_Rotate <> 0 THEN BEGIN

    {ACCELERATE}

    table.funNo := CMD_SetStep;
    table.channel := _CH1 or Direction;
    table.vi := 0;
    table.vf := Sett.Max_Speed;
    table.step := Sett.Acc_Steps;

    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);
    intr($2f,regs);

    {CONST VELOCITY PHASE}

    table.funNo := CMD_SetStep;
    table.channel := _CH1 or Direction;
    table.vi := Sett.Max_Speed;
    table.vf := Sett.Max_Speed;
    table.step := Steps_to_Rotate - 2*Sett.Acc_Steps;

    regs.ax := $d202;
    regs.bx := ofs(table);
    regs.cx := seg(table);

    intr($2f,regs);

    {DECELERATION PHASE}

    table.funNo := CMD_SetStep;
    table.channel := _CH1 or Direction;

    table.vi := Sett.Max_Speed;
    table.vf := 0;

    table.step := Sett.Acc_Steps;

    regs.ax := $d202;

    regs.bx := ofs(table);
    regs.cx := seg(table);

    intr($2f,regs);

```

{TRANSFER DATA TO DRIVER CARD}

```
table.funNo := CMD_Transfer;
table.channel := _CH1;
regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);
```

{RUN CURVES}

```
table.funNo := CMD_Run;
table.channel := _CH1;
regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
intr($2f,regs);
```

END;

Current_Pos := New_Pos;

Delay(Steps_to_Rotate + Sett.Pause_Time);

END;

PROCEDURE Tool_Number;

{GETS INPUT OF TOOL NUMBER FROM EITHER THE PCL-838}
{INPUT PORT (3-BIT) OR KEYBOARD}

VAR

Temp1 : STRING; {TEMPORARY STORAGE FOR TOOL NUMBER}

```

Num1, Num2, Count, Temp_x :INTEGER;
err      :INTEGER;          {FOR VALIDITY CHECK -KEYBOARD INPUT}

BEGIN

  ClrScr;
  Writeln('                          Tool Change Robot for CNC Milling
Machine');
  Writeln;Writeln;Writeln;
  Writeln('Press "x" to Exit');
  Writeln;
  Writeln('Press "  " to Modify Control Settings');
  Writeln;Writeln;

  Exit_Var := 'a';

  CASE Sett.Input_Method OF

  1: BEGIN                      {IF INPUT_METHOD = 1, INPUT FROM MILL}

    Writeln('  Automatic Mode');

    Writeln;

    Count := 0;

    REPEAT                      {LOOP TO READ DATA FROM PORT 0}

      table.funNo := CMD_Input;  {UNTIL A VALID NUMBER IS INPUT}

      table.channel := 0;

      regs.ax := $d202;
      regs.bx := ofs(table);
      regs.cx := seg(table);
      intr($2f,regs);

      IF KeyPressed THEN        {CHECKS IF 'X' IS PRESSED TO EXIT}
        Exit_Var := ReadKey;

      IF Table.Data = Temp_x THEN

        Count := Count + 1

      ELSE

        Count := 0;

      Temp_x := Table.Data;

    UNTIL ((table.data < 223) AND (Count = 10))

      OR (UPCASE(Exit_Var) = 'X') OR (UPCASE(Exit_Var) = 'c');

```

```

IF UPCASE(Exit_Var) <> 'X' THEN

    BEGIN

        Num1 := table.data SHR
        2;      {CALCULATING VALUE
        OF} {BITS 3 - 5}
        {FROM 8 BIT INPUT}

        Num2 := num1 SHR 3;
        New_Tool := (Num1 - (Num2 SHL 3));

        Writeln;

        Writeln('New Tool No: ',New_Tool,' Requested')

    END

END;

2: BEGIN          {INPUT_METHOD= 2, INPUT FROM KEYBOARD}

    Writeln('Enter the New Tool Number (1 - 6)');

    REPEAT          {LOOP WAITING FOR VALID ENTRY}
        Readln(Temp1);

        IF (Temp1 = 'X') OR (Temp1 = 'x') {ENTERING AN 'X' WILL
EXIT} THEN Exit_Var := 'X'
        ELSE VAL(Temp1,New_Tool,err);

        IF (err <> 0) OR (New_Tool < 1) {VALIDITY CHECKING}
        OR (New_Tool > 6) THEN

            Writeln('Error - enter a Number between 0 and 6');

        UNTIL ((err = 0) AND (New_Tool > 0) AND (New_Tool < 7))
            OR (UPCASE(Exit_Var) = 'X') OR (UPCASE(Exit_Var) = 'S');

    END;

END {case}

END;

```

PROCEDURE Clamp_Unclamp;

{SENDS OUTPUT TO RELAY 4 FOR 1 SECOND TO ACTIVATE MILL CLAMP}

BEGIN

```

table.funNo := CMD_Output;
table.channel := 0;

table.data := (Current_Output_0 + 16);

regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);

intr($2f,regs);

Delay(1000);

```

```

table.funNo := CMD_Output;
table.channel := 0;
table.data := Current_Output_0;

regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);

intr($2f,regs);

Delay(500);

```

END;

PROCEDURE Waiting;

{SENDS OUTPUT TO RELAY FOR 24V SIGNAL TO MILL TO INDICATE}

{ROBOT IDLE AND JOB COMPLETE}

BEGIN

```
table.funNo := CMD_Output;
table.channel := 0;
table.data := (Current_Output_0 + 32);
```

```
regs.ax := $d202;
regs.bx := ofs(table);
regs.cx := seg(table);
```

```
intr($2f,regs);
```

```
Delay(1000);
```

END;

PROCEDURE Test;

{PROCEDUTE TO TEST SYSTEM FUNCTIONS IN MANUAL MODE, INDIVIDUALLY}

VAR

Value : INTEGER; *{TEMPORART STORAGE OF VARIABLES}*

BEGIN

```
Current_Pos := 0;
```

REPEAT

```
clrscr;
WRITELN('Enter the Number Representing the Function');
Writeln; Writeln;
```

```
WRITELN('1. Down_In_Close');
```

```
WRITELN('2. Up_in_Close');
```

```
WRITELN('3. Up_Out_Cl.  ');
```

```
WRITELN('4. Down_Out_Close');
```

```
WRITELN('5. Down_Out_Open');
```

```
WRITELN('6. Up_Out_Open');
```

```

WRITELN('7. Rotate to Base');
WRITELN('8. Rotate Tool1');
WRITELN('9. Rotate Tool3');
WRITELN('10. Rotate Tool6');
WRITELN('11. Rotate Mill');
WRITELN('12. Down_Close_Half');
WRITELN('13. Down_Open_Half');
WRITELN('14. EXIT');

READLN(Value);

CASE Value OF
                                {USES STATEMENT LIST FUNCTIONS}

1 : Down_Close_In;
2 : Up_Close_In;
3 : Up_Close_Out;
4 : Down_Close_Out;
5 : Down_Open_Out;
6 : Up_Open_out;
7 : Rotate(Current_Pos,Base_Pos);
8 : Rotate(Current_Pos,Tool_Pos[1]);
9 : Rotate(Current_Pos,Tool_Pos[3]);
10: Rotate(Current_Pos,Tool_Pos[6]);
11: Rotate(Current_Pos,Mill_Pos);
12: Down_Close_Half;
13: Down_Open_Half;
14: Writeln('Good Buy');

ELSE Writeln('Error - Try Again')
END {Case}
UNTIL Value = 14
END;
```

```
PROCEDURE Check_Arm_Down;
```

```
{PROCEDURE TO CHECK ARM MOVES DOWN AFTER TOOL IS RELEASED}
```

```
{FROM MILL, INCASE IT GETS STUCK. IN SUCH CASE, THERE IS}  
{AN AUDABLE BEEP}
```

```
VAR
```

```
    Check1    : INTEGER;      {Temp storage for checking sensor value}  
    Test1     : BOOLEAN;
```

```
BEGIN
```

```
REPEAT
```

```
    Test1 := False;  
    Delay(1000);
```

```
    table.funNo := CMD_Input;  
    table.Channel := 0;  
    regs.ax := $d202;  
    regs.bx := ofs(table);  
    regs.cx := seg(table);  
    intr($2f,regs);
```

```
    Check1 := table.data shr 1;
```

```
IF ODD(Check1) THEN
```

```
    BEGIN
```

```
        IF NOT Test1 THEN
```

```
            BEGIN
```

```
                WRITELN('There is a problem with the Arm not moving down');  
                Test1 := True
```

```
            END;
```

```
                SOUND(700);  
                Delay(500);  
                NoSound;  
                Delay(1000);
```

```
        END;
```

```
UNTIL NOT ODD(Check1);
```

```
IF Test1 THEN
```

```
    WRITELN('Problem is resolved');  
END;
```

```

PROCEDURE Get_Current_Tool;

    {MOVEMENT OF ARM TO FETCH TOOL IN USE}

BEGIN
    Down_Close_In;
    Rotate(Current_Pos, Mill_Pos);
    Up_Open_In;

    CASE Sett.Mill_Ext OF

        1 : BEGIN
            Up_Open_Out;
            Up_Close_Out;
            Clamp_Unclamp; {Send Output to Mill to Release }
            Down_Close_Out;
            IF Sett.Vert_Sensor = 'Y' THEN
                Check_Arm_Down
            END;

        2: BEGIN
            Up_Open_Half;
            Up_Close_Half;
            Clamp_Unclamp;
            Down_Close_Half;
            IF Sett.Vert_Sensor = 'Y' THEN
                Check_Arm_Down
            END

        END;

    Down_Close_In;
    Rotate(Current_Pos, Tool_Pos[Current_Tool]);
    Down_Close_Out;
    Down_Open_Out;
    Down_Open_In;
    Down_Close_In;

END;

```

```

Procedure Put_New_Tool;

{MOVEMENT OF ARM TO PUT NEW TOOL ON MILL}

BEGIN

    Rotate(Current_Pos, Tool_Pos[New_Tool]);

    Down_Open_In;

    Down_Open_Out;

    Down_Close_Out;

```

```

Down_Close_In;
Rotate(Current_Pos, Mill_Pos);

CASE Sett.Mill_Ext OF

1 : BEGIN
    Down_Close_Out;
    Up_Close_Out;
    Clamp_Unclamp;  {Output to Mill to grip tool}
    Up_Open_Out;
    END;

2 : BEGIN
    Down_Close_Half;
    Up_Close_Half;
    Clamp_Unclamp;  {Output to Mill to grip tool}
    Up_Open_Half;
    END

END;

Up_Open_In;
Down_Close_In;
Rotate(Current_Pos, Base_Pos);
Current_Tool := New_Tool
END;
```



```

REPEAT
  Test1 := True;

  Ch1 := Readkey;

CASE Upcase(Ch1) OF

  'Y' : BEGIN
    Test;

    Test1 := False;

    END;

  'N' : Test1 := False

  END;

UNTIL NOT Test1;

Current_Pos := 0;
Writeln;Writeln;
WRITELN('Is there currently a Tool in Use (Y / N) ?');

REPEAT
  Test1 := TRUE;
  Ch := Readkey;

CASE Upcase(Ch) OF

  'Y' : BEGIN
    Write('Please Enter Tool Number (1-6)');
    REPEAT
      Readln(Tool);
      IF (Tool < 1) OR (Tool > 6) THEN
        BEGIN
          Writeln;
          WRITELN('Error - Please re-enter Tool Number');
          Writeln;
        END;
      UNTIL (Tool > 0) AND (Tool < 7);
      Current_Tool := Tool
    END;

  'N' : Current_Tool := 0;

  ELSE Test1 := FALSE;

END; {CASE}
UNTIL Test1;
END;

```

{MAIN PROGRAM BLOCK}

BEGIN

Exec('c:\robot\prog838\pcl-838.exe',''); {LOAD SOFTWARE DRIVER
PCL-838}

Check1; {CHECK SOFTWARE INSTALLED}

Base_Active; {SETS THE CARDS BASE ADDRESS}
{AND ACTIVATES CHANNEL 1}

Down_Close_In; {ENSURES ARM IN HOME POSITION}

Get_Settings; {LOADS VARIABLE SETTINGS}

Current_Pos := 0;

Welcome; {INTRODUCTION SCREEN}

Set_Posn_Steps; {CALCULATES RADIAL POSITIONS IN}
{STEP COORDINATES}

Half_Out_Test := False;

REPEAT

{LOOP WHICH GETS TOOL NUMBER AND EXECUTES TOOL CHANGE}

Waiting; {OUTPUT TO INDICATE NOT BUSY TO MILL}

Tool_Number; {CONTINUOUS LOOP UNTIL 3-BIT INPUT}
{FOR TOOL NO}

IF (New_Tool <> Current_Tool) THEN {CHECK NEW TOOL IS NOT}
{CURRENT ONE}

BEGIN

IF (Current_Tool <> 0) AND (UPCASE(Exit_Var) <> 'X') THEN
{CHECKS A TOOL IS IN USE}

```

BEGIN

    WRITELN('Getting tool');

    Get_Current_Tool;          {PROCEDURE TO GET TOOL FROM MILL}

END;

WRITELN('Putting Tool');

    Put_New_Tool;              {PROCEDURE TO PUT NEW TOOL ON MILL}

END;

IF UPCASE(Exit_Var) = 'S' THEN

    BEGIN

        Change_Settings;

        Set_Posn_Steps;;

    END;

UNTIL UPCASE(Exit_Var) = 'X';    {EXITS LOOP IF A 'X' IS ENTERED}

Regs.ax := $d201;                {UN-INSTALLS PCL-838}
                                   {SOFTWARE DRIVER}

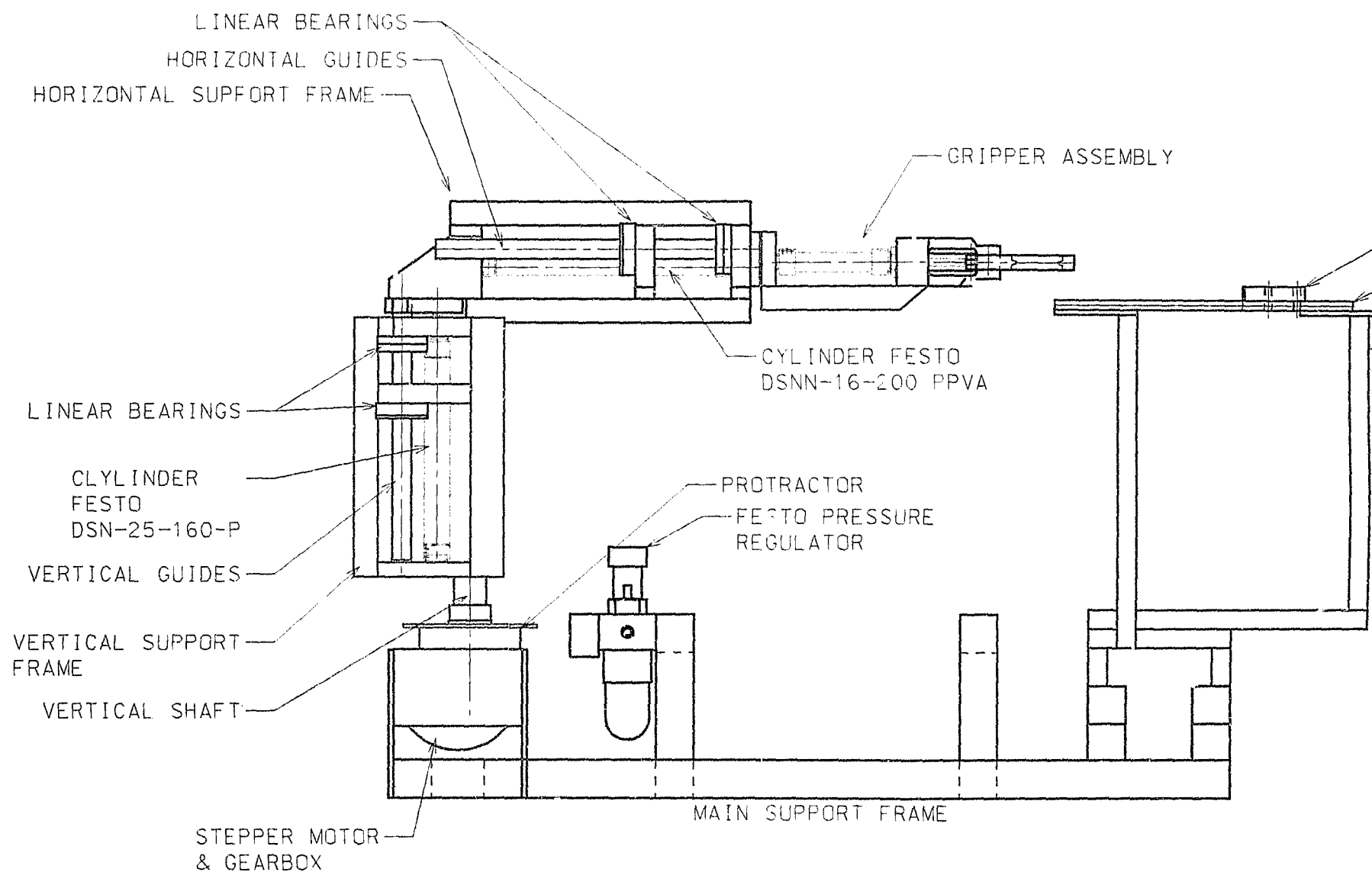
Intr($2f,regs)

END.

```

APPENDIX B

GENERAL ASSEMBLY DRAWINGS

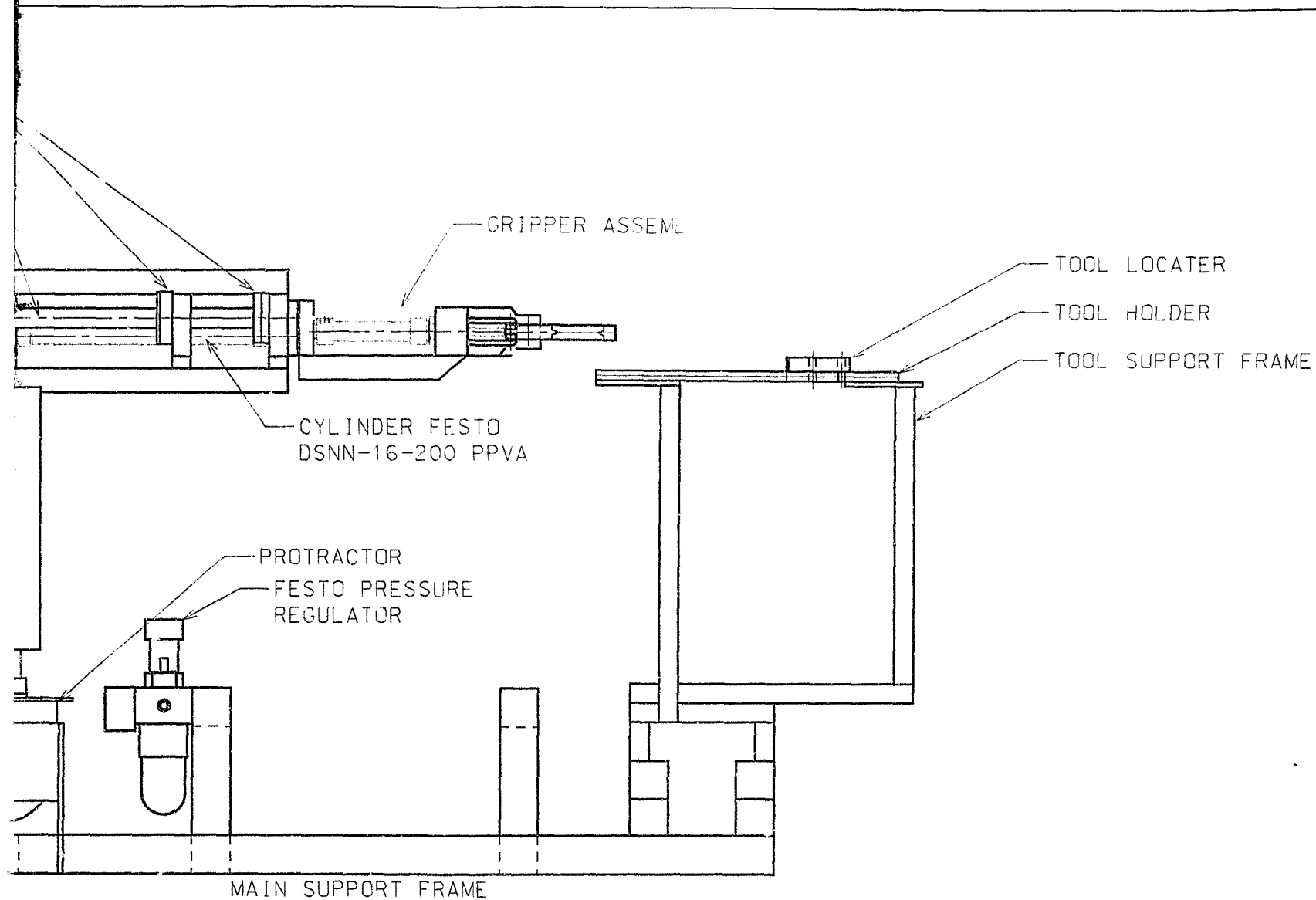


ELEVATION ON ROBOTIC ARM TOOL SELECTOR

TITLE:

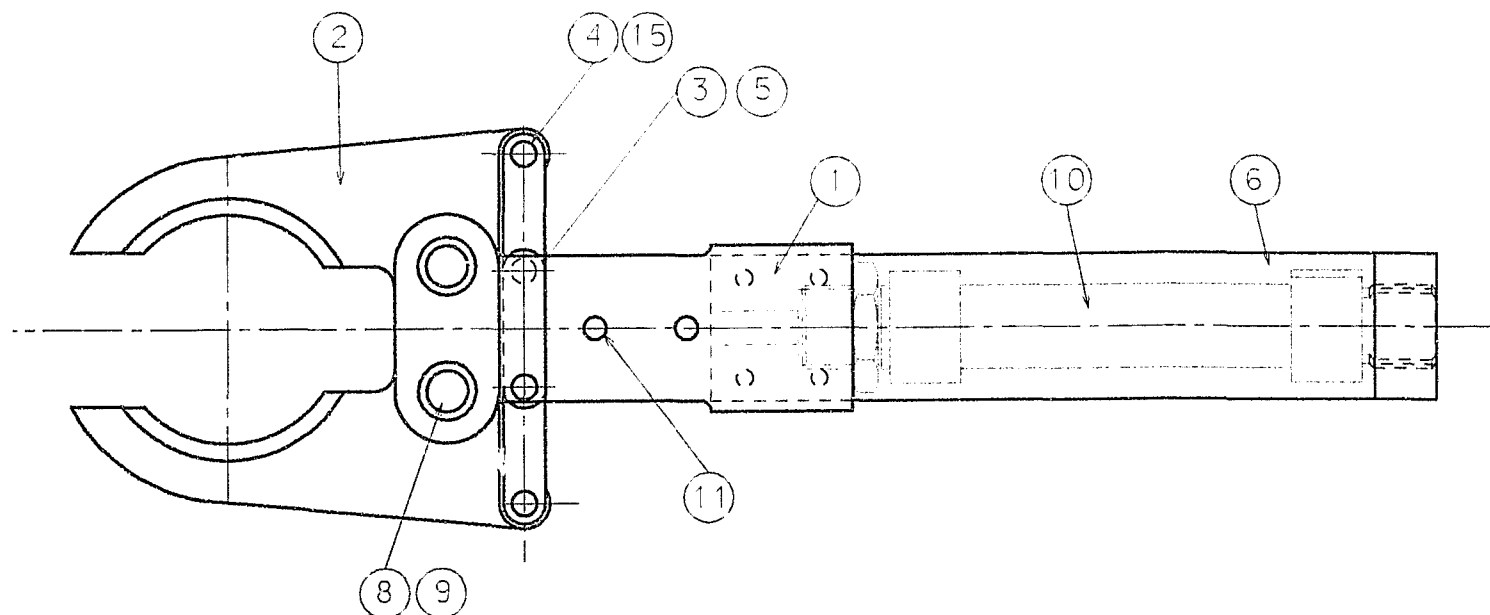
DRAWN

SCALE

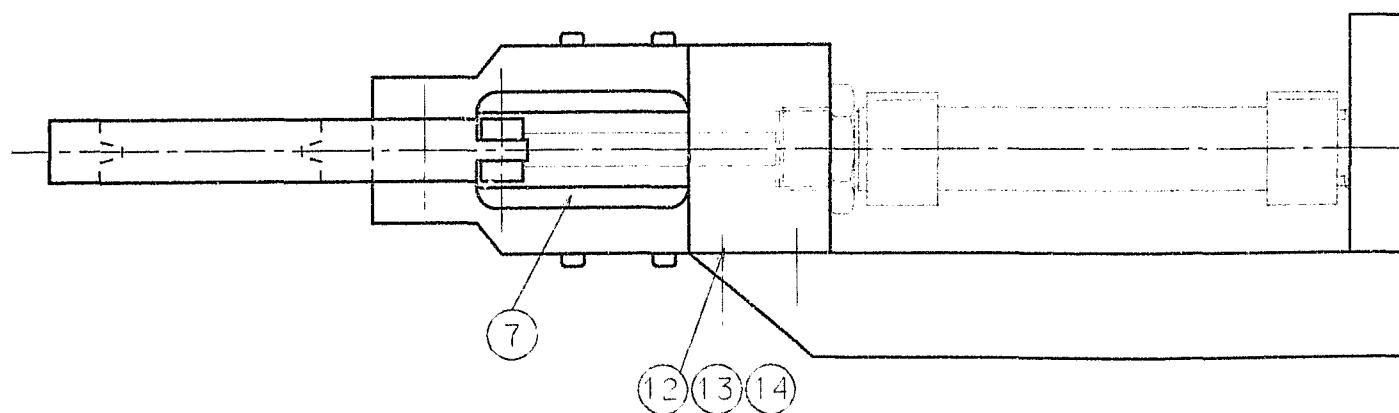


ATION ON ROBOTIC ARM TOOL SELECTOR

TITLE: ROBOTIC ARM TOOL SELECTOR (PREVIOUS UNIT)			
DRAWN		DRG. NO.	REV
SCALE	1:5		



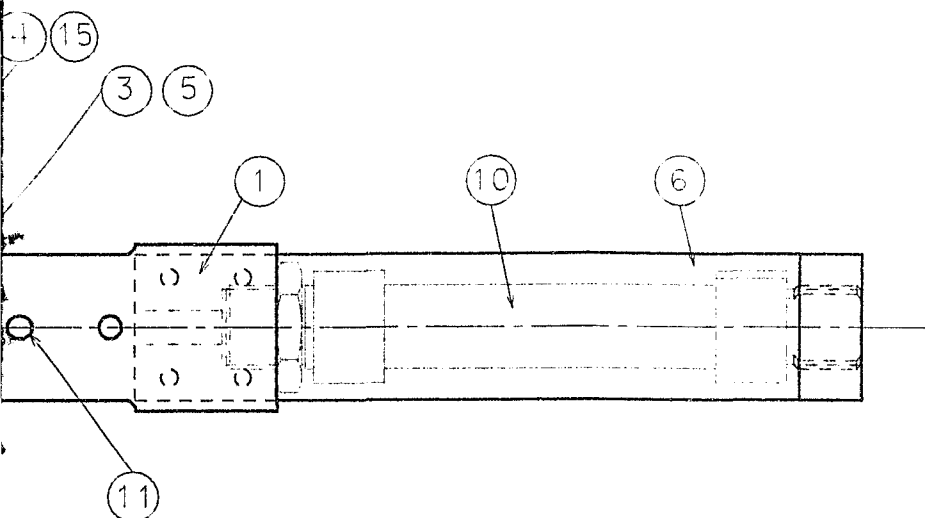
PLAN



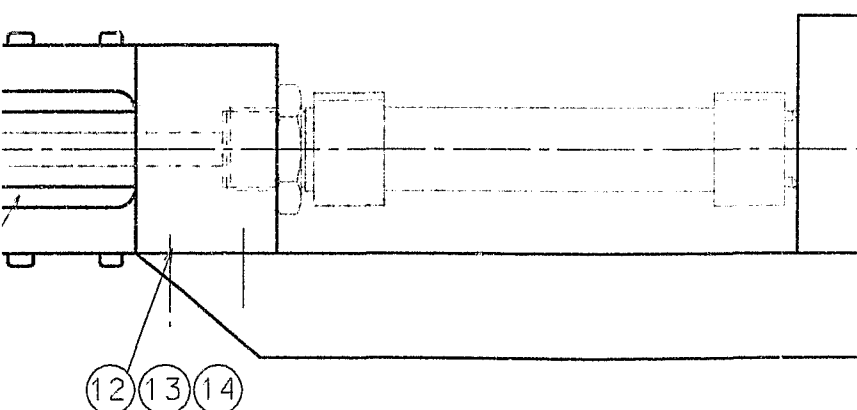
ELEVATION ON ROBOT GRIPPER

15	FINGER PIN
14	WASHER, LOCK
13	WASHER, FLAT
12	SCREW CAP SOCKET H
11	SCREW CAP SOCKET H
10	CYLINDER, PNEUMAT
9	SWIVEL BUSH
8	SWIVEL PIN
7	PLATE GUIDE
6	GRIPPER BRACKET
5	PUSHER BEAM PIN
4	STRUT
3	PUSHER BEAM
2	GRIPPER FINGER
1	GRIPPER CHASSIS
ITEM N.O	NAME

TITLE:	
ROBOT GRIPPER	
DRAWN	
SCALE	1:1.5



PLAN



ELEVATION ON ROBOT GRIPPER

15	FINGER PIN	2	Ø5 X 15 LG. BRASS
14	WASHER, LOCK	4	
13	WASHER, FLAT	4	
12	SCREW CAP SOCKET HD.	4	M4 X 15 LG.
11	SCREW CAP SOCKET HD.	4	M3 X 15 LG.
10	CYLINDER, PNEUMATIC	1	DGS-16-40-P FESTO
9	SWIVEL BUSH	4	
8	SWIVEL PIN	2	
7	PLATE GUIDE	2	
6	GRIPPER BRACKET	1	
5	PUSHER BEAM PIN	2	Ø5 X 15 LG. BRASS
4	STRUT	2	
3	PUSHER BEAM	1	
2	GRIPPER FINGER	2	
1	GRIPPER CHASSIS	1	
ITEM N.O	NAME	QUANTITY	DESCRIPTION

TITLE:

ROBOTIC ARM
GRIPPER ASSEMBLY

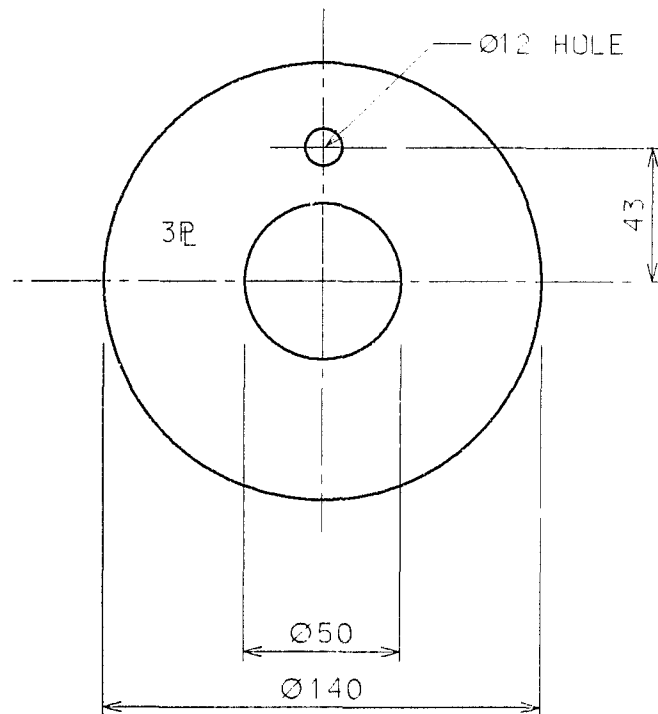
DRAWN

SCALE

1:1.5

DRG. NO.

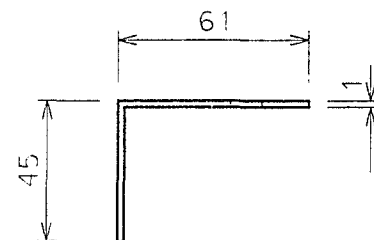
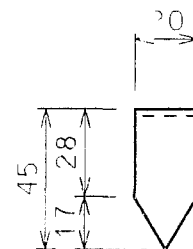
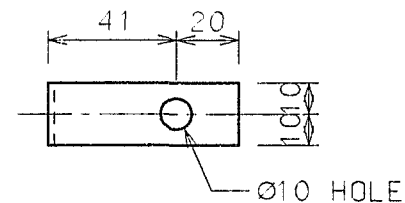
REV



ELEVATION ON PROTRACTOR

MATERIAL ALUMINIUM

ENGRAVE 5mm INDICATORS AT 5°
AROUND THE PLATE



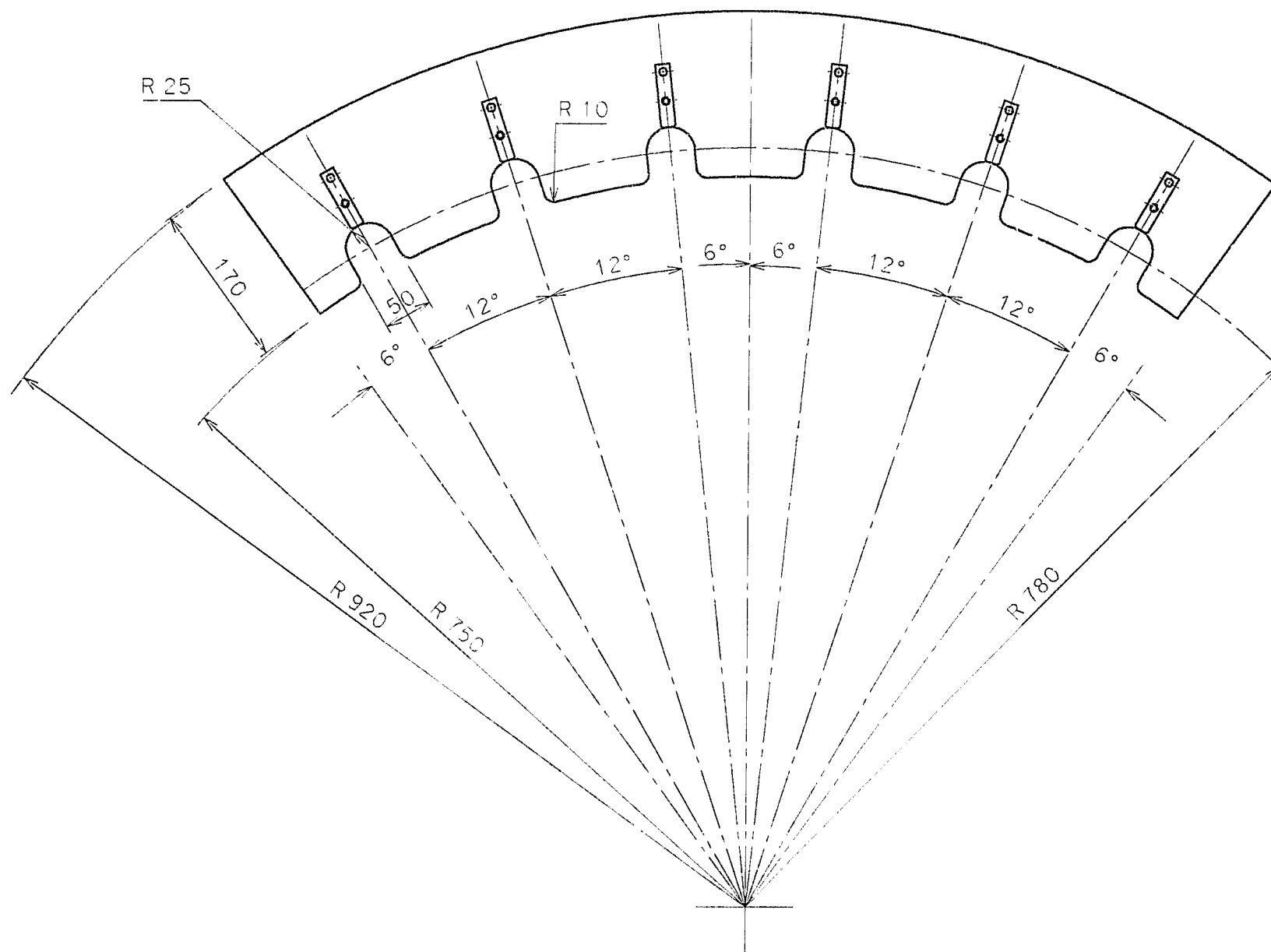
ELEVATION ON INDICATOR

MATERIAL ALUMINIUM

TITLE: ROBOTIC ARM
PROTRACTOR & INDICATOR
DETAIL

DRAWN			DRG. NO.	REV
SCALE				

ALL DIMENSIONS ± 0.3



ELEV.

PLAN ON TOOL HOLDER
MATERIAL ALUMINIUM

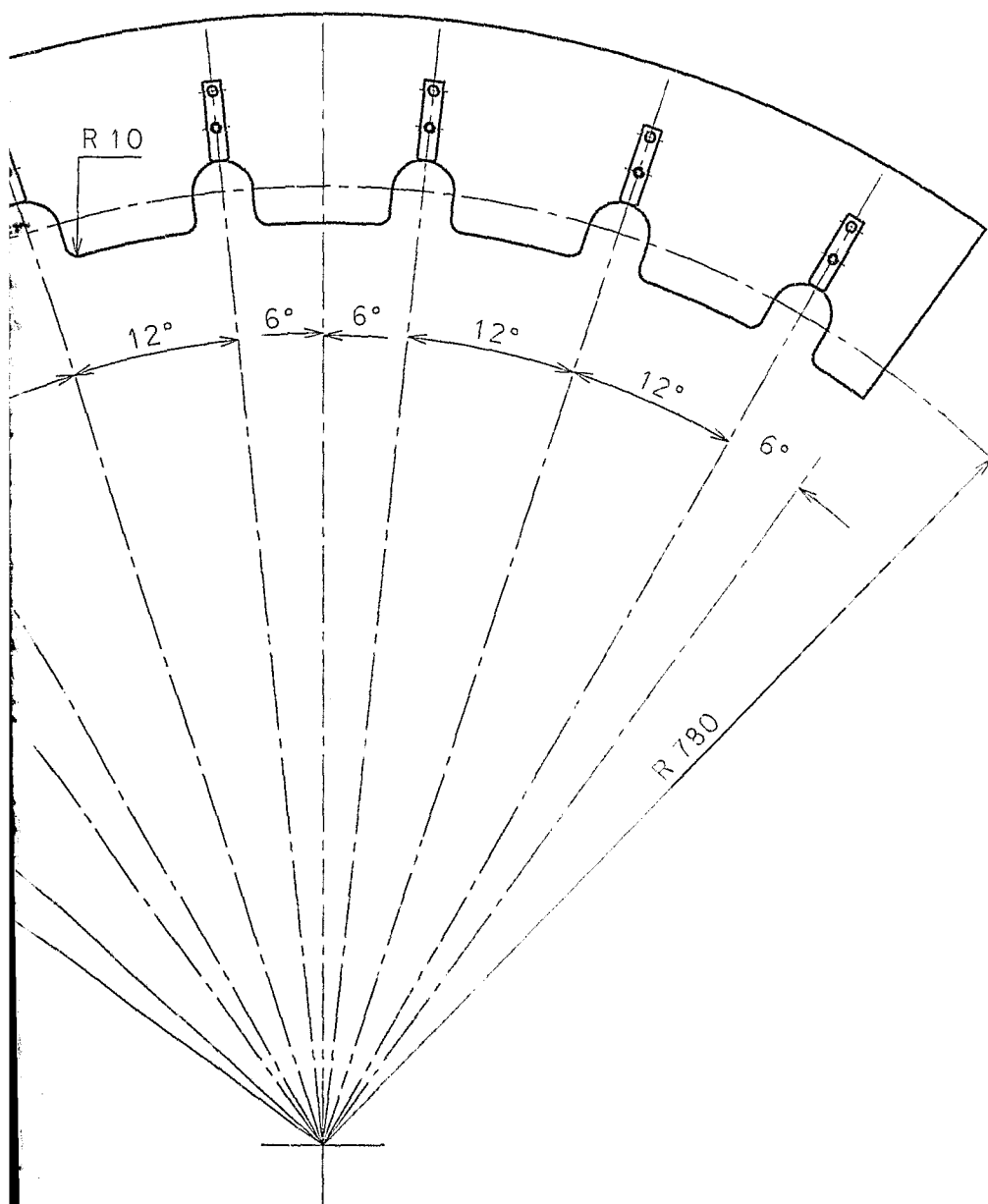
ALL DIMENSIONS ± 0.2

TITLE:

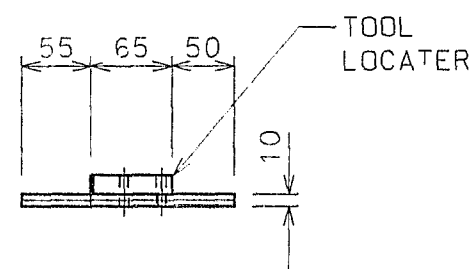
DRAWN

SCALE

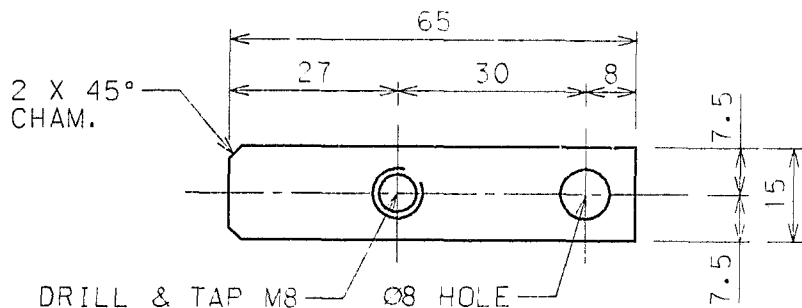
1:



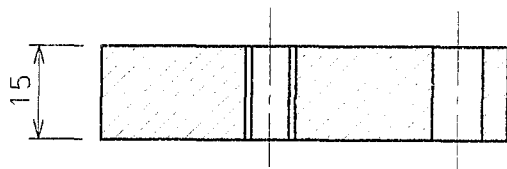
ALL DIMENSIONS ± 0.2



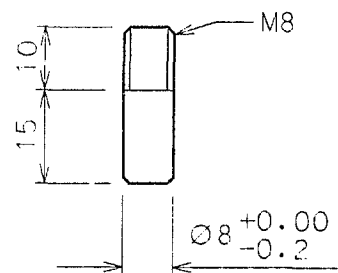
TITLE: ROBOTIC ARM TOOL HOLDER			
DRAWN		DRG. NO.	REV
SCALE	1:5		



PLAN



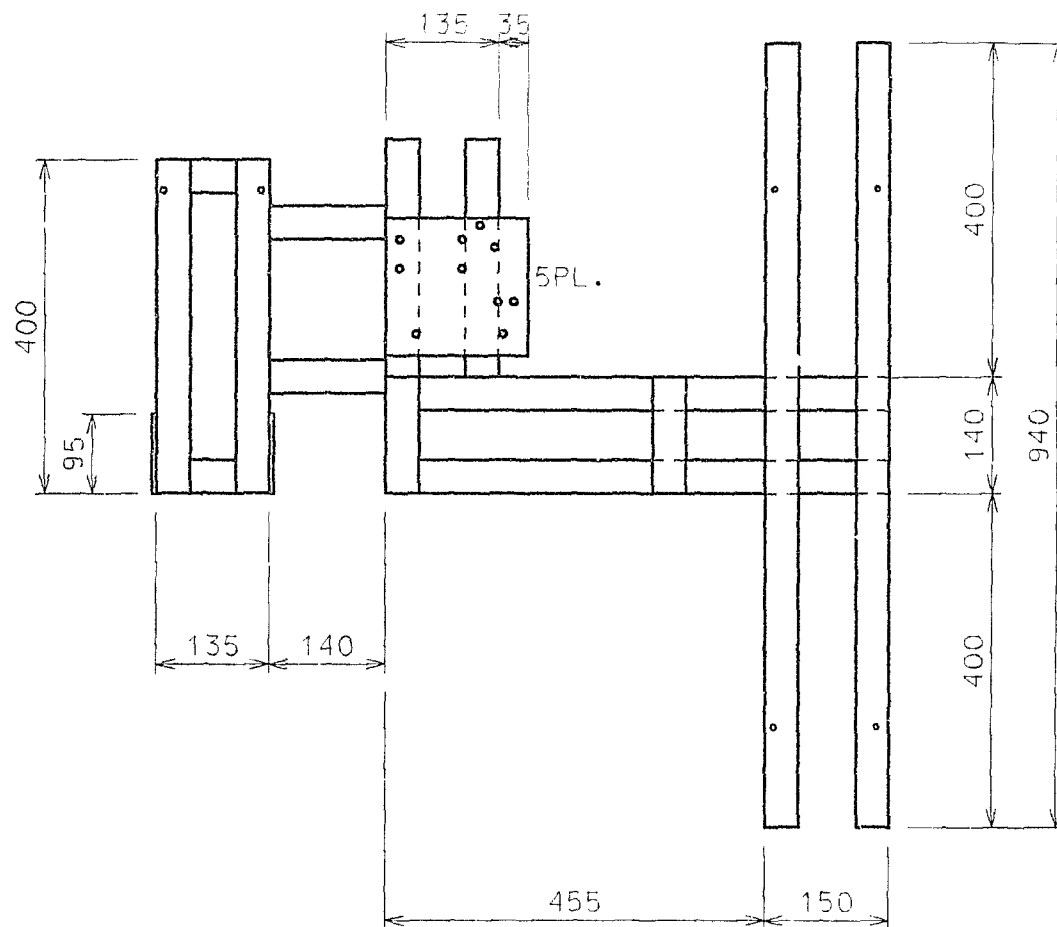
SECTION THRO' TOOL LOCATER
MATERIAL MILD STEEL



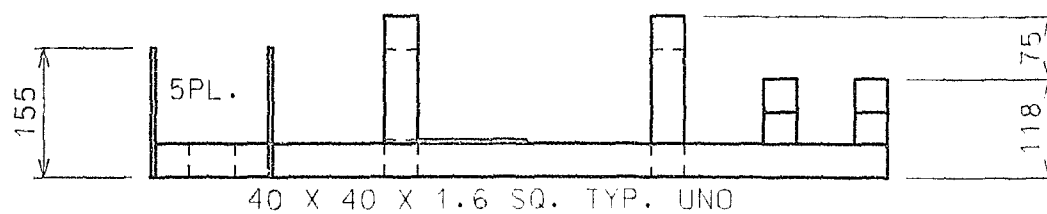
LOCATING PIN
MATERIAL MILD STEEL

TITLE: ROBOTIC ARM TOOL LOCATER DETAIL			
DRAWN		DRG. NO.	REV
SCALE	1 : 1		

TOLERANCE 0.5 UNO



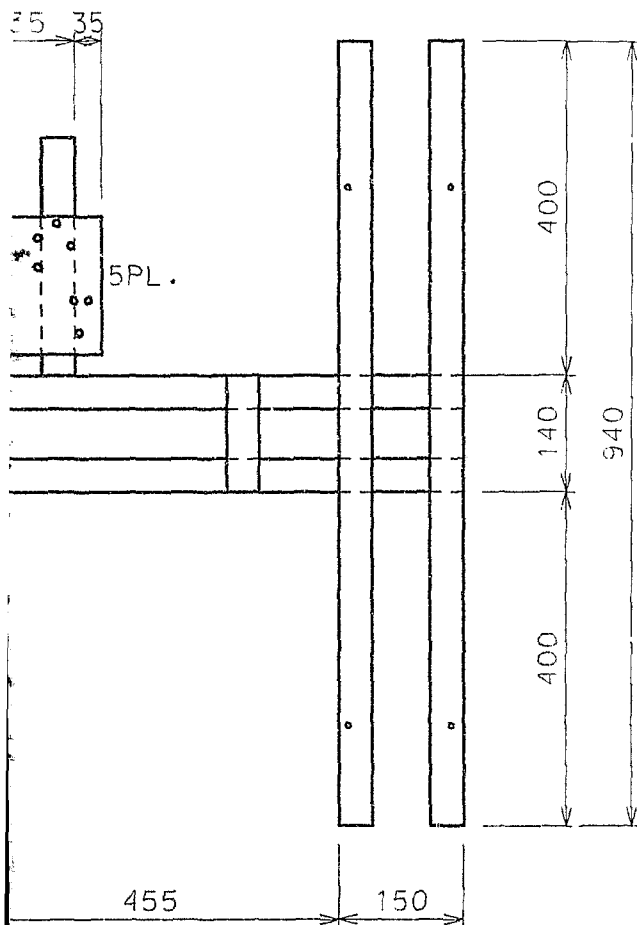
PLAN



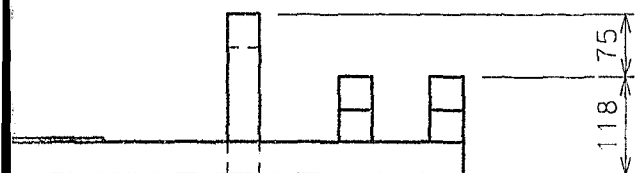
ELEVATION MAIN SUPPORT FRAME

TOLERANCE 0.5 UNO

TITLE:		RO
		MAIN S
DRAWN		
SCALE		1:7.



PLAN

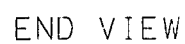


6 SQ. TYP. UNO

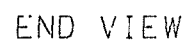
MAIN SUPPORT FRAME

TOLERANCE 0.5 UNO

TITLE: ROBOTIC AR MAIN SUPPORT FRAME			
DRAWN		DRG. NO.	REV
SCALE	1:7.5		

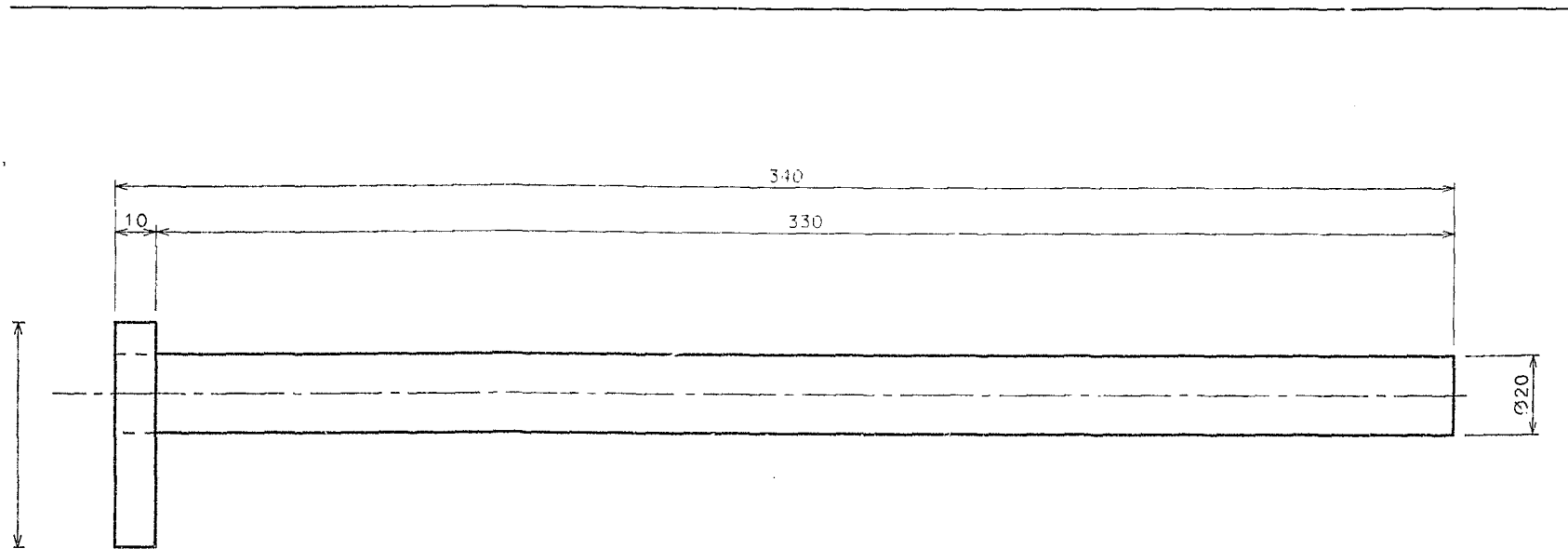


MATERIAL:- MILD STEEL

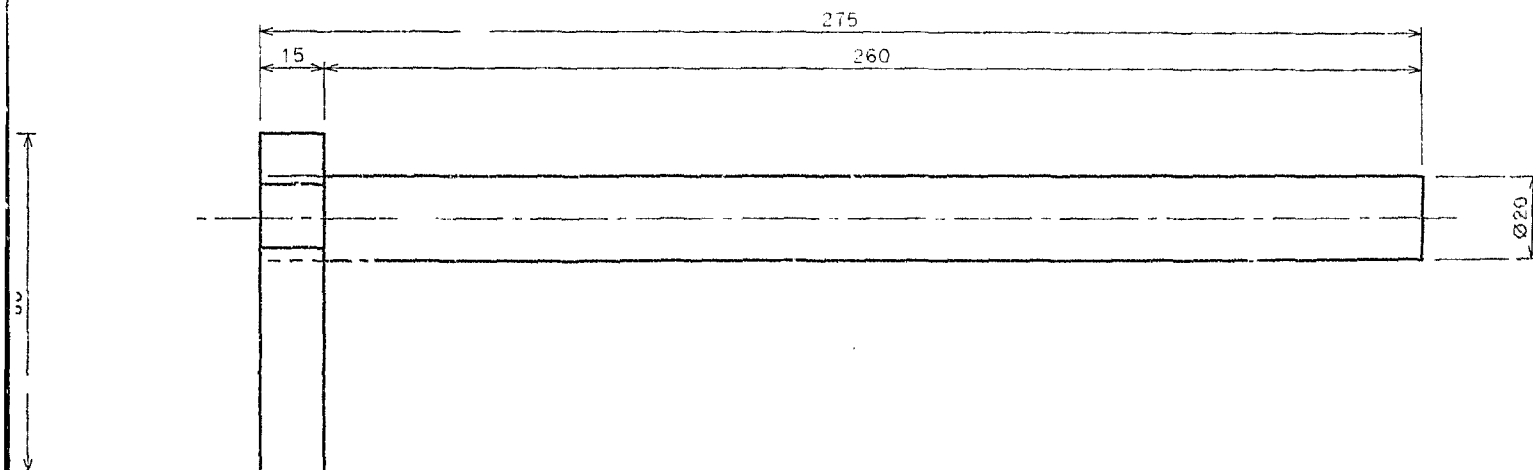


MATERIAL:- MILD STEEL

TITLE:	VERT
DR/ W/	
SCALE	

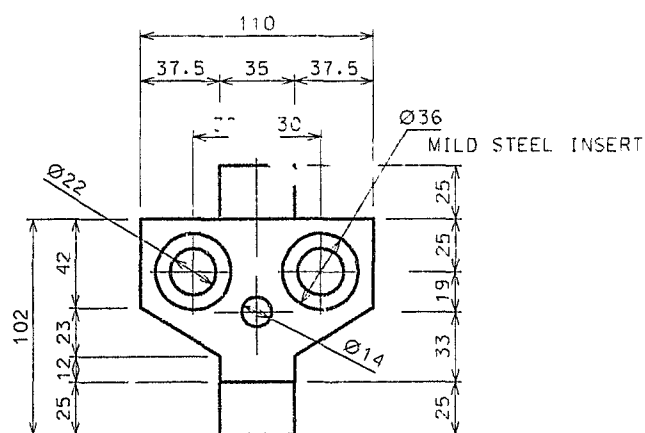


ELEVATION ON HORIZONTAL GUIDES
MATERIAL:- MILD STEEL

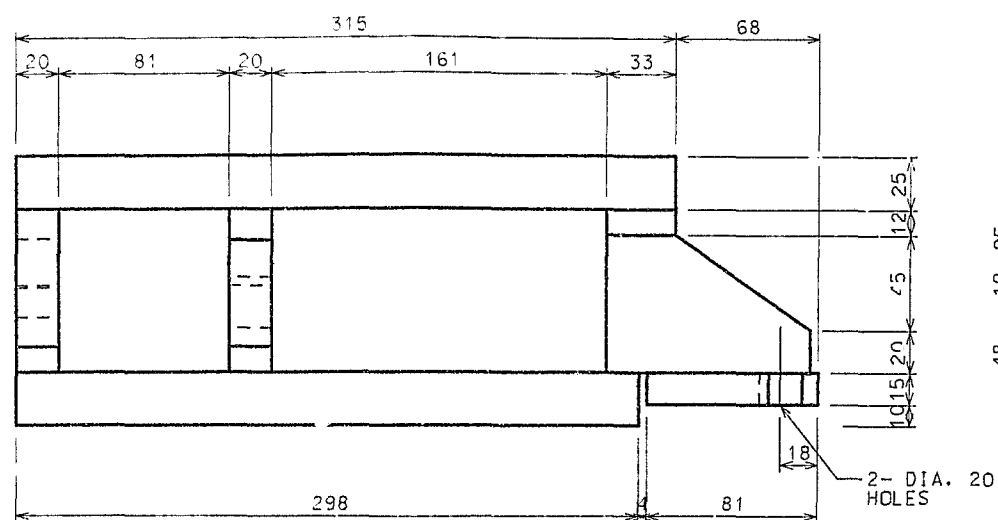


ELEVATION ON VERTICAL GUIDES
MATERIAL:- MILD STEEL

TITLE: ROBOTIC ARM VERTICAL & HORIZONTAL GUIDES			
DRAWN		DRG NO.	REV
SCALE	1:2		



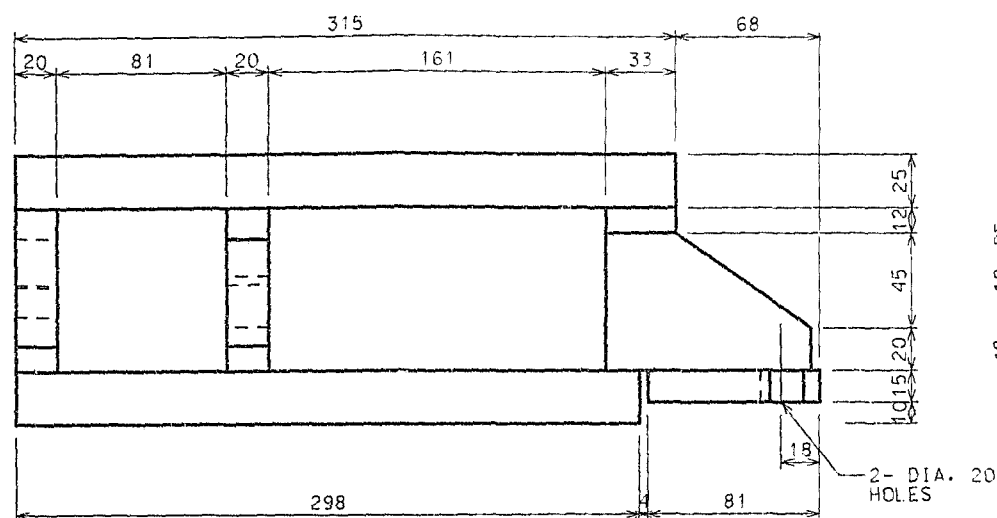
END VIEW



ELEVATION ON HORIZONTAL GUIDE ARM

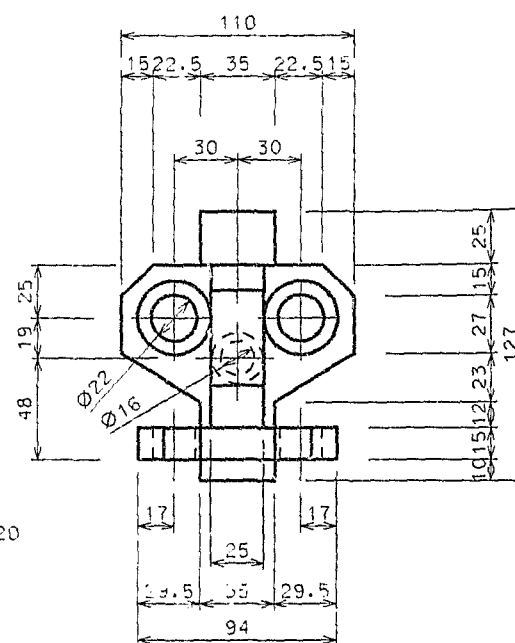
MATERIAL :- ALUMINIUM

EL INSERT



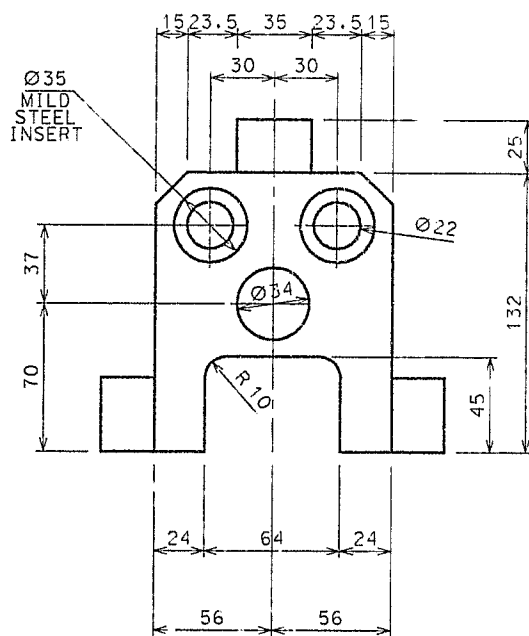
ELEVATION ON HORIZONTAL GUIDE ARM

MATERIAL :- ALUMINIUM

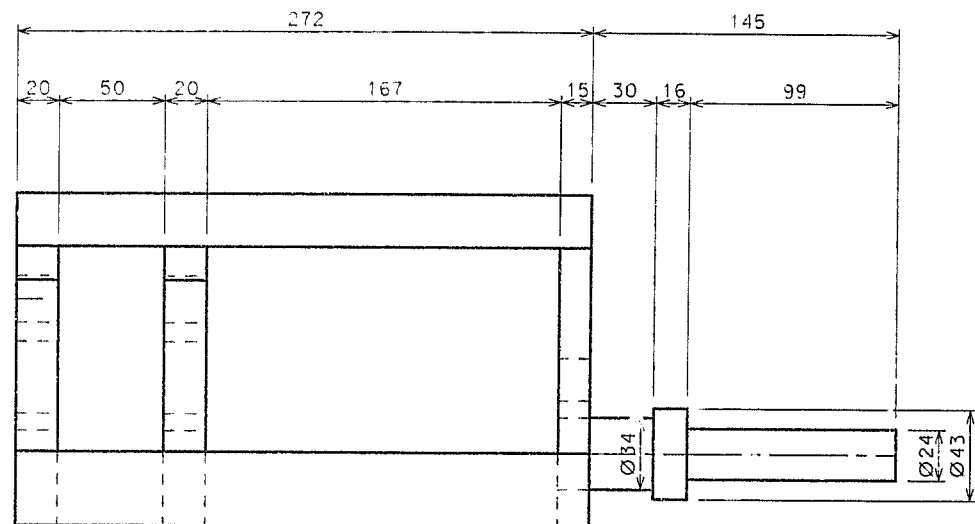


END VIEW

TITLE:			
ROBOTIC ARM HORIZONTAL GUIDE ARM			
DRAWN		DRG. NO.	REV
SCALE	1:2		

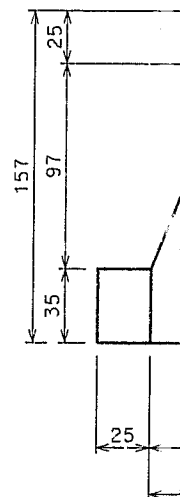


END VIEW

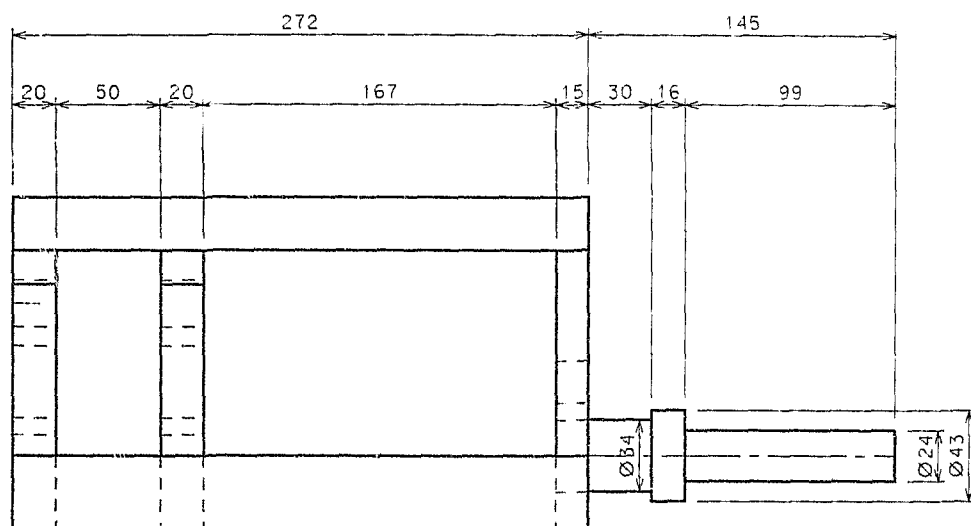


ELEVATION ON VERTICAL GUIDE ARM

MATERIAL :- ALUMINIUM

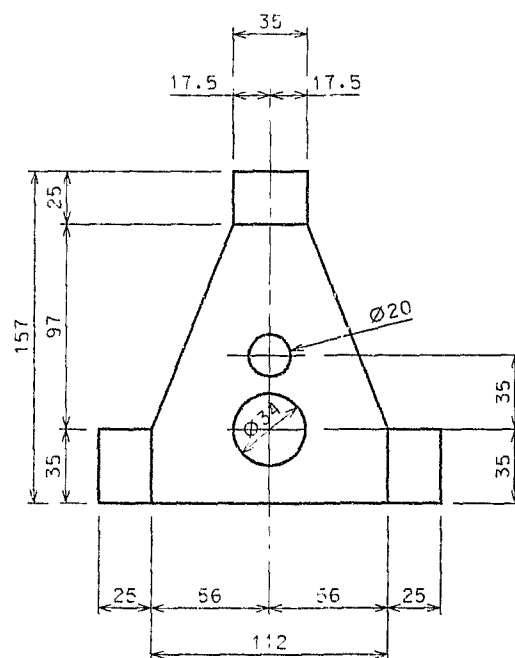


TITLE
DRAWN
SCALE



ELEVATION ON VERTICAL GUIDE ARM

MATERIAL :- ALUMINIUM



END VIEW

TITLE:			
ROBOTIC ARM VERTICAL GUIDE ARM			
DRAWN		DRG NO.	REV
SCALE	1:2		

APPENDIX C

QUOTATION FOR CNC MILL MODIFICATIONS



00

Reg. No. 24 of 2003/23

RETROFITTING OF MACHINE TOOL
SALES AND SERVICE OF CNC MACHINESTEL: (011) 845-3278/9
FAX: (011) 421-3373

20/10/1998

REF NO: DBM/10/765

ATTENTION: MR. GRAIG REIDRE: DYNAPATH DELTA 20M CONTROL

Dear Sir

Herewith our quotation to change the software on the Dynapath Delta 20M control to accommodate a robot Toolchanger as well as to supply the outputs and inputs required.

Software changes (P.L.C) 16HRS @ R220-00	R 3 520-00
Travel (1 x to Wits and Back) 1HR @ R220-00	R 2
90KM @ R1-70	R 15

SPARES

Plugs and cables	R 150-00
14% VAT	R 4 043-00
TOTAL	R 556-02
	<u>R 4 609-02</u>

C.N.C TO ROBOT 24V D.C

Pin1 = Binary Bit 1
Pin2 = Binary Bit 2
Pin3 = Binary Bit 4
Pin4 = Toolchange request

ROBOT TO C.N.C 24V D.C

Pin5 = Toolchange done
Pin6 = Tool unclamp
Pin7 = OV

UNIMATION

Reg No. OK 89-00935/22

As our work allocation schedule is currently full till our annual Shutdown in December, I cannot commit myself to a specific date, I will however try and make a plan to do the changes you requested before then but will advise you in this regard.

Thanking you



D.B. Maas
for Unimation cc

Author Read, David Craig.

Name of thesis Development of a tool changing robot for a computer numerically controlled milling machine. 1999

PUBLISHER:

University of the Witwatersrand, Johannesburg

©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.