RAPID PROTOTYPING USING A PRECISION ROBOTIC MANIPULATOR

Tang Sai Hong, B. Eng.

A thesis submitted to the Dublin City University in partial fulfilment for the degree of

Doctor of Philosophy

Sponsoring Establishment:

Universiti Putra Malaysia

September 2000

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: [illegible] ID No.: 97970018

Tang Sai Hong

Date: September 2000

ACKNOWLEDGEMENT

I am indebted to Professor M. S. J. Hashmi for his excellent supervision, remarkable guidance, helpful advice and truthful encouragement throughout the duration of this research.

I would also like to express my sincere gratitude to Dr. Shamsuddin Sulaiman, who continued to be one of my project mentors after my undergraduate studies.

My special thanks to Dr. Paul Young, who contributed his valuable time in my project.

The technical assistance provided by Mr. Keith Hickey, Mr. Liam Domican, Mr. Jim Barry and Mr. Martin Johnson are much appreciated.

Last but not least, my special thanks to my parents, brothers, friends and all postgraduates for their constructive suggestions, understanding and support.

DEDICATION

This research work in specially dedicated to my wife, who has been so supportive and understanding throughout the years of my studies. Thank you for being extremely patient with me.

ABSTRACT

A rapid prototyping system using a precision robotic manipulator has been developed. The system is comprised of a latest personal computer (Pentium II, 300 MHz, 128 MB RAM and 5 GB hard disk capacity), interfacing system (PS-23 indexer, KS-drives and servomotors), a four degrees of freedom precision manipulator and a ball nosed end milling equipment.

The hardware is integrated with the AutoSurf (CAD software), which is used in designing engineering models, section cut the surface models and changing graphic file into DXF files (neutral format files). The AutoLISP (AutoSurf programming language) has been used to simulate the additive prototyping process. The hardware is also linked with the self-developed CAM programs for data processing and motion control.

With the above hardware and software configuration, subtractive prototyping models have been produced successfully. Simple additive prototyping process was also simulated graphically in AutoSurf environment. The CAM programs were also tested to be fine with the additive prototyping models' data files.

Generally, the rapid prototyping system using the precision robotic manipulator has the advantage of being cheaper, effective, time and space saving, with dual purposes (subtractive and additive processes) and it is an all in one system.

CONTENTS

Tang, S. H.; Sulaiman, S. and Hashmi, M. S. J., Precision Robotic Manipulator based Rapid Prototyping System, *Proceedings of the IASTED International Conference of Robotics and Automation 2000*, 17 – 23, August 14 – 16, 2000.

LIST OF TABLES

LIST OF FIGURES

LIST OF PLATES

CHAPTER 1: INTRODUCTION

Robots are used in the manufacturing and non-manufacturing environments. They are either replacing human functions or working together with human operators. Robots have different skills and capabilities. Some of them are general-purpose robots that are capable of a wide range of tasks and others have specific applications.

Most of the robots are used in the manufacturing industries. Industrial tasks, which are categorised as dirty, dangerous, dull, hot, heavy, hazardous and demeaning but necessary, are best suited for robots. Besides the tasks mentioned above, robots are also used in other fields for improving productivity, increasing product quality and reducing labour costs.

Prototyping is one of the important phases in product development cycle. With a prototyped product, the designer can examine its aesthetic aspect, measure its physical dimensions, accomplish functionality test, validate its performance and so on. If the prototype is not up to the standard requirements, the designer will have to redesign the product and, prototyping will be needed as to check the product properties again.

As a result, the product design and validating cycle will continue as long as the prototype does not meet the desired product specifications. Hence, reducing the prototyping time is one of the crucial aspects in speeding up the product development cycle and increasing a company's competitiveness in the global market, especially when globalisation is inevitable.

Rapid prototyping is a popular term for describing any technology that can reduce the prototyping time drastically. With rapid prototyping technologies, designers and engineers can visualise a real-life prototype part in hours or days instead of weeks or months if traditional prototyping methods are used.

Rapid prototyping reduce the lead-time needed to complete the design and place a newly designed product on the production line faster. Because of the prototyping leadtime is greatly reduced, designers have the luxury of being allowed to have multiple iterations of a design and the engineers have numerous opportunities for validating the physical part so as to introduce a right-at-the-first-time product to the production line.

Currently, most people are focusing on new prototyping technologies like stereolithography apparatus (SLA), selective laser sintering (SLS) and others trying to minimise the prototyping time. But, improving existing or traditional prototyping methods can also reduce the prototyping time. In this study, the integration of a precision robotic manipulator with a traditional subtractive prototyping process (ball nosed end milling process) was developed in order to materialise the rapid prototyping concept.

It is not common to have an integrated system of rapid prototyping using a precision robotic manipulator. But in this information technology age, integration of various systems into a hybrid system is not impossible since in manufacturing industries' computer-aided design and manufacturing (CAD/CAM) technologies are not scarce.

The computer is particularly important in this integrated system because it is taking up the role as a brain. A robot has the ability to manipulate objects in the real world. This unique function when connected to a computer will enable the computer to interact with the real world. Literally, robots are the arms of the computer. So, with the availability of the CAD/CAM knowledge base, a hybrid system of a precision robotic manipulator and a subtractive prototyping tool is possible and viable.

Hence, a feasibility study of an integrated system of a personal computer controlled precision robotic manipulator coupled with CAD/CAM programs and ball nosed end milling process has been carried out. The results of the study will contribute to the knowledge base of the vast manufacturing industries.

1.1 AIM AND OBJECTIVES

The aim of the present work is to integrate a four degrees of freedom precision robotic manipulator, a high specification personal computer, computer-aided design (CAD) software, self-developed computer-aided manufacturing (CAM) programs and self-assembled subtractive prototyping equipment to create a rapid prototyping system. The objectives of the project are:

(1) To interface the precision robotic manipulator with the high specification personal computer replacing the existing PC/XT microcomputer.

(2) To develop postprocessors for Drawing Interchange File (DXF) format for subtractive prototyping process.

(3) To develop postprocessors for Drawing Interchange File (DXF) format for additive prototyping process.

(4) To develop programs for controlling the motion of the precision robotic manipulator in subtractive prototyping process.

(5) To develop programs for controlling the motion of the precision robotic manipulator in additive prototyping process.

(6) To construct a customised ball nosed end milling equipment for subtractive prototyping process.

(7) To produce three-dimensional complex shaped objects by subtractive prototyping process.

(8) To develop a program for simulating the additive prototyping process in a computer graphic environment.

1.2 PROJECT OVERVIEW

A precision robotic manipulator based rapid prototyping system was built and evaluated.

In the project, three-dimensional complex shaped models were designed using commercially available CAD software. Model definition data were then exported out from the software as neutral format data file. Self-developed CAM programs were used to extract important data out from the neutral format file and was later processed for creating the motion parameters for controlling the robotic manipulator in the final stage.

A ball nosed end milling equipment was also built to serve as the subtractive prototyping process of the system. The robotic manipulator end effector was modified so that it can clamp a cylindrical polystyrene block for subtractive and additive prototyping processes.

In the system, a personal computer was used to control a four degrees of freedom precision robotic manipulator. The motion control interfacing system between the personal computer and the manipulator is comprised of PC-23 indexer and KS drives with alternate current brushless servomotors.

With the motion parameter available, a self-developed motion control program can activate and control the motion of the robotic manipulator so as to manipulate the work material (polystyrene block) to be milled by the ball nosed end milling equipment. Three-dimensional complex shaped components were produced. Thus, a better performance traditional prototyping tool was upgraded to the level that is almost similar to the latest and expensive rapid prototyping tools.

Postprocessors for DXF file and control programs were developed for the additive prototyping process. The additive prototyping process was also simulated in a computer graphic environment. A commercially available CAD programming language was used to develop the simulation program.

CHAPTER 2: LITERATURE REVIEW

2.1 INTRODUCTION

The present work is to develop a rapid prototyping system by integrating a precision robotic manipulator with traditional subtractive prototyping process (ball nosed end milling) and computer-aided design and manufacturing (CAD/CAM) software. As a result, the following subsections will consider all the related elements of the integrated system. The subtopics are:

(1) The origin of the robot, robotics and its various definitions by different organisations.

(2) Different methods of classifying robots' physical configurations, power supplies, motion paths and so on.

(3) Industrial applications and specifications of robots.

(4) Computer-aided design (CAD) and its function in design process.

(5) Computer-aided manufacturing (CAM) and its role in planning and control.

(6) CAD/CAM, its scope in a product cycle development and database exchange.

(7) Prototyping methods - subtractive process (milling) and additive processes (STA, LOM, FDM, SLS and SGC)

2.2 ROBOTICS

The word Robot was added into the English vocabulary with the translation of Karel Capeks's play, R. U. R. (Rossum's Universal Robots) in 1923 [1]. Robot is derived from a Czech word, 'robota', means worker [2] or compulsory (forced) labour [3] and from 'robotnik', meaning 'serf' [4].

Isaac Asimov, the master science fiction writer [4], invented the word 'robotics'. He also propounded the famous Three Laws of Robotics [1].

(1) A robot must not harm a human being, nor through inaction allow one to come to harm.

(2) A robot must always obey human beings, unless that is in conflict with the first law.

(3) A robot must protect itself from harm, unless that is in conflict with the first or second laws.

The Three Laws are used to ensure that the designed robots are always 'keeping their place' [1] and they remain worthy design standards for roboticists to this day [4].

In the year of 1954, George Devol applied to patent a design for what is generally considered to be the first 'industrial' robot. Later in 1958, his robot was actually built [4].

Robot technology's advancement rate is closely related to the computer technologies. Advancements in computer technology such as the introduction of solid states' range of products greatly increase the computational capabilities and reduce the size of the control system [4].

Assembly is one of the areas where the applications of robotics grow rapidly nowadays. Robots assemble almost all personal computer boards, and they are moving into assembly applications in other industries. Besides, the industry is emphasising on flexible and small-lot manufacturing. As a result, computer-controlled robots are gaining popularity now [5]. Major countries that use robots are Japan, USA, Germany, Sweden, Italy, Britain and France [4].

2.2.1 Robot Definitions

Definition of the word ‘robot’ varies according to the geographical locations and communities. So far, robotics associations like the International Standards Organisation (ISO), Robotic Industries Association (RIA) and the British Robot Association are providing different definitions.

Since late 1970s, the Robotic Industries Association (RIA; formerly the Robot Institute of America) has defined a robot as “a manipulator, designed to move material, parts, tools or specialised devices through variable programmed motions for the performance of a variety of tasks [3].”

The International Standards Organisation (ISO) has a more lengthy definition of an industrial robot [6]:

“A machine formed by a mechanism including several degrees of freedom, often having the appearance of one or several arms ending in a wrist capable of holding a tool or a work piece or an inspection device. In particular, its control unit must use a memorising device and sometimes it can use sensing or adaptation appliances taking into account environment and circumstances. These multipurpose machines are generally designed to carry out a repetitive function and can be adapted to other functions."

British Robot Association's robot definition is almost same as the RIA's definition of a robot. But, the important of the reprogrammable capability of the robot has been emphasised. Its definition for an industrial robot is stated as 'An industrial robot is a reprogrammable device designed to both manipulate and transport parts, tools or specialised manufacturing implements through variable programmed motions for the performance of specific manufacturing tasks' [7].

Robots should be easily reprogrammed to carry out work on new tasks. By referring to the previous paragraph's definition of a robot, a numerical controlled (NC) machine tool is not a robot since, although it can be reprogrammed easily, but it is not designed to do anything other than cutting material. Neither is the type of arm used to handle toxic or radioactive material in the nuclear industry; human operator constantly controlling the devices remotely, i.e. they are not programmed to operate autonomously [7].

2.3 CLASSIFICATIONS OF ROBOTS

There are various ways of classifying robots, such as structural configurations, power sources, motion systems and so on. The following subsections will describe some of the common classification methods of robots.

2.3.1 Arm Geometry

The role of robot arm is to move the end effector (grippers or tool) to a given position in a desired orientation. In order to get to any point in space, an arm needs to have six degrees of freedom; namely three translational (right or left, forward or backward, up or down) for reaching the point and three rotational (roll, yaw, pitch) to get any orientation [4].

A common way of classifying the structural configuration of the arm is by looking at different co-ordinate systems of the three major axes (translational). The major axes will provide the vertical lift stroke, the in and out reaching stroke, and the rotational or traversing motion about the vertical lift axis of the robot [4]. Such a classification can distinguish between five basic types commonly available in commercial industrial robot [8] and they are described as below.

Cylindrical Co-ordinate Robot. The robot body is a vertical column that swivels about a vertical axis. The arm consists of a few orthogonal slides that allow it to be moved up or down and in or out with reference to the robot body [9]. This is illustrated schematically in Figure 2.3(a) [8]. In the figure, types of motions are shown by arrows; namely two translational motions and one rotational motion. This kind of robot allows good access into cavities and machining working area [10]. Typical manufacturers are Fanuc, Prab and Seiko [4].

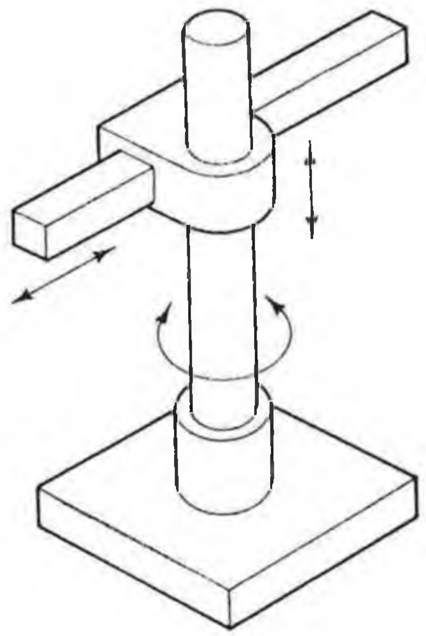


Figure 2.3(a) Cylindrical Co-ordinate Robot

Spherical (Polar) Co-ordinate Robot. As shown in Figure 2.3(b) [8], this type of robot geometry has two rotary axes combined with a linear axis. The base axis is a rotary axis with a second rotary axis providing vertical motion. The linear axis makes the radius of the sphere [11]. Unimation is one of the companies that produce this kind of robot [9]. As shown by the arrows in the figure, it has two rotational motions and one translantional motion. This kind of configuration can cover a large workspace from a central support. It can also pick up objects from the ground by bending down its upper part of the structure [10].

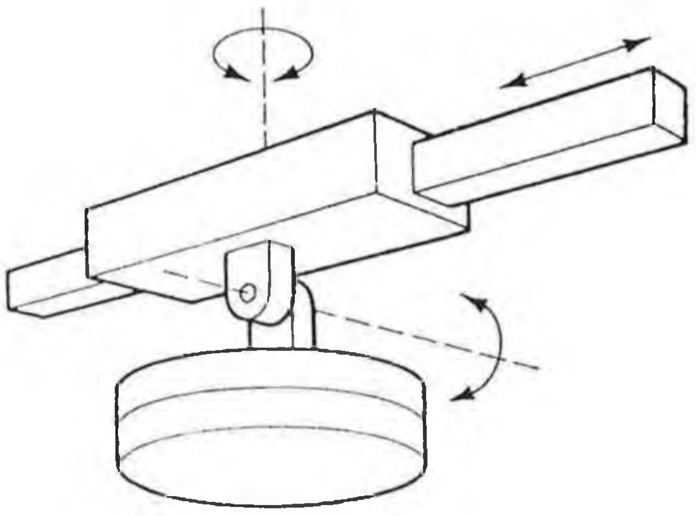


Figure 2.3(b) Spherical (Polar) Co-ordinate Robot

Cartesian (Rectangular) Co-ordinate Robot. Other names for this configuration include rectilinear robot, x-y-z robot [8] and gantry cranes [12]. This robot has joints that move in rectangular orthogonal direction [12] as illustrated in Figure 2.3(c) [8]. As shown by the arrows in the figure, it can have three translational motions. Examples of this kind of robot are IBM 7565 (originally RS1) assembly robot, the Olivetti Sigma and the DEA Pragma [4]. Generally, it has a rigid structure and can use inexpensive pneumatic drives for pick and place operations [10].

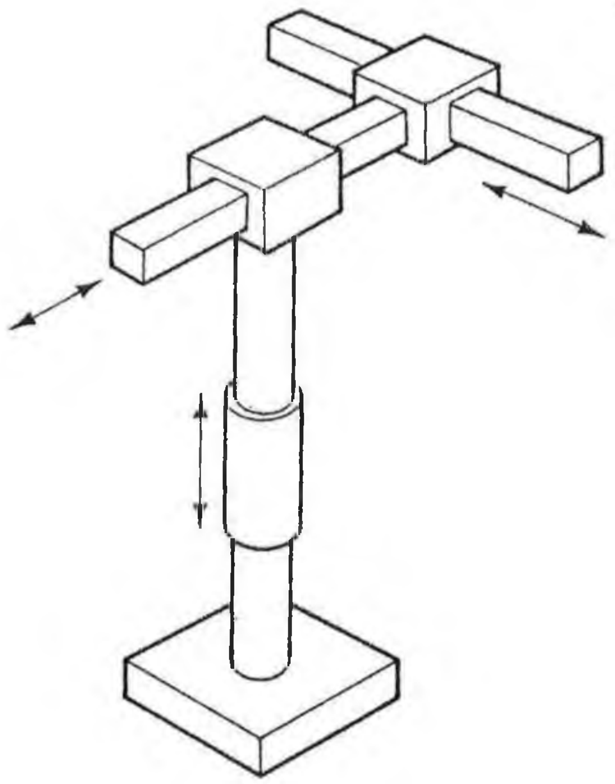


Figure 2.3(c) Cartesian (Rectangular) Co-ordinate Robot

Revolute Co-ordinate (Jointed Arm) Robot. Sometimes it is known as anthropomorphic robot. It is having three rotational motions, as shown by the arrows in Figure 2.3(d) [8]. It consists of rotary joints called the 'shoulder' and the 'elbow' (corresponding to the human arm) all mounted on a 'waist' consisting of a rotating base that provides the third degree of freedom [4]. It provides maximum flexibility. It can cover a large workspace relative to the volume of the robot and can also reach over and under an object. Electric motor is best suited for this kind of robot [10]. The Cincinnati Milacron, Asea and Unimation [9] are some of the typical manufacturers of this kind of robot.

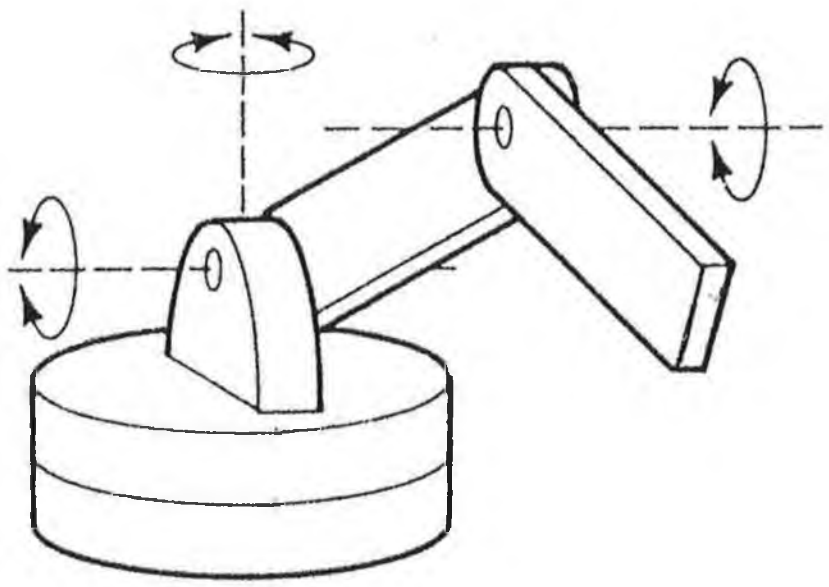


Figure 2.3(d) Revolute Co-ordinate (Jointed Arm) Robot

SCARA Robot. SCARA is an acronym for Selective Compliance Assembly Robot Arm. The robot is almost the same as the jointed arm robot except that the shoulder and elbow rotational axes are vertical. In another word, the arm can be constructed to be very rigid in the vertical direction, but compliant in the horizontal direction [8]. Compliance is a desirable property for arms that are used in assembly processes. The robot can flex its arm to accommodate the small errors in position that occur when one object is brought into contact with another [10]. Manufacturers of the robot are Pentel, NEC and IBM [4], to name a few. Figure 2.3(e) [8] is describing the robot schematically. The arrows in the figure show the two rotational motions and one translational motion.

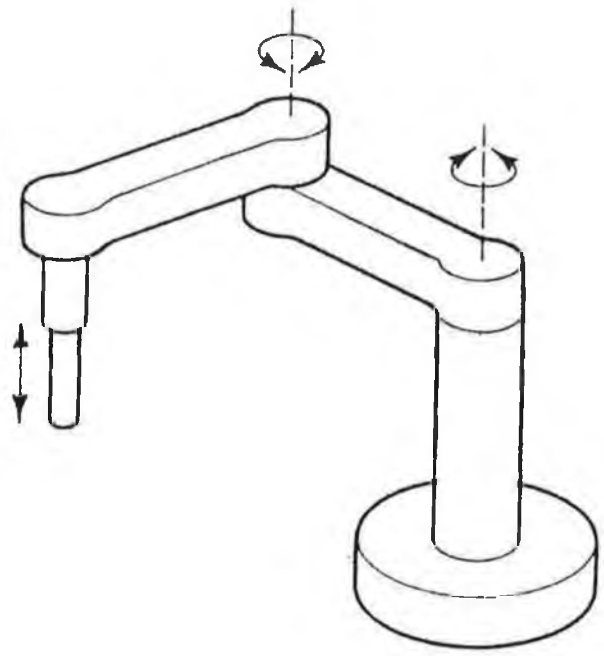


Figure 2.3(e) SCARA Robot

2.3.2 Fixed and Variable Sequence

The fixed sequence robot, also called a pick-and-place robot [2] (nicknamed bang-bang machine [4]), is the least sophisticated end of the robot scale [1]. It is programmed for a specific sequence of operations [2] only. The stroke of the motion axes is determined by adjusting the mechanical end stops. Limit switches are the typical sensors that sense the end points, and none of the points in between [4].

Its movements are from point to point, and the cycle is repeated continuously [2]. Generally, it is very difficult to reprogramme this robot or to execute a different sequence of operations. This is because its control system and memory, which are all embodied in a complex and interdependent set of limit switches, interlocks, end stops and electrical connections [1]. Many pneumatic driven robots are fixed sequence robots [8].

Variable sequence robots are usually associated with computer devices. In other words, their operation motions are controlled digitally. Variable sequence robots can immediately execute new and different tasks or a sequence of operations by running a new program [4]. Besides, the operator can create new programs offline and input them into the robot control system right after the old program ends.

Basically, a fixed sequence robot has a lower cost than the variable sequence robot because its control system is not as complicated as the variable sequence robot. The variable sequence robot is a general-purpose robot, whereas its counterpart is an operation-specific robot. One has to consider the requirement and complexity level of the functions of operations before choosing the appropriate type of robot for industrial usage.

2.3.3 Drive System

Robots can move because actuators activate their joints. Actuators are in turn powered by a particular form of drive system [1]. Three basic types of drive systems are used in commercially available robots, namely hydraulic, electrical and pneumatic [6].

In hydraulic drive system, devices like rotary vane actuators and linear pistons are used to accomplish the motion of the joint [8]. It uses fluid as the energy transfer medium. Hydraulically driven robots are usually large in size and take up more floor space than other kinds of robots. But, it has greater strength and higher speed if compared to the other two kinds of energy sources. Besides, maintenance personnel are more familiar with hydraulic systems. As a result, this system is considered to have the advantage of mechanical simplicity [9].

In the early 1980s, robots moved from hydraulic to electrical system and provided a major upheaval in technical breakthrough [13]. Nowadays, electric drive systems are becoming more prevalent in commercially available robots [8]. Stepping motors, servomotor and other kind of a. c. or d. c. electrical motors are used to drive the system [1]. It requires smaller floor space since it does not have a large hydraulic power unit [9]. Although it has moderate speed, its accuracy is the highest among all the drive systems [10]. Besides, its repeatability is better [9] and more readily adaptable to computer control [10] if compare to other systems. It also provides the cleanest and quietest actuation [6].

Like hydraulic drive systems, pneumatic drive systems also use fluid as the energy transfer medium. As a result, it uses devices like linear pistons and rotary vane actuators to move the joint [8]. But, pneumatic systems use air instead of liquid as in hydraulic system. Basically, it is only suitable for simpler robots [6] because of its natural problem like noise, leakage and compressibility property [10]. The advantages and disadvantages of the drive systems are summarised in Table 2.3(a).

Table 2.3(a) Advantages and Disadvantages of Various Drive Systems [10]

Pneumatic Drive System

Advantages

- Relatively inexpensive
- High speed
- Do not pollute work area with fluids
- Can be used in laboratory work
- No return line required
- Common energy source in industry
- Suits modular robot designs
- Actuator can stall without damage

Disadvantages

- Compressibility of air limits control and accuracy aspects
- Noise pollution from exhausts
- Leakage of air can be of concern
- Additional drying/filtering may be required
- Difficulties with control of speeds, take up of loads, and exhausting of lines

Hydraulic Drive System

Advantages

- Large lift capacity
- High power to weight ratio
- Moderate speeds
- Oil is incompressible, hence once positioned joints can be locked to a stiff structure
- Very good servo control can be achieved
- Self lubricating and self cooling
- Operate in stalled condition with no damage
- Fast response
- Intrinsically safe in flammable and explosive atmospheres
- Smooth operation at low speeds

Disadvantages

- Hydraulic systems are expensive
- Maintenance problems with seals causing leakage
- Not suitable for high speed cycling
- Need of a return line
- Hard to miniaturise because high pressures and flow rates
- Need for remote power source which uses floor space
- Cannot back drive links against valves

Table 2.3(a) Advantages and Disadvantages of Various Drive Systems (*Continue*)

Electric Drive System (DC motors and stepper motor)
Advantages
• Actuators are fast and accurate
• Possible to apply sophisticated control techniques to motion
• Relatively inexpensive
• Very fast development times for new models
• New rare earth motors have high torque, reduced weight, and fast response times
Disadvantages
• Inherently high speed with low torque, hence gear trains or other power transmission units are needed
• Gear backlash limits precision
• Electrical arcing may be a consideration in flammable atmospheres
• Problems of overheating in stalled condition
• Brakes are needed to lock them in position

2.3.4 Point-to-Point and Continuous Path

The motion systems of industrial robots can be classified into point-to-point (PTP) and continuous path (also called contouring) [12].

A point-to-point robot's movement is controlled from one point location in space to another. Before the robot executes its sequence of operations, the programmer key in the points of desired locations into its control memory. Motion parameters like velocity and acceleration are not controlled when the robot moves from one point to another [9].

Since the PTP robot axes may move at different velocities and accelerations, or even different linear or angular distances, each axis may reach its subsequent point location at different time frame. As a result, the path of motion of its end effector is unpredictable [12]. Some of the productive operations that are often performed by PTP robots are machine loading, spot welding and pick-and-place tasks [9]. PTP robots also need a fairly empty working volume because of its unpredictable path of motions [12].

Continuous path (CP) robots are capable of following a closely spaced locus of points, which describe a smooth compound curve. The memory and control requirement of a CP robot is greater than PTP robot since it needs to remember the complete path of motion instead of merely the end points of the motion sequence [9]. The CP robot is best suited for complex working environment, where it may need to interact continuously with its surroundings. Examples are paint spraying, continuous welding processes and picking up objects from a moving conveyor [12].

By looking at the type of tasks that can be accomplished by point-to-point robot and continuous path robot, one can understand that a CP robot will cost more than a PTP robot.

2.4 APPLICATIONS OF ROBOTS

Nowadays, almost all industrial robots are applied in manufacturing operations [8]. It is important to know the reasons behind the selection of robots in the manufacturing world. Some of the general characteristics of an industrial work environment which have tended to promote the replacement of human labour with a robot are described as below:

(1) *Hazardous or Uncomfortable Working Conditions.* If the working environment has the potential dangers or health hazards due to heat, radiation, or toxicity, or where the work area is uncomfortable and unpleasant to human being, then, a robot should

be considered doing the task instead of human [9]. Typical job situations that have the mentioned characteristic are hot forging, spray painting, continuous arc welding and spot welding [8].

(2) *Repetitive Work Cycle.* When the sequence of operations are relatively simple and repeated in every work cycle, a robot will perform better than its human counterpart. It is because robot has greater consistency and repeatability than human worker [8]. This is especially true if the job is accomplished within a limited work area. Pick-and-place operations and machine loading are two kinds of repetitive tasks.

(3) *Difficult Handling.* If the involved work part or tool in the operation is awkward or heavy, then, a robot might be a better choice in accomplishing the job because some industrial robots are capable of lifting up payloads that are several hundred kilograms. A human worker would increase the production cycle time since he or she will need the assistance of some mechanical devices to accomplish the task [9].

(4) *Multishift or Continuous Operation.* The labour savings will result in a quicker payback if the initial investment cost of the robot can be spread over two or three shifts [9]. Besides, continuous manufacturing process is better because equipment set up time can be reduced or eliminated. In some cases, machine set up time is crucial in determining the manufacturing cost of the product.

2.4.1 Robot Specifications

From a company's point of view, robots can improve the process quality by providing higher accuracy and repeatability. These can also perform the process under conditions that lead to higher quality but is not conducive for human operation. Besides, costs can be reduced because of saving labour cost, elimination of some processes and discarding some safety or extra equipment. Performance can also be improved because robots can perform tedious, repetitive tasks at greater speeds and with continuous operation [6].

But, how can it be determined whether a robot can perform the desired task or not? Or is there any guideline and reference for robot specifications? Bearing in mind the two questions above, a robot-human chart was developed. The chart is partially shown in Table 2.4(a). Characteristics like strength and power, consistency, overload – underload performance and environmental constraints are compared between human and robot manipulation.

Table 2.4(a) Robot-Human Charts: Comparison of Robot and Human Characteristics [14]

Characteristics	Robot	Human
Manipulation		
Strength and power	a. 0.1 – 1000 kg of useful load during operation at normal speed: reduced at above normal speeds b. Power relative to useful load	a. Maximum arm load: <30 kg; Varies drastically with type of movement, direction of load, etc. b. Power: 2 hp ≈ 10 s 0.5 hp ≈ 120 s 0.2 hp ≈ continuous 5 kc/min Subject to fatigue: may differ between static and dynamic conditions
Consistency	Absolute consistency if no malfunctions	a. Low b. May improve with practice and redundant knowledge of results c. Subject to fatigue: physiological and psychological d. May require external monitoring of performance
Overload-underload performance	a. Constant performance up to a designed limit, and then a drastic failure b. No underload effects on performance	a. Performance declines smoothly under a failure b. Boredom under local effects is significant
Environmental constraints	a. Ambient temperature from -10 to 60°C b. Relative humidity up to 90% c. Can be fitted to hostile environments	a. Ambient temperature range 15 to 30°C b. Humidity effects are weak c. Sensitive to various noxious stimuli and toxins, altitude, and air flow

2.5 COMPUTER-AIDED DESIGN (CAD)

A definition for Computer-Aided Design (CAD) is, "the effective use of computer in creating, modifying, or documenting engineering design in any design activity" [8]. It is also a technique where man and machine are blended into a problem solving team, intimately coupling the best characteristics of each [15].

A computer-aided design system is comprised of [16]:

(1) the computer and associated peripheral equipment which are called hardware;

(2) the computer programs which are called software that run on the hardware;

(3) the data structure which is created and manipulated by the software;

(4) human knowledge and activities.

CAD systems often have large and complex computer programs, perhaps using specialised computing hardware. Normally, the software is comprised of the following elements that process the data stored in the database in different ways [16].

(1) *Model Definition.* Adding geometric elements to a model of the form of a component is a typical example.

(2) *Model Manipulation.* Moving, copying, deleting, editing or modifying the design model's elements.

(3) *Picture Generation.* Generation of design model images on a computer screen or on some hardcopy devices.

(4) *User Interaction.* Handling user's input commands and presenting output to the user about the operation of the system.

(5) *Database Management.* Management of the files that make up the database.

(6) *Applications.* Generating information of revaluation, analysis or manufacture.

(7) *Utilities.* It is a 'catch-all' term for parts of the software that do not directly affect the design model, but modify the operation of the system in some ways such as selection of line type, display colour, units and so on.

There are four important reasons for using a computer-aided design system to support the engineering design operation [9]:

(1) *Increasing Designer's Productivity.* Designer can reduce time in operations like synthesising, analysing, and documenting the design because CAD can help him or her in conceptualising the product and its components.

(2) *Improving Design Quality.* Utilising CAD system with suitable hardware and software capabilities allows the designer to produce a more complete engineering analysis and to consider a larger quantity and variety of design alternatives.

(3) *Improving Design Documentation.* The graphical output of a CAD system is better than manual drafting in terms of quality of documentation. The engineering drawings are superior, and there is more standardisation among the drawings, fewer drafting errors, and greater legibility.

(4) *Creating Manufacturing Database.* Database for manufacturing a product is created in the process of product design documentation. Important manufacturing data like geometric specification of the product, dimensions of the components, materials specifications, bill of materials and others are documented in an engineering design.

Roughly 80 percent of a product's ultimate cost is determined and fixed during the design phase. Typically, companies only allocate about 5 percent of their resources on design and engineering. Contrasting these patterns of expenditure and commitment of resources over a product's life cycle, one can see that the 5 percent can have a great effect on competitive advantage [17]. A CAD system can have a significant effect on that 5 percent by increasing the effectiveness of the design and engineering tasks. CAD is not really used to reduce the percentage; it is used to come up with a better decision about the 80 percent of the product's cost and performance characteristics [18].

Some of the common and commercial available computer-aided design software programmes are AutoCAD, Microstation 32, I/EMS, CATIA and MENTOR [6]. The leaders in the mechanical CAD software arena are IBM, Computervision, Hewlett-Packard, Schlumberger, Autodesk, SDRC, EDS/Unigraphics and Parametric Technology [19].

CAD programming languages are essential part of modern engineering activities. During the last decade, AutoLISP programming language has been used in a few areas such as a surface-climbing robot simulation [20], water pollution simulation over a river basin [21], geometry simulation in sand mould casting [22], shape similarity assessment of mechanical parts [23] and fragmentation assessment in blasting [24]. However, AutoLISP has never been used in simulating a rapid prototyping process before because not many industrial companies are using AutoLISP in their manufacturing operations. AutoLISP is more common in the education institutions and home users that rely on low cost personal computers. Its use in helping to simulate an additive prototyping process would be a novel research project.

2.6 COMPUTER-AIDED MANUFACTURING (CAM)

Computer-aided manufacturing (CAM) is defined as the effective use of computer technology in the planning, management, and control of the manufacturing functions [8]. Computer-aided manufacturing is the business end of computer graphics, where all the physical work gets done on the factory floor to actually produce the parts designed with computer-aided design system. In 1952, the first numerically controlled machine tool was successfully demonstrated at the Massachusetts Institute of Technology (MIT). Since then, CAM has played an important role in the manufacturing industry [25].

The application of CAM can be grouped into two categories. The two categories represent two different levels of involvement of the computer in plant operations and they are [8]:

(1) Manufacturing *planning*. A computer is used indirectly to support the production function, and there is no direct connection between the process and the computer. Computer is used "off-line" to provide essential information for the effective planning and management of production activities. Applications of CAM in planning are cost estimating, computer-aided process planning, computer-assisted NC part programming, development of work standards, computer-aided line balancing and, production and inventory planning.

(2) *Manufacturing control.* The control function in manufacturing includes individual processing and assembly operations regulation, and plant-level activities management. Process level control involves the achievement of certain performance objectives by proper manipulation of the inputs to the process. Plant level control includes effective use of labour, equipment maintenance, transferring material in the factory, shipping good quality products on schedule, and maintaining the lowest plant operating costs. CAM applications in the manufacturing control aspect are

concerned with developing computer systems for managing and controlling the plant's physical operations like quality control, shop floor control, and so on.

2.7 COMPUTER-AIDED DESIGN AND MANUFACTURING (CAD/CAM)

CAD/CAM technology brings together two major elements of computerised manufacture. CAD involves creating a file resemblance of a finished product and generating the data needed for its manufacture and CAM will use the data to build programs that run the machine tools that will make the designed product [26].

The origin of computer-aided design and manufacturing (CAD/CAM) can be traced back to the beginning of civilisation when engineers of ancient Egypt, Greece and Rome acknowledged graphics communication. A few existing drawings on Egyptian tombs can be considered as technical drawings. Even today's graphic conventions like isometric views and crosshatching are found in Leonardo da Vinci's available work and notes.

A French mathematician, Gaspard Monge (1746 – 1818) invented orthographic projection, who was then working as a government designer. It was the inventions of computer and xerography that have given graphics, and consequently CAD/CAM technology, their current dimensions and power [27]. CAD/CAM technology continue to evolve with the rapid growing of computer capabilities since 1940s, when the first appearance of the huge, electromechanical machines that used clicking relays to perform computations that we called digital computer [28].

CAD/CAM involves the application of digital computers in completing certain functions in design and production [8]. The combination of CAD and CAM in the CAD/CAM term symbolised the efforts to integrate the design and manufacturing functions in an organisation into a continuum of activities, rather than treating them as two distinct and separate activities, as they have been considered in the past [8].

In fact, the application of microcomputers in design and manufacturing constitutes the most significant opportunity for substantial gains in industry; namely higher productivity, better quality, and lower cost [29]. Advances in the hardware and software of microcomputers have also made it easier to link CAD/CAM system to other manufacturing processes, so that product design and manufacturing can be integrated into a single system [30].

One of the aims of a CAD/CAM system for robotics is off-line programming [31]. Modern methods in robot programming utilise the technology of off-line programming, where computers are used for programming without the need to connect physically to the robot or even to be anywhere near its physical presence [32]. Off-line programming is suited to full computer integration of a facility [7].

2.7.1 CAD/CAM in a Product Cycle

Product cycle is the activities and functions that must be accomplished in the design and manufacture of a product. So, what is the scope of CAD/CAM in the operations of a manufacturing firm? Figure 2.7(a) shows the various steps in the product cycle [9]. Customers and the markets who drive the product cycle. The customers might activate the cycle if they design the part and later pass it to a different company for manufacturing. Or, the design and manufacturing of a product might be accomplished by the same organisation.

Whatever the case, the cycle will begin with a concept, an idea for a product. The concept is then cultivated, refined, analysed, improved, and translated into a plan for the product through the design engineering process. Later, a set of engineering drawings are drafted for documenting the plan to show how a product is made and providing a set of specifications indicating how the product should perform. Except for engineering changes that normally follow the product throughout its life cycle, this completes the design activities in Figure 2.7(a).

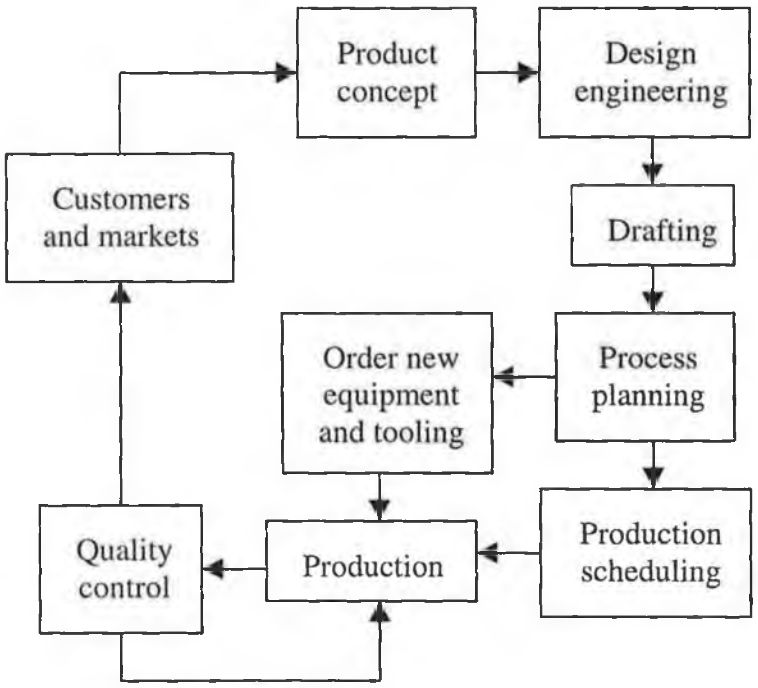


Figure 2.7(a) Product Cycle (Design and Manufacturing)

The next phase is the manufacture of the product. A process plan that specifies the required sequence of production operations for making the product is formulated. Sometimes, new machine and tools must be acquired for producing a new product. Scheduling activity provides a plan that specifies the correct quantities and dates of delivery of the final product. Once the plans are ready, the raw materials goes into the production line, followed by quality testing, and delivery to the customer.

Figure 2.7(b) shows the revised product cycle after the introduction of CAD/CAM [9]. Computer-aided design and automated drafting are used in the conceptualisation, design, and documentation of the product. Planning and scheduling processes are performed more efficiently with the aid of computers. Computers are also used in monitoring and controlling manufacturing operations. In quality control, the product and its components are inspected and tested by computers too.

One example where CAD/CAM system has brought significant benefit to the manufacturing industries happened at Daimler Benz in Germany. This company applied a CAD/CAM system to its tool making operations for six years. The benefits are 2 percent cast metal savings in pressure die casting, 3 percent in the cast iron foundry and 6 percent in gravity die casting. Moreover, the firm claimed reductions in tooling costs and reduction of up to 50 percent in delivery times. With a CAD/CAM system, the die shapes are more consistent. The elimination of the substantial variations in volume between apparently identical tools could cost as much as 3.5 percent in cast metal overweight [33].

Another example is the application of CAD/CAM in sheet metal production for making the Citation III aircraft (product of Cessna). Manual methods for part creation is about forty hours. If a minicomputer is used, the time of production is about sixteen hours. When high-speed modem linked with a mainframe computer, it needs eight hours to produce the same part. With CAD/CAM, the production time is approximately one hour [34].

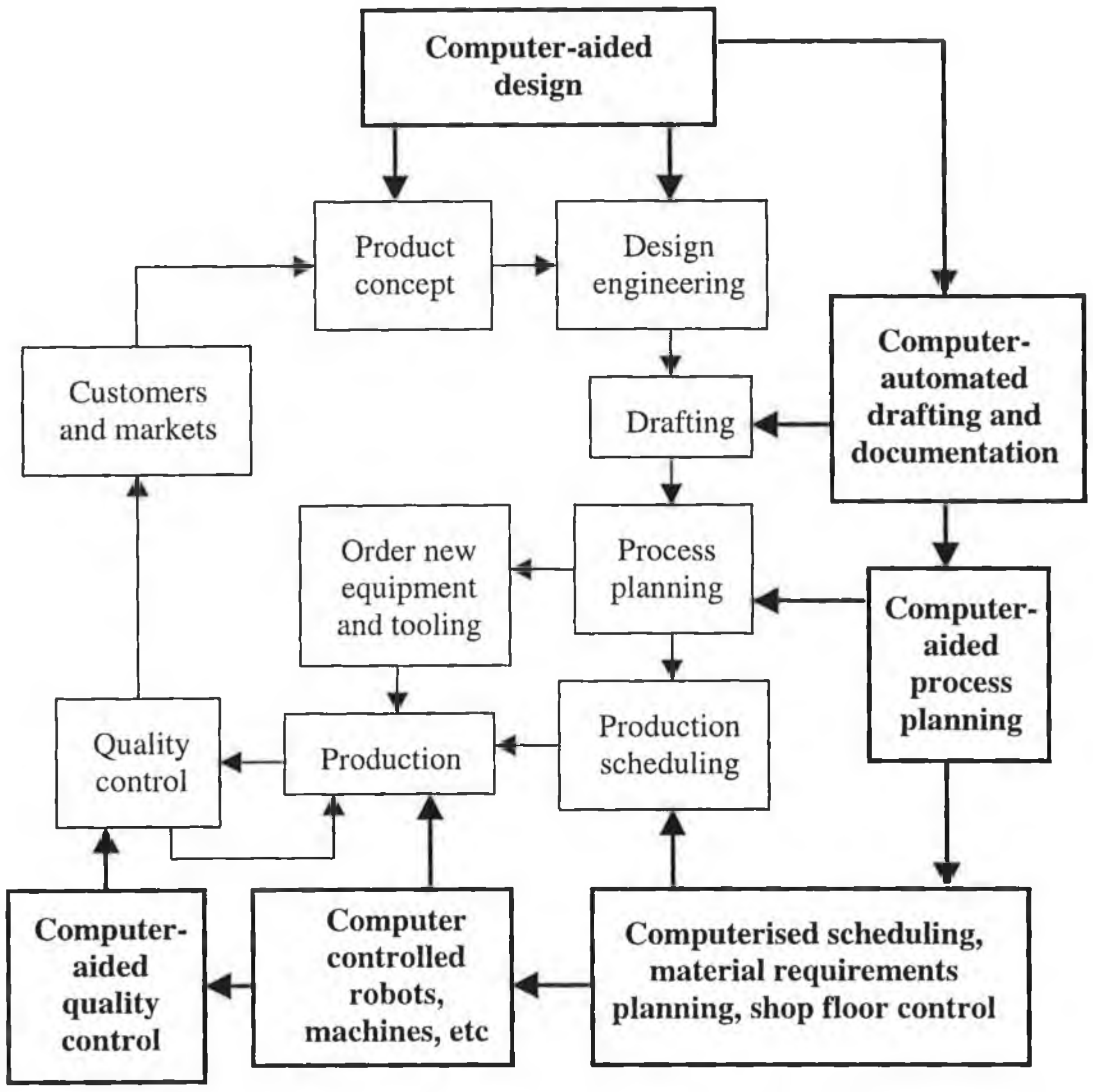

Computer-aided design
Product concept
Design engineering
Customers and markets
Computer-automated drafting and documentation
Drafting
Order new equipment and tooling
Process planning
Computer-aided process planning
Quality control
Production
Production scheduling
Computer-aided quality control
Computer controlled robots, machines, etc
Computerised scheduling, material requirements planning, shop floor control

Figure 2.7(b) Revised Product Cycle with CAD/CAM

2.7.2 CAD/CAM Data Exchange

One of the most important aims for CAD/CAM integration is the supply of interfaces to provide a data structure by which all product definition data can be transferred between various systems. All entities of the various applications should be converted without loss of information via the interface [35].

Computer databases are used to define product geometry and non-geometry for all phases of product design and manufacturing. The complete product description consists of the four types of modelling data as below [27]:

(1) *Shape Data.* Geometrical data such as font, colour, layer and other, topological data applied to solid modelling as well as part or form features which allow high-level concept communication about parts such as hole, flange, web, pocket, chamfer and so on.

(2) *Non-Shape Data.* Graphics data such as shaded images, and model global data as measuring units of the database and the resolution of storing the database numerical values.

(3) *Design Data.* Information generated from geometric models for analysis purposes such as mass property, finite element mesh data and others.

(4) *Manufacturing Data.* Production information likes tooling, NC tool paths, tolerancing, process planning, tool design, bill of materials and so on.

Data need to be transferred from system to system like CAD-to-CAD, CAD-to-CAM, CAM-to-CAM and CAD/CAM-to-CAD/CAM, when an organisation has many kind of software in the manufacturing work place. The receiving program cannot interpret streams of binary digits constituting the data unless its format or coding scheme is known.

Normally, databases exchange is not an issue if only one software house produces all the programs because the programmers are constrained by their management to agree a standard. When the programs are from different suppliers, problems arise because there is no common management and standards among the software companies [36].

In many cases, program routines pass data to other programs when they call each other, that is, giving each other work to do. A programmer writing a routine that calls other programmer's routine must know precisely what operation the subroutine will perform and in what form it will present the results.

The programmer also needs to know the kinds of arguments that the subroutine can accept and what it will do if it is unable to perform the operation for any reason. Thus, the communication between different routines is complex and requires careful documentation of the interface between them. It is even more important when a package of subroutines is supplied as a product by one company and then used by other companies [36].

There is often a requirement to be able to combine two or more design and manufacturing systems into an application that shares common data. This requirement may exist either internally among different departments of an organisation, or externally as in the case of subcontract manufacturers or component suppliers. As a result, the need to exchange databases is directly motivated by the need to integrate and automate the design and manufacturing processes to obtain the maximum benefits from CAD/CAM systems.

A good example describing the importance of having a complete data exchange process is by looking at the development in JAMA (Japan Automobile Manufacturers Association). JAMA has set up a task force for standardising the data exchange procedures. It is because more and more Japanese automobile companies conduct their business transactions beyond the framework of affiliation such as sharing part specifications to reduce developmental costs. This tendency was spurred by the increasing of local procurement of parts and moulds from overseas factories as the foreign products are cheaper and have equal quality [37].

However, database exchange among CAD/CAM systems is complicated because of the incompatibilities among entity representations, the complexity of commercial CAD/CAM systems, varying usage requirements of organisations, restrictions on access to proprietary database information and rapid pace of technological change [27]. It takes an expert in mathematics and CAD/CAM systems to understand the translation problem and develop a solution [38].

In view of the need and challenges described above, two solutions were found; namely direct and indirect database exchange.

Direct data exchange usually translates the modelling data stored in a product database directly from one CAD/CAM system's internal format to another system's internal format in one step. On the other hand, indirect data exchange is more general and adopts the concept of creating a neutral database structure (also called neutral file) which is independent of any existing or future CAD/CAM systems [27].

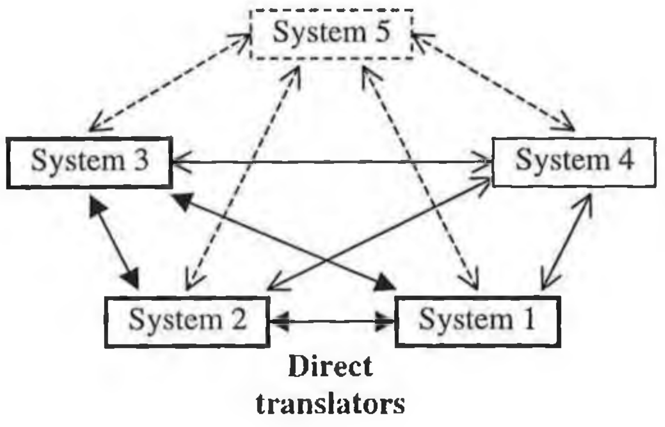
The structure of the neutral file is only governed by the minimum required definitions of any of the modelling data types, and is independent of any vendor format. Basically, the format of the neutral file is influenced by the structures of existing vendors' databases and can be considered as the common denominator among them [27].

Figure 2.7(c) [39, 40] shows the different methods of data exchange between direct and indirect translators. Direct translators convert data format directly in one step and are typically written by computer services companies that specialise in CAD/CAM database conversion. They can be considered as dedicated translator programs, that link a system pair as shown by the dual direction arrows shown in the figure. Two translators are needed to transfer data between two systems. For example, one translator transfers data from system 1 to system 2 and the other from system 2 to system 1.

On the other hand, each system that uses the indirect translation method has its own pair of processors to transfer data to and from the neutral file. Pre-processor will transfer a given system's data format to neutral format and postprocessor will do the opposite transfer [27].

Direct translators will give satisfactory solutions when the number of involved systems is small. When the number of involved systems increases, the number of translators that need to be written becomes prohibitive. In fact, indirect solution does not need as many translator programs as direct solution when the number of systems increase.

Let's look at Figure 2.7(c) again. When the number of systems is three (System 1, 2 and 3), both methods need six translators (dual directions darker arrowheads). When the systems quantity reaches four (System 4 is added), direct method lose out because it needs six more translators if compare to the indirect method. When there are five systems (System 5 is added), indirect solution is a lot better since it only needs to build ten translators instead of making twenty translators as needed by direct method [27].



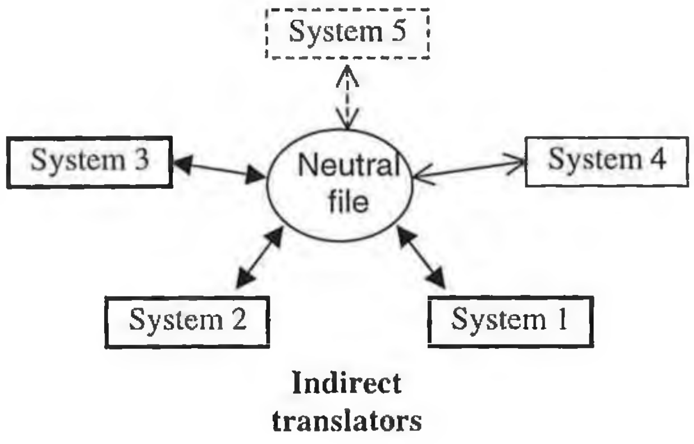


Figure 2.7(c) Methods of Exchanging Databases

Besides, if one of the systems in Figure 2.7(c) changes, eight of the direct translators programs need to be updated and re-tested. Currently, there are more than fifty different CAD/CAM systems on the market. If five of them revise their systems every year, then, it would require a lot of programming effort just to maintain a status quo [39].

Apart from the advantages mentioned above, indirect translator philosophy also provides stable communication between CAD/CAM systems, protects against system obsolescence, and eliminates dependence on a single system supplier (against monopoly). Neutral file has the potential being archived especially by companies in aerospace industry that need to keep CAD/CAM databases for twenty to fifty years [27].

There are quite a number of neutral formats available in the CAD/CAM industry. They can be categorised into three areas as listed below and their respective standards are shown in Table 2.7(a) [36].

(1) The CAD drawing or product definition itself.

(2) Graphics packages of subroutines that provide standard display and user interaction facilities to the actual CAD/CAM program independent of the actual hardware used.

(3) Device control languages providing standard commands to control the display, printer or plotter.

IGES (Initial Graphics Exchange Specifications) was the first widely available neutral format. It was used as the neutral file for exchanging mechanical engineering data (i.e. that traditionally presented as engineering drawings) [41]. In U.S., IGES is recognised as the industry standard and monitored by the National Institute of Standards and Technology (NIST). IGES is the most widely accepted neutral file format for transferring complex surfaces like NURBS or trimmed surfaces. In Europe, the VDA2 standard (from Germany) is popular [42]. However, the most common used data exchange standards are IGES and Autodesk Inc.'s DXF [43].

Table 2.7(a) Data Exchange Standards By Applications

Product Definition Standards	Graphic Package Standards	Device Control Languages
1) IGES (Initial Graphics Exchange Specification) 2) SET (Standard d'Echange et de Transfert) 3) VDA-FS (DIN 66301) 4) VDA-IS 5) STEP (Standard for the Exchange of Product Model Data) 6) PDES (Product Data Exchange Specification) 7) EDIF (Electronic Design Interchange Format) 8) DMC (Digital Mapping for Customers) 9) DXF (Drawing Interchange File) 10) ISIF (Intergraph Standard Interchange Format) 11) CGM (Computer Graphics Metafile)	1) GKS ISO 7942 (Graphics Kernel System) 2) The CORE System 3) PHIGS (Programmer's Interactive Graphics System)	1) The X Window System 2) Calcomp plot calls 3) Tektronix graphics commands 4) HPGL (Hewlett Packard Graphics Language) 5) PostScript 6) CGI (Computer Graphics Interface)

Since personal computers are becoming more and more popular and powerful nowadays, DXF format will surely play a more important role in the years to come because most of the Autodesk software is operated on personal computers. This is especially true in developing countries where personal computers are more widely used than workstations.

The feasibility of using DXF as a way to exchange CAD data among heterogeneous CAD systems in the Pohang and Kwang-yang (both locations are in South Korea) steelworks has been investigated [44]. It shows that DXF is a major player in the data exchange format world.

2.8 PROTOTYPING

Prototyping has been one of the essential processes in the design and manufacturing cycle. Most of the time, a conceptual design has to be developed into a physical product so that the designers and engineers can rate its aesthetic features, validating its functionality, checking for specifications conformance, testing for performance, and so on.

There are two important challenges for product manufacturing industry at present. One of it is to find a better way of reducing the product development time substantially. Another challenge is the improvement in flexibility for manufacturing small batch size products and a variety of types of product [45]. If the product development time is shorter, then, the lead-time to market will be reduced. Subsequently, a firm can grab a bigger market share.

In fact, more than 70 percent of senior management staff rate the lead-time to market as one of the three most important criteria that drive them in the businesses. Thus, the key to success for most manufacturers is the capability to provide quality products to market, at the shortest possible lead-time with the right cost [46]. Besides, the gradual shift from mass production to customised production for satisfying a growing number of 'niche markets' has also push for the reduced time to market [47].

One way of having a shorter time to market for a product is to reduce the prototyping time. Other benefits of having a shorter prototyping time are [45]:

(1) *Visualisation.* Conceptual models are very important in the product design phase. CAD is used to generate computer representations of design concepts. No matter how good the engineers interpret the blue prints and how well the CAD images of

complex shape objects are, it is still very difficult to visualise exactly what the actual complex shape products will look like. Some errors may escape from the review of designers and engineers. The touch of the real life objects can reveal unexpected problems and sometimes lead to a better design modification. Shorter prototyping time enable the design and manufacturing staff evaluates the design very quickly.

(2) *Verification and Optimisation.* Improving the quality of product is always an important task in the manufacturing. With shorter prototyping time, the design concept verification and optimisation tasks can be accomplished faster. As a result, the product quality can be improved within the limited time frame and with affordable cost.

(3) *Iteration.* With shorter prototyping time, it is possible to go through numerous design iterations within a short time and substantially reduce the overall model development time.

(4) *Planning and Tooling.* By having the physical product at an earlier design stage, process planning and tooling design can be sped up. The physical prototype can also be used to reduce problems in interpreting the blue prints on the shop floor by describing the complex geometry accurately.

(5) *Marketing.* Prototypes can be used to demonstrate the concept, design ideas and the company's capability in producing it. The reality of the physical model shows that the design concept is feasible and acceptable. With a faster prototype manufacturing time, customers can give their valuable feedback faster for design modifications so that the final product will meet the customers' requirements. Total customer satisfaction is one of the key factors in penetrating the market.

There are two ways to reduce the prototyping time. Both ways will need computers to control the processing operations and producing complex shape components. The computers are able to represent objects in a three-dimensional co-ordinate system [48].

One way is to develop new prototyping technologies like stereolithography apparatus, selective laser sintering, fused deposition modelling, and so on [49]. These methods are categorised under additive prototyping processes. Another method is to improve the principal existing technique like integrating a precision robotic manipulator into a conventional machining process like milling, with the aid of CAD/CAM. The improved system will save as much as 40 percent of floor space with the same size of workspace [50]. The second method is considered as subtractive prototyping process.

2.8.1 Subtractive Prototyping

Numerical control (NC) systems were commonly available in the 1960s. NC systems are used in controlling conventional subtractive processes. NC processing equipment can reduce the prototyping time of producing complex shape objects compared with the conventional methods that normally takes weeks and was primarily a job for highly skilled machinists [51]. NC machine tools were the earliest types of digital controlled subtractive rapid prototyping process [52]. Until now, subtractive prototyping is still one of the favourite methods at producing prototype because it can work on virtually any kind of materials.

The subtractive prototyping processes are widely used to produce prototype parts quickly. A subtractive prototyping process involved carving a solid block of material to reveal the shape of the desired object. In another word, material was removed from the raw part [53, 54]. Machining is the broad term used to describe removal of material from a work piece. Machining can be divided into the following categories [38]:

(1) *Cutting.* Material is removed from the surface of a work piece by producing chips. It generally involves a single-point or multi-point cutting tools, each with a clearly defined geometry. Examples of this category are turning, boring, drilling, milling, planing, shaping, broaching, sawing and filing.

(2) *Abrasive.* Material is removed from the surface of a work piece by producing tiny chips. It is used when the required surface finish and dimensional accuracy for a part are too fine, or the work piece material is too hard, of the work piece material is too brittle. Examples of abrasive processes are grinding, sanding, honing, lapping, buffing, polishing, shot-blasting and ultrasonic machining.

(3) *Non-Traditional.* It utilised electrical, chemical, thermal, and hydrodynamic to remove the material from a work piece. It is suitable for materials that are above 400 HB in hardness, very brittle, very flexible or delicate or difficult to be clamped onto the work holding devices. It is also used when the shape of the part is complex, such as internal and external profiles or small-diameter holes. If the surface finish and tolerances required are extremely stringent, non-traditional processes are the preferred methods. It is also suitable for products in which temperature rise or residual stresses are undesirable or unacceptable. Examples are chemical machining, electrochemical machining, electrochemical grinding, electrical-discharge

machining, laser-beam machining, hydrodynamic machining, abrasive-jet machining and electron beam machining.

2.8.2 Milling

Milling is defined as a machining process in which material is removed by the relative motion between a work piece and a rotating cutter having a single cutting edge [55] or multiple cutting edges [2]. In some cases, the work piece is held stationary while the rotating cutter is moved past it at a given feed rate (traversed) [56]. In other applications, both the cutter and the work piece are moved in relation to each other and in relation to the milling machine [57]. However, in the most common type of application, the work piece is advanced at a relatively low rate of movement or feed to a milling cutter rotating at a comparatively high speed, with the cutter axis remaining in a fixed position [56].

A characteristic feature of the multi-tooth cutter milling process is that each cutter tooth removes a small amount of metal with each revolution of the spindle [57]. Thus, a number of chips are produced in one revolution [2]. Milling can perform a wide variety of operations since both its cutter and work piece can be moved relative to one another, independently or in combination. Production of flat or contoured surfaces, slots, grooves, recesses, threads, and other configurations can be done by milling processes [56].

The main differences between milling and other machining processes are [57]:

(1) The interruptions in cutting that occur as the cutter teeth alternately engage and leave the work piece.

(2) The size of milling chips is relatively small.

(3) The thickness variation within each chip.

Chip thickness is different during the cut of each tooth because feed is measured in the direction of table motion (work piece moving into the cutter), while chip thickness is measured along the radius of the cutter [57].

Milling is one of the most universal, yet complicated machining process. The process has more variations in the types of machines used, work piece movements, and kinds of tooling than any other basic machining method. Important advantages of milling include high stock removal rates, producing relatively smooth surface finishes, and the wide variety of cutting tools that are available. Cutting edges of the tools can be shaped to form any complex or irregular surface [56]. Milling is most efficient when the work piece material's hardness is less than or equal to 25 HRC. But, steel at 35 HRC is commonly being milled. Steel with the hardness of 56 HRC has been successfully milled too [57].

2.8.3 Milling Methods

The major milling methods are peripheral, face and end milling [2]. These terms refer to the kind of cutter used and to the relationship of the milled surface to the spindle. A milling operation usually consists of a combination of two methods and only a number of cases where the three methods are clearly defined. The selection of a milling method for a specific application depends largely on the amount of material to be removed, size and shape of work piece and the configuration to be milled [57].

Peripheral milling is also called slab [2] or plain milling [55]. In peripheral milling, the axis of cutter rotation is parallel to the work piece surface to be machined [57]. Cutting the work piece with the teeth on the periphery of a cutter generates the milled surface [56]. Cutters applied in peripheral milling may have helical or straight teeth producing oblique or orthogonal cutting action [2]. Figure 2.8(a) [2] illustrated the helical teeth cutter in operation. The cutter rotation and work piece movement directions were also shown in the diagram.

Face milling is a process where the work piece surface is perpendicular to the axis of the cutter (the cutter axis can be horizontal or vertical) [55]. The milled surface is generated by the combined action of cutting edges located on the periphery and face of the cutter [56]. It is normally used for machining flat surfaces [57]. The flat surfaces have no relation to the contour of the teeth, except when milling a shoulder [56]. Face milling is illustrated in Figure 2.8(b) [2] with cutter rotation and work piece direction of motion.

Figure 2.8(c) [2] illustrates the end milling methodology. End milling cutters have cutting edges on the end faces as well as on the periphery [56]. Cuts can be produced with an end mill by using the side and end consecutively or simultaneously [57]. The cutter can rotate on an axis perpendicular or tilted with regard to the work piece. Thus, end milling can produce flat surfaces as well as various profiles [2].

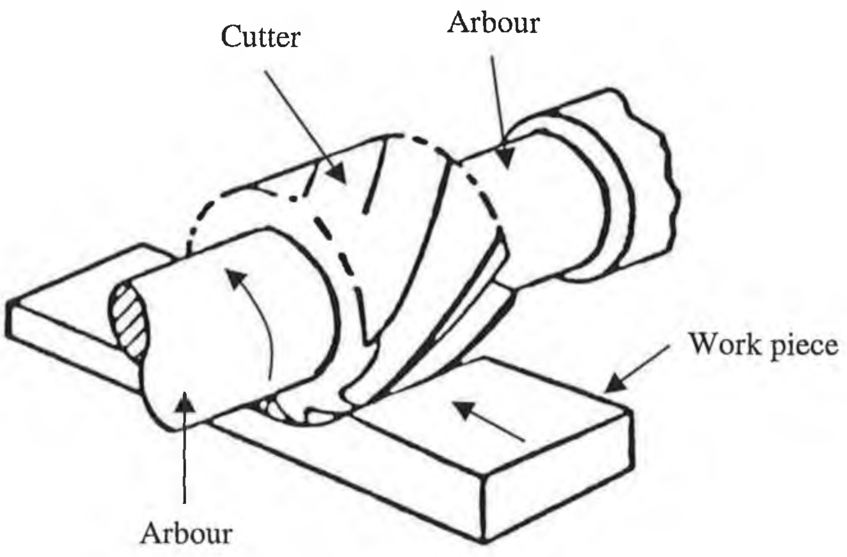


Figure 2.8(a) Peripheral Milling

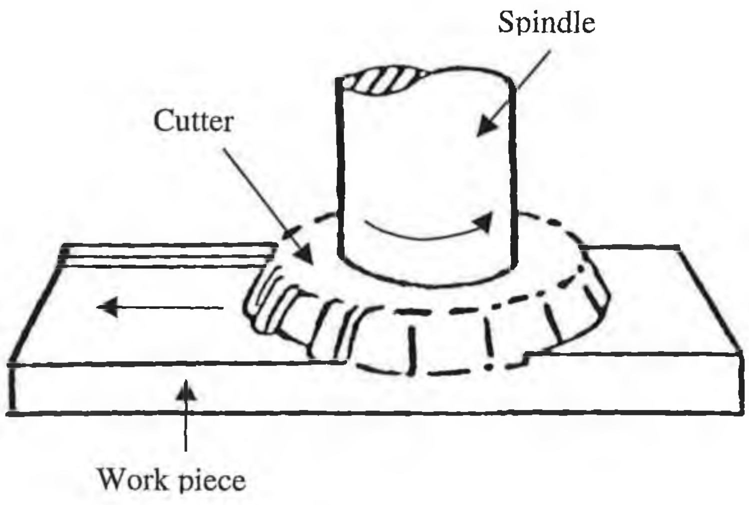


Figure 2.8(b) Face Milling

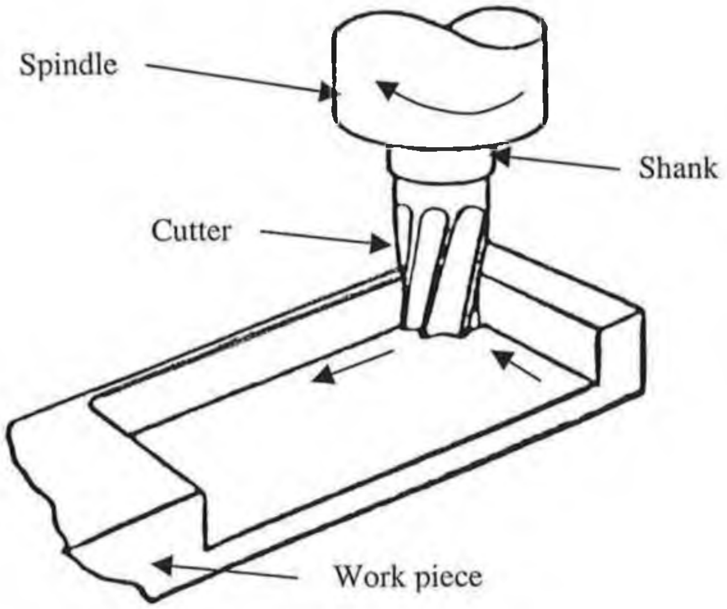


Figure 2.8(c) End Milling

Because the end faces of some end mills have cutting teeth, they can be used as a drill to start a cavity. Some of the end mills have hemispherical ends for producing curved surfaces, as in making dies. Hollow end mills have internal cutting edges and are also used for machining the cylindrical surface of solid round work pieces [2]. Some of the typical end milling cutters are shown in Figure 2.8(d) [58].

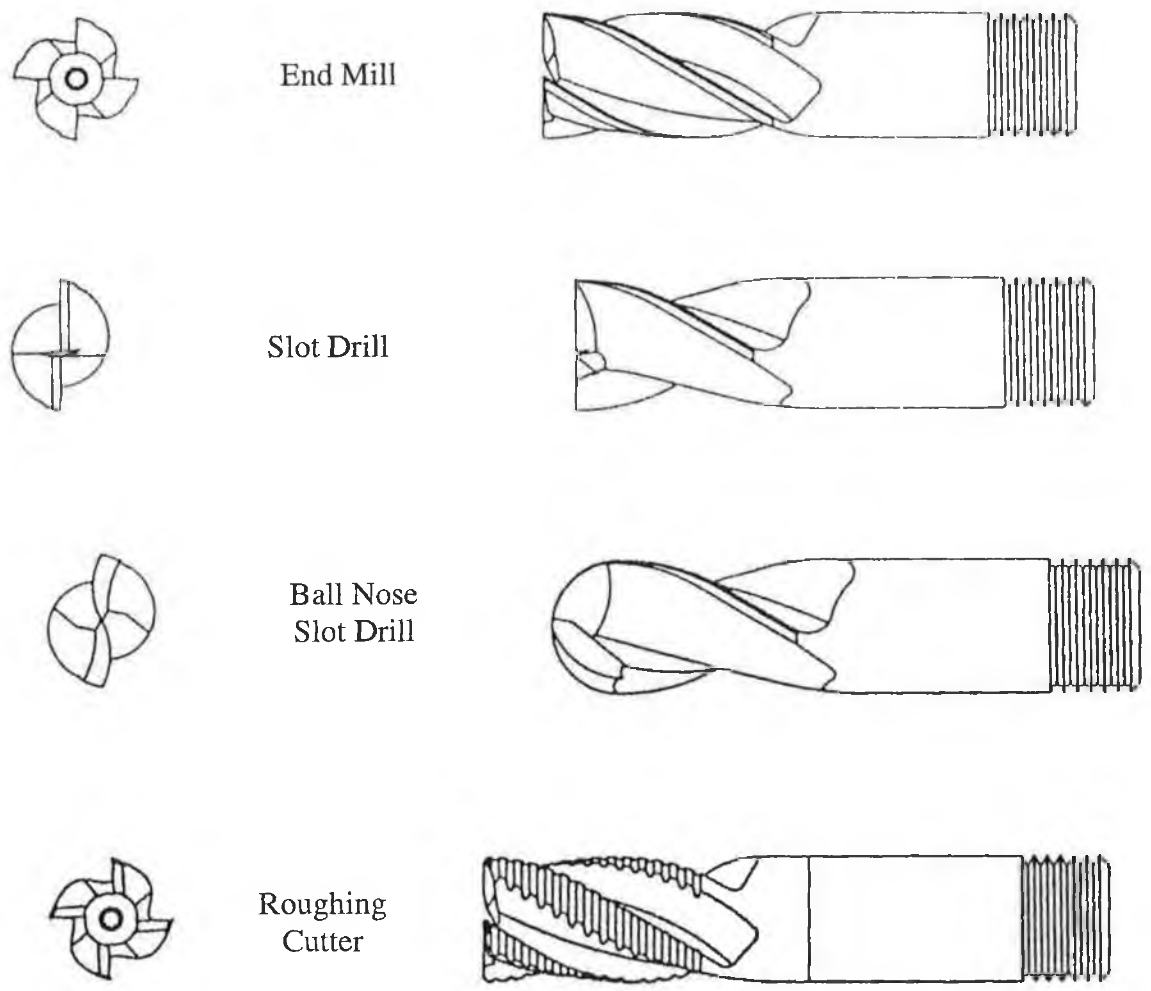


Figure 2.8(d) Typical End Mills

Based on Figure 2.8(d), end mill is designed for profile milling. A slot drill is used to produce precision slots and keyways, whereas a ball nosed slot drill (ball nosed cutter) is meant for producing slots and profiles with a radius form. A roughing cutter is designed for rapid metal removal rates due to the chip breaking cutting form. Among all the cutters, ball nosed cutter is the most versatile in application wise. It can be used to produce almost all kind of surfaces and features if compare to peripheral and face milling. As a result, complex surfaces like scupltured or free-form surfaces are normally produced by ball nosed cutter.

2.8.4 Additive Prototyping

Generally, all the additive prototyping processes are categorised under the term of rapid prototyping. Nobody really knows the exact person or research group that invented the subject of rapid prototyping. But, there were researchers in Europe, Japan and the United States working in this area during the 1970s and early 1980s [59]. There are also related patent applications that date from 1902 [60]. What is certain is that the rapid prototyping methodologies did have great impact and will continue to influence the design and manufacturing processes. Rapid prototyping processes are regarded as the missing link between design and manufacturing [61].

Rapid prototyping system has played an important role in the manufacturing industries because of the greater use of several technologies as stated below.

(1) CAD – Once the three-dimensional geometrical shape of the part has been unambiguously defined, then, the CAD model can be sliced into thin cross-sectional layers. The data on these layers can then be used by the rapid prototyping system to build the part in a real material. As a result, it can be claimed that CAD is the key to rapid prototyping.

(2) Laser or Light Processing of Materials – Most of the pioneers in rapid prototyping use laser to cut out the profile of individual layer or to activate a material phase change from a liquid to a solid. The cost and reliability of laser has been improved

significantly in the last two decades. Some of the techniques use a flood of ultraviolet (UV) light to solidify a polymer liquid layer.

(3) Additive Manufacturing – Almost all the rapid prototyping techniques rely on making objects by adding one layer of material to another until the final shape is produced [62]. This idea is not new since model makers have used this method to make large wooden models for so many years [63]. The method of building models from laminated wood avoids the gross distortion and cracking seen in models made from a single piece of wood. Laminations have also been assembled to make relief maps [64].

Various methods and machines have been developed to compete in the market. Five generic additive prototyping methods [65] are listed in Table 2.8(a), together with the vendors and quantity installed in the market [25]. The market shares in terms of units sold among different technological companies kept changing for the past few years. For an example, three years ago fused deposition modelling equipment became as popular as stereolithography machine in which both of them almost had the same percentage of market shares.

Table 2.8(a) Additive Prototyping Methods

Technology	Vendor	Installations
Stereolithography	3D Systems	469
	Misutbishi – CMET	43
	Sony – DMEC	28
	EOS GmbH	19
	Teijin – Seiki	3
	Mitsui Engineering	1
		Sub Total: 563
Laminated Object Manufacturing	Helisys	52
	Sparx	15
	Kira	10
		Sub Total: 77
Selective Laser Sintering	DTM	43
	EOS GmbH	4
		Sub Total: 47
Fused Deposition Modelling	Stratasys	44
	Sanders	1
		Sub Total: 45
Solid Ground Curing	Cubital	18
		Sub Total: 18
Total of all additive prototyping system installations as of June 1994: 750		

The founder of the additive prototyping market was 3D Sytems [66, 67]. Their additive prototyping system was called stereolithography and was patented by Charles Hull in 1984 [68]. In 1987, their first commercial unit, the Stereolithography Apparatus (SLA-1) was sold [69]. Stereolithography (SLA) is the most widely used additive prototyping system in the market [25, 70]. The SLA technique is illustrated in Figure 2.8(e) [71].

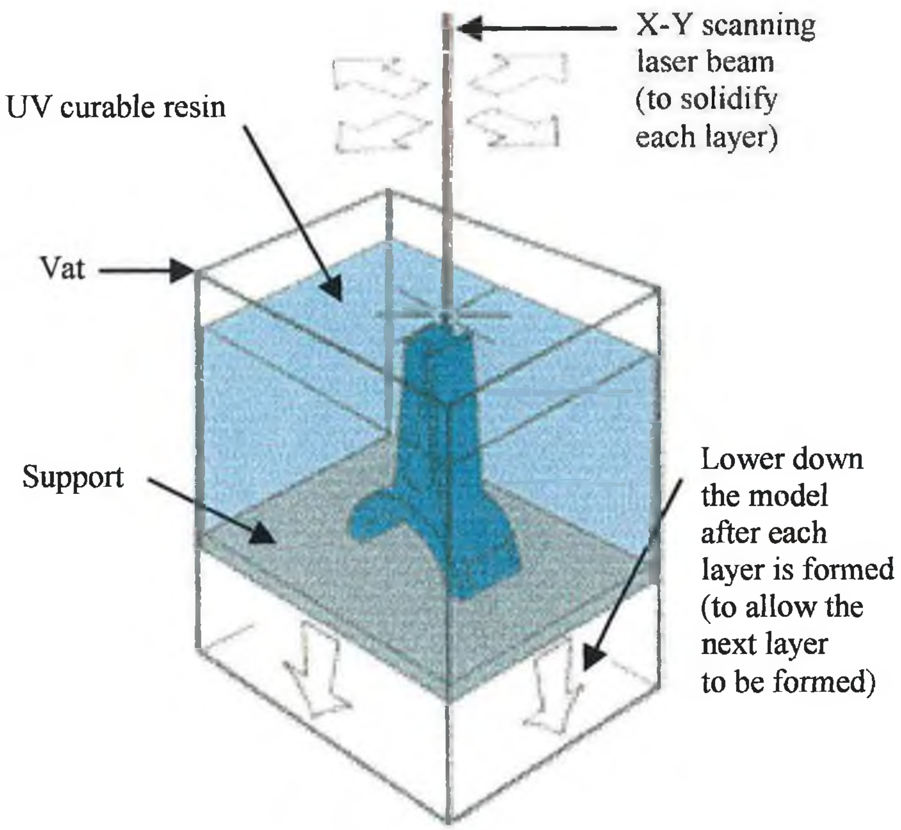

X-Y scanning laser beam (to solidify each layer)
UV curable resin
Vat
Support
Lower down the model after each layer is formed (to allow the next layer to be formed)

Figure 2.8(e) Stereolithography Technique

SLA uses local photocuring of specially developed liquid polymer resins as the process enabling technology. The solid geometry is defined in an .stl file. The object is gradually built, layer by layer from a pool of liquid photopolymer. The local photocuring is facilitated by a 200 micron, focused diameter beam of actinic light in a low power, He-Cd or Argon ion laser [6]. Hence, the SLA process is limited to producing solids geometry that are originated in specialised photocurable resins [72]. The other disadvantages include parts require support structures, some models can warp and resin handling needs care. However, the machines can run with unattended operation, the accuracy is often within ±0.1 mm, and the parts have very good detail and surface finish.

A Chicago company called Hydronetics was formed in 1985 to commercialise a system of building models by stacking sheets of steel foil and then bonding them together [73]. This process was known as laminated object manufacturing (LOM). The earliest patent in this area was filed by Dimatteo [74] where transducers measured a part and slices were simultaneously produced and stacked to give a model in different material. Another patent was filed in 1988 that further improved the LOM technique into producing coloured model by using sheets with coloured edges [75].

LOM technology uses gradual lamination of two-dimensional contours cut from paper sheet feed stock [76]. The method uses an X, Y position controlled mirror together with a 25 or 50 watt CO_2 laser to cut through paper feedstock ranging from 200 microns in thickness [77]. A computer driven platform is used as the working substrate to control z-position. The part is created by successively gluing together layers of foils that have been cut to the desired shape. The LOM technique is illustrated in Figure 2.8(f) [71].

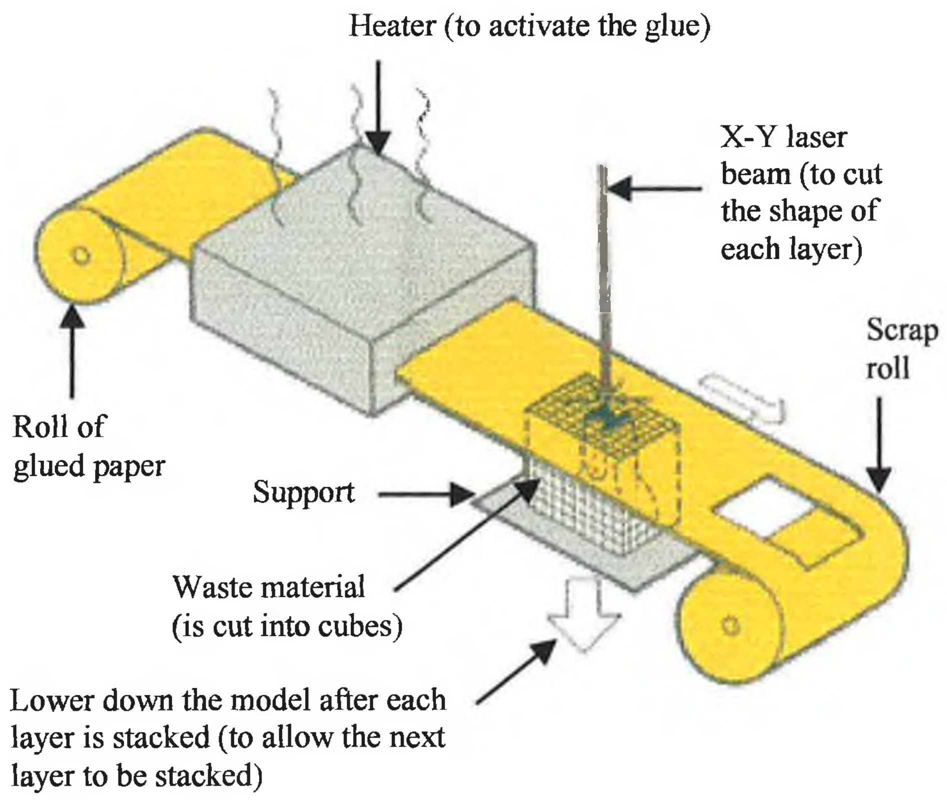

Heater (to activate the glue)
X-Y laser beam (to cut the shape of each layer)
Scrap roll
Roll of glued paper
Support
Waste material (is cut into cubes)
Lower down the model after each layer is stacked (to allow the next layer to be stacked)

Figure 2.8(f) Laminated Object Manufacturing Technique

LOM process is common in foundry applications. The difficulty in removing the excess material restricts the range of parts that can be manufactured by LOM [76]. Compared to the other rapid prototyping techniques, it gives a poorer surface finish, the machine has been reported as unreliable and the parts absorb moisture. Its advantages are that there is no post-curing, no supports are needed, material will not experience phase change and it is a simple process.

Carl Deckard and Professor Joe Beaman jointly developed selective laser sintering technique in 1986 [78]. Deckard started work on that method during his Masters work [79] and subsequently obtained a PhD in that field. The first laboratory equipment was made using an abandoned machine and involved equipment costing about US$30,000 [80]. The work was commercialised by Nova Automation Corporation which later became DTM [81, 82]. DTM [83] holds the patent right [84] on this technique [78]. The SLS technique is illustrated in Figure 2.8(g) [71].

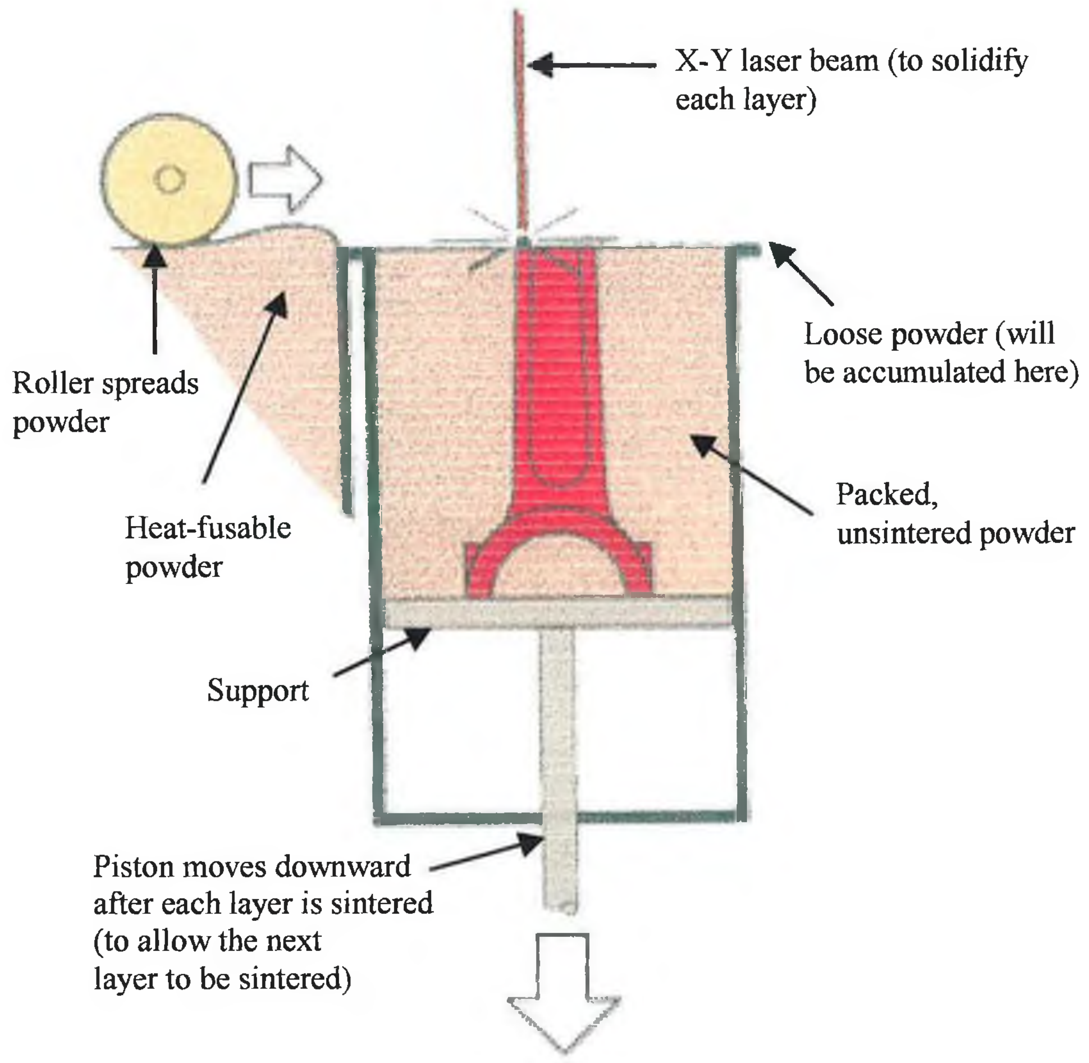

X-Y laser beam (to solidify each layer)
Roller spreads powder
Loose powder (will be accumulated here)
Heat-fusable powder
Packed, unsintered powder
Support
Piston moves downward after each layer is sintered (to allow the next layer to be sintered)

Figure 2.8(g) Selective Laser Sintering Technique

Selective laser sintering (SLS) method uses modulation of a focused laser to selectively fuse or sinter powder feed stock materials [85]. SLS uses a computer controlled, high-power laser to sinter thermoplastic powder (partial melting), one layer at a time, and builds the complete part from CAD data of the sliced part-geometry. A number of different materials like investment castings waxes, nylons, polycarbonate, metals, and ceramics can be processed by this method.

The drawbacks of SLS are poor product surface finish and the parts are quite porous [86]. Besides, it requires long period of time to heat up and cool down the material chamber after building and supports are needed on some materials. However, its main advantages are that post-curing is not needed, nylon or polycarbonate does not need support, tough parts can be produced and it will be possible to use ceramics or metals in the future.

S. Scott Crump developed fused deposition modelling (FDM) technique in 1988 and a patent was filed the following year [87]. FDM relies on the CNC extrusion and rapid cooling of ribbon like, molten thermoplastic materials as its enabling technology [88]. The thermoplastic filament is fed through a heated extruding head. The filament which are materials like wax or nylon, melts at a temperature just above its solidification state before depositing on a platform to produce the part. The FDM technique is illustrated in Figure 2.8(h) [71].

The accuracy of the process depends on the physical properties of the molten material. The surface finish capability of FDM is inferior to other additive prototyping methods [89]. Furthermore, it needs support structures, there is poor strength in the vertical direction and it is slow on bulky parts. Nevertheless, it is easy to change materials, it is a simple equipment, is fast on small hollow parts and there are a good variety of materials which can be used.

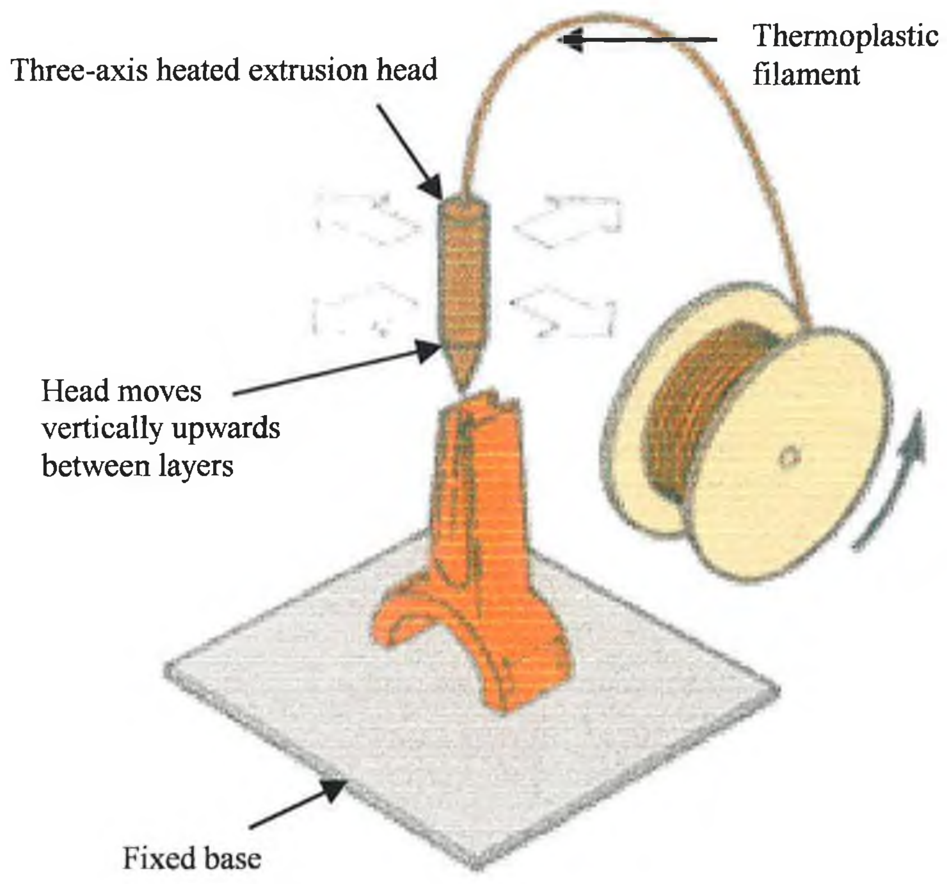

Three-axis heated extrusion head
Thermoplastic filament
Head moves vertically upwards between layers
Fixed base

Figure 2.8(h) Fused Deposition Modelling Technique

Solid ground curing (SGC) was developed by Itzchak Pomerantz in Israel at Cubital Limited [90]. SGC also called photomasking. It is a hybrid system that has the characteristic of both additive and subtractive prototyping technologies. Here, the selective curing of liquid resins by masked, UV radiation is the enabling technology. Multiple layers form the product. A photomask is used to cover the photopolymer liquid in each layer and cured in a few seconds by a strong UV lamp. The unexposed liquid is then removed and the voids are filled with molten wax to support the next layer. The steps are repeated until the entire part is made [91]. The SGC technique is illustrated in Figure 2.8(i) [71].

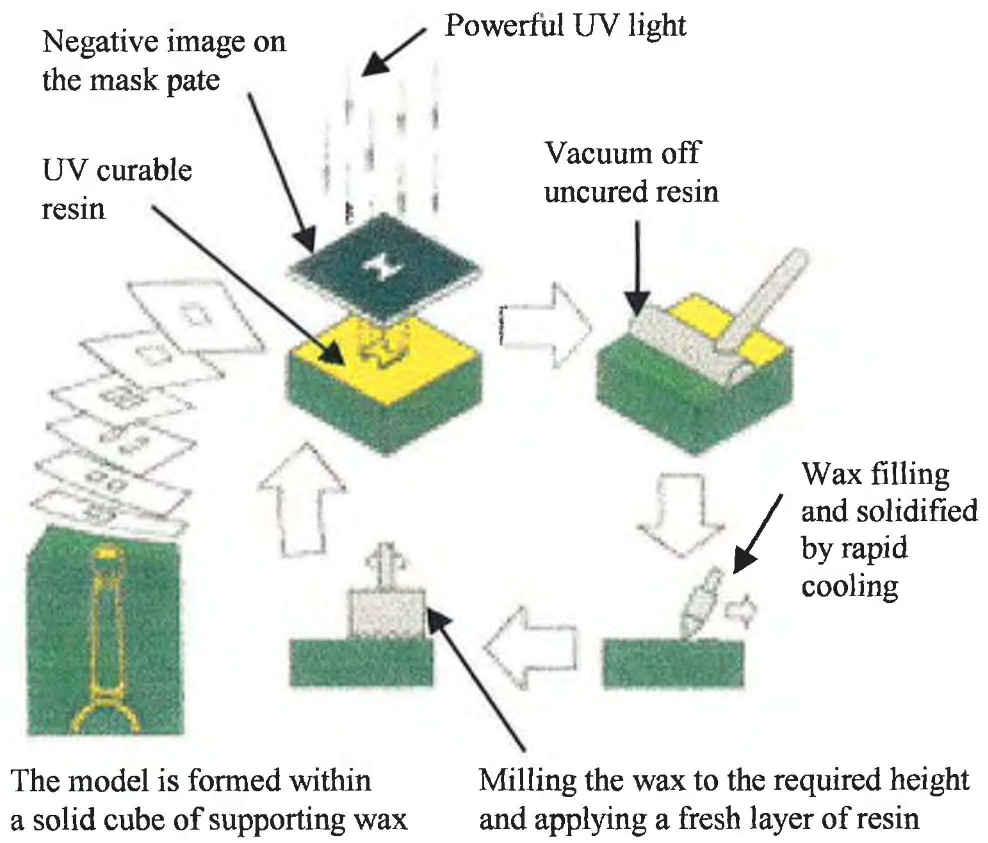

Negative image on the mask pate
Powerful UV light
UV curable resin
Vacuum off uncured resin
Wax filling and solidified by rapid cooling
The model is formed within a solid cube of supporting wax
Milling the wax to the required height and applying a fresh layer of resin

Figure 2.8(i) Solid Ground Curing Technique

A major disadvantage with the SGC is that it requires attended operation, there is excessive waste of resin and wax, and excessive down-times have been reported on commercial machines. But, this kind of system has a high output, no supports are needed on the parts, nesting of parts above each other is common and it has predictable build times.

CHAPTER 3: EQUIPMENT

3.1 INTRODUCTION

This chapter is concerned with the basic hardware of the precision robotic manipulator based rapid prototyping system. The whole system consists of a personal computer, PC-23 indexer, KS-drives, a. c. brushless servomotors, a four degrees of freedom precision manipulator and a ball nosed end milling equipment. The general layout of the equipment is shown in Plate 3.1(a).

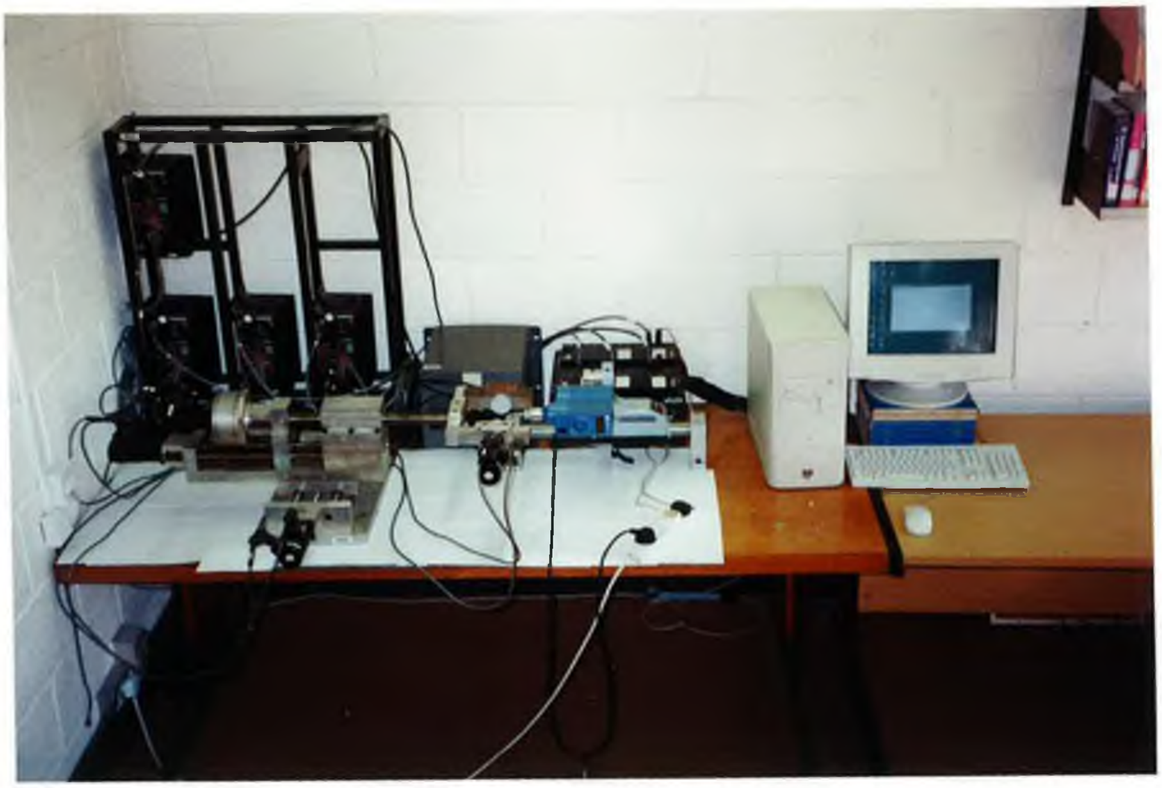


Plate 3.1(a) General Layout of Equipment

A personal computer is used to control the system by sending commands and for receiving responses from the PC-23 indexer. The indexer will in turn communicate with the KS-drives for controlling the a. c. brushless servomotors. The a. c. brushless servomotors will drive a four degrees of freedom precision robotic manipulator. The manipulator will feed the work material, polystyrene cylindrical block, to the ball nosed cutter for milling three-dimensional complex shaped objects. The system can also be used for additive prototyping since it is a multipurpose precision robotic manipulator. The control system configuration is shown diagrammatically in Figure 3.1(a).

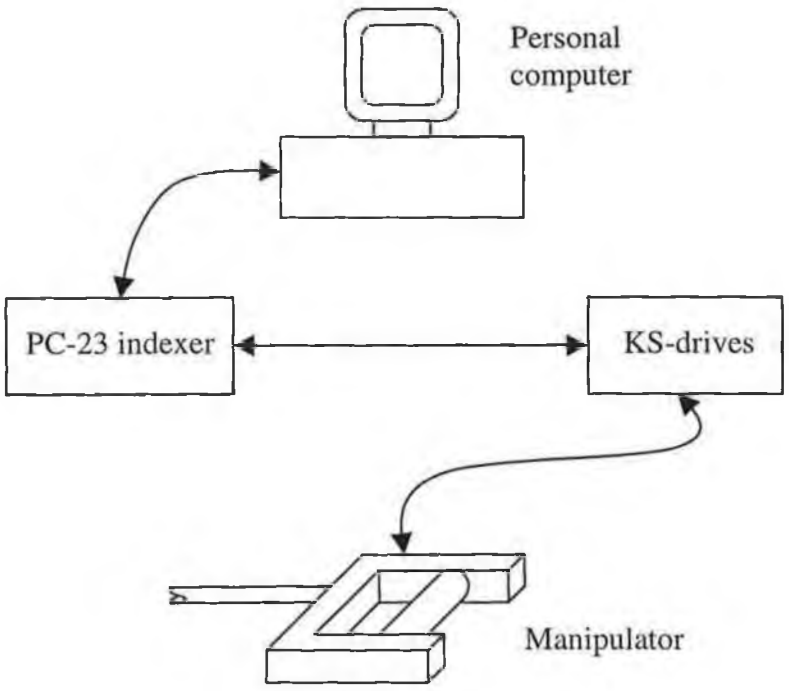


Figure 3.1(a) Control System Configuration

Thus, the following subtopics will cover:

(1) *Precision Manipulator.* A four degrees of freedom precision robotic manipulator's structural design, capabilities and modification will be elaborated.

(2) *Interfacing*. PC-23 indexer's general functions, programming aspect, motion control, and related issues will be discussed. KS-drives' functions and capabilities together with the a. c. brushless servomotors will be described.

(3) *Subtractive Prototyping Process*. The construction and assembly of a customised ball nosed end milling equipment and the milling material will be illustrated.

3.2 PRECISION MANIPULATOR

The precision manipulator has four degrees of freedom, namely two translational and two rotational motions [92]. Plate 3.2(a) illustrated the precision manipulator clearly.

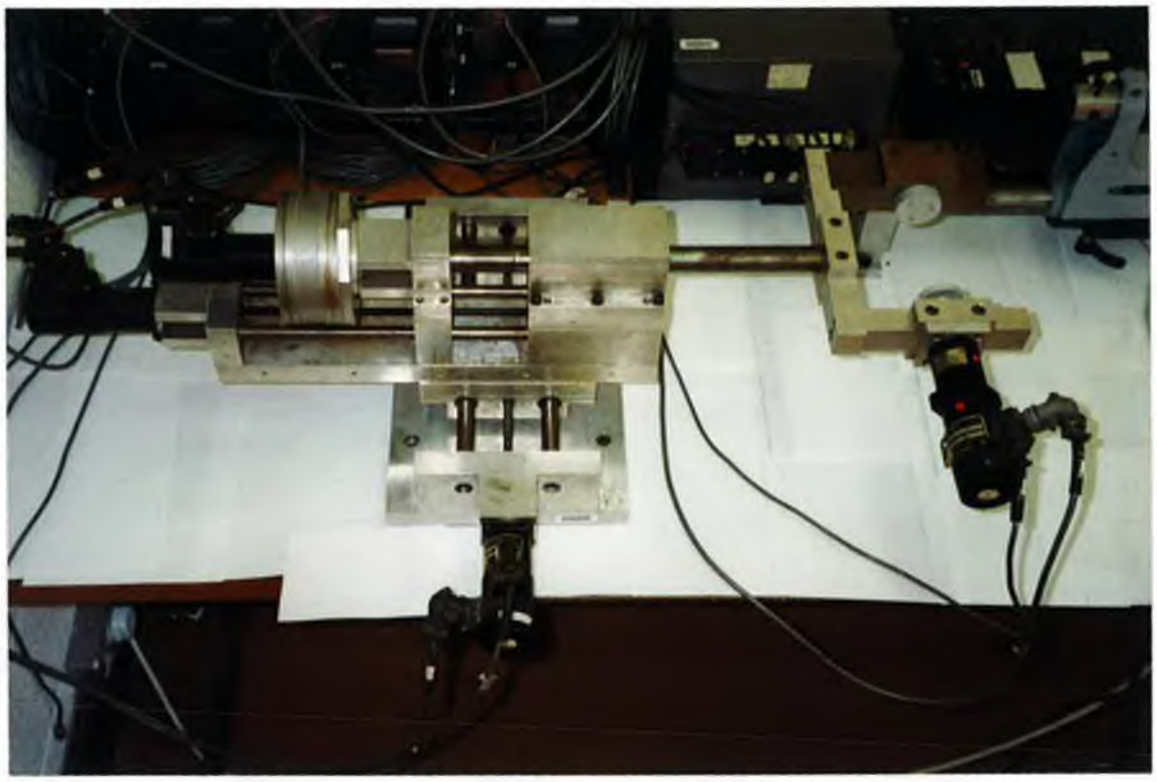


Plate 3.2(a) Four Degrees of Freedom Precision Manipulator

The manipulator is described further in Figure 3.2(a). In the diagram, the plan and side views of the manipulator are shown. The length of the manipulator is 1270 mm. Its width and height are 616 mm and 440 mm respectively. The four degrees of freedom are y linear motion axis, x linear motion axis, roll (rotation around y-axis) and pitch (rotation around x-axis). The direction of motion for each motion axis is also shown in the figure.

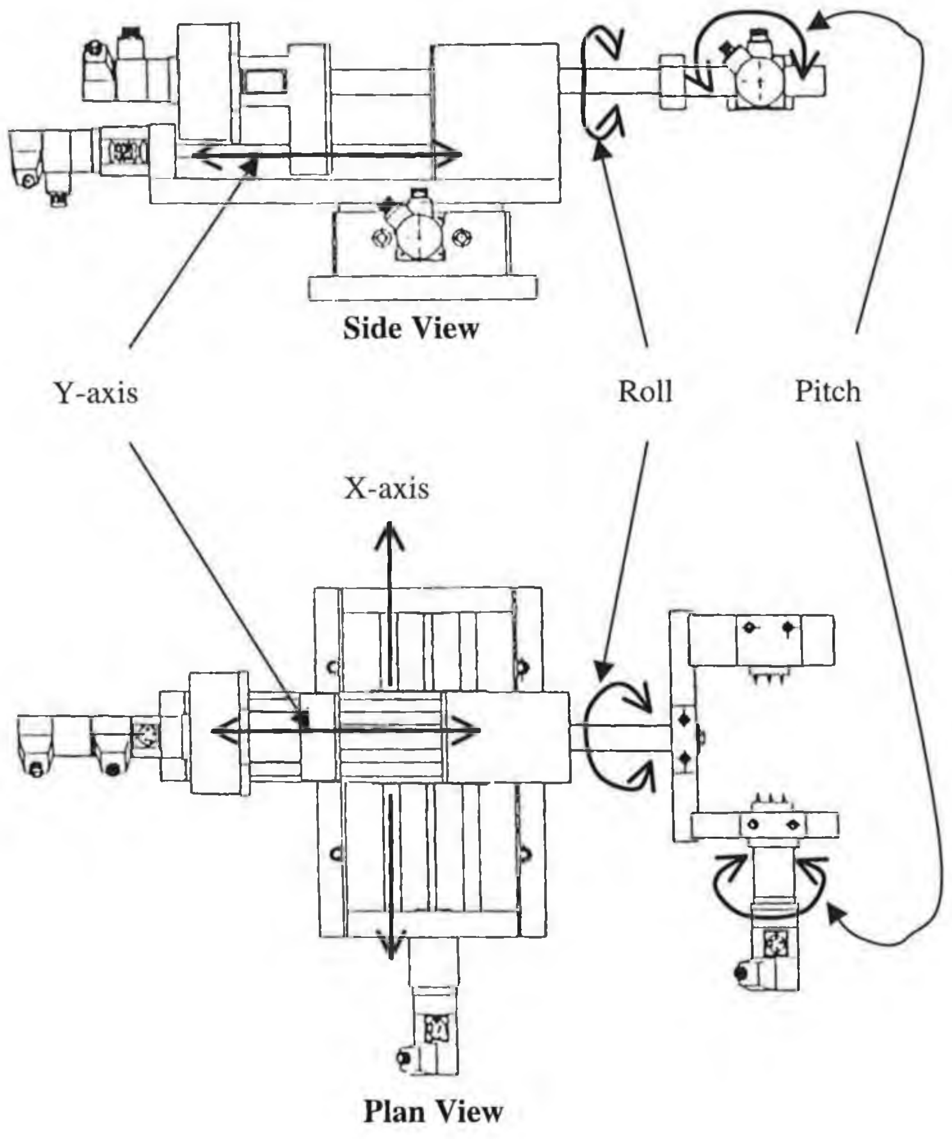


Figure 3.2(a) Side and Plan View of Manipulator

The precision manipulator is a general-purpose robot. It can be used in subtractive prototyping processes like wire electro-discharge machining or other kind of traditional machining process like milling. It can also be used for additive prototyping process like semi-liquid deposition process. The weight of the manipulator is about 55 kg. Aluminium alloy (BS HE30 TF) that has the tensile strength of 280 MN/m^2 was used to manufacture the main parts of the manipulator. The diameters of the steel shafts used are 16 mm, 20 mm and 30 mm, having hardness of 60 HRC.

3.2.1 Manipulation Around X-Axis

Figure 3.2(b) shows the plan view of the manipulation unit around x-axis (pitch). The direction of motion is also shown.

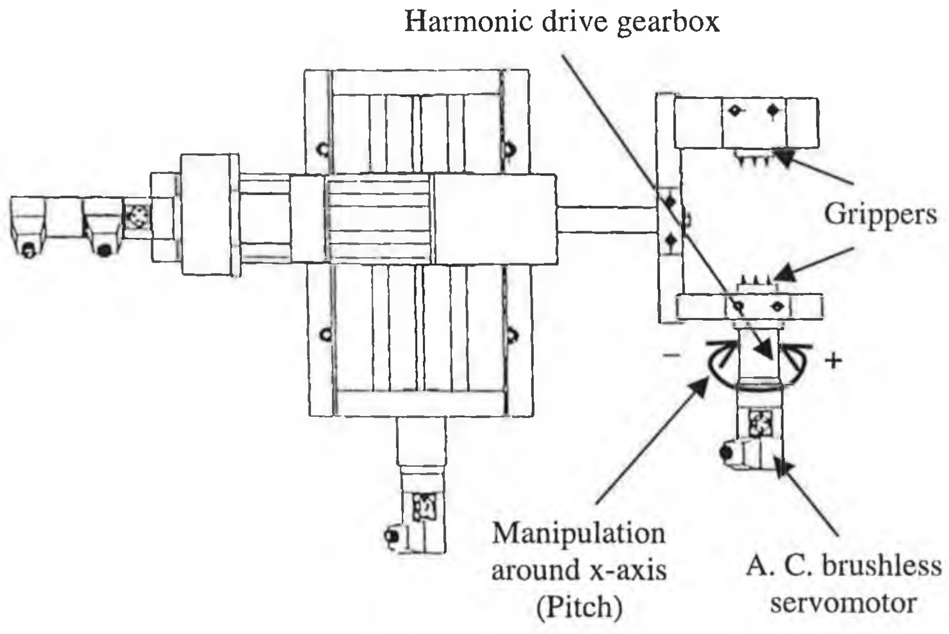

Harmonic drive gearbox
Grippers
−
+
Manipulation around x-axis (Pitch)
A. C. brushless servomotor

Figure 3.2(b) Manipulation Around X-Axis

The manipulation unit around x-axis consists of an a. c. brushless servomotor, harmonic drive gearbox and a pair of three-pin grippers. The range of the work piece size that can be held by the grippers is 120 mm to 125 mm in length and 40 mm to 150 mm in diameter. An a. c. brushless servomotor (KS 210) in conjunction with a harmonic drive gearbox (HDUC 14) having a gear ratio of 100:1 are used to drive the grippers. The motor resolution is set at 5000 steps per revolution. The motor for the pitch manipulation can travel an angular distance of 7.2 x 10^{-4} degrees per full motor step.

3.2.2 Manipulation Along and Around Y-Axis

Figure 3.2(c) shows the side view of the manipulation unit along and around (roll) y-axis together with motion directions.

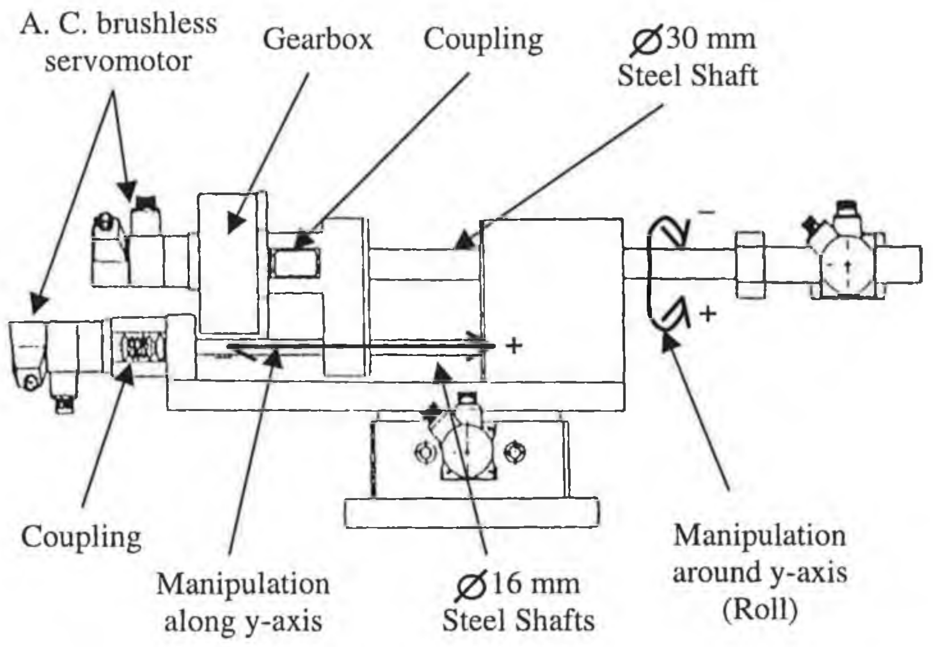

A. C. brushless servomotor
Gearbox
Coupling
Ø30 mm Steel Shaft
–
+
+
Coupling
Manipulation along y-axis
Ø16 mm Steel Shafts
Manipulation around y-axis (Roll)

Figure 3.2(c) Manipulation Along and Around Y-Axis

The manipulation unit along y-axis consists of an a. c. brushless servomotor, a coupling, a lead screw with pitch of 1.5 mm and two 16 mm diameter steel shafts. An a. c. brushless servomotor (KS 220) drives this unit. The lead screw is connected to the motor shaft by means of a flexible coupling (Compumotor No. CPG. 2 – 6). The steel shafts are used for ensuring the motion is straight and supporting the manipulation unit around the y-axis (roll). The motor resolution is set at 5000 steps per revolution. The motor can travel a linear distance of 3.0 x 10^{-4} mm per full motor step. Total linear distance that can be travelled by the unit is 130 mm.

The manipulation unit around y-axis (roll) is directly located above the manipulation unit along the y-axis. It consists of an a. c. brushless servomotor, a gearbox, a coupling and a 30 mm diameter steel shaft. An a. c. brushless servomotor (KS 220) in conjunction with a gear box (Drivematic No. SA1002) having gear ratio of 18:1 generated the roll motion around y-axis. The gearbox is attached to a 30 mm diameter steel shaft by a rigid coupling. The motor resolution is set at 5000 steps per revolution. The motor for roll motion can travel an angular distance of 4.0 x 10^{-3} degrees per full motor step. At the end of the steel shaft is the manipulation unit around x-axis (pitch).

3.2.3 Manipulation Along X-Axis

Figure 3.2(d) shows the plan view of the manipulation unit along x-axis together with motion direction. This unit consists of an a. c. brushless servomotor, a coupling, a lead screw with pitch of 1.5 mm and a pair of 20 mm diameter steel shafts. The a.c brushless servomotor (KS 220) is connected to the lead screw through the flexible coupling (Compumotor No. CPG. 2 – 6). The steel shafts are for supporting the manipulation unit around x-axis as well as the manipulation units along and around y-axis. They are also for ensuring straight motion. The maximum distance of 240 mm can be travelled by this manoeuvring part along x-axis. The motor is set at 5000 steps per revolution. The motor can travel a minimum linear distance of 3.0 x 10^{-4} mm per full motor step.

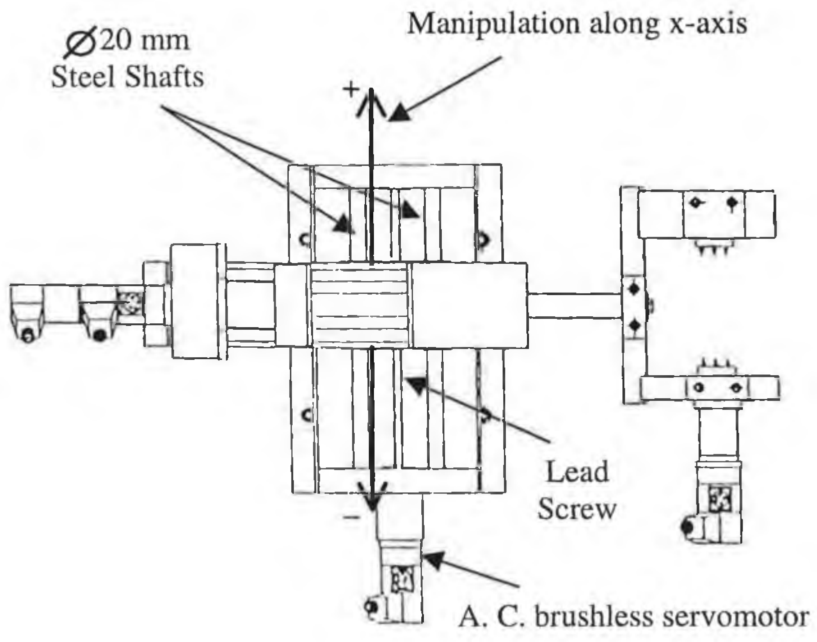


Figure 3.2(d) Manipulation Along X-Axis

3.3 INTERFACING

One of the important components in producing three-dimensional complex shaped product from the robotic manipulator based rapid prototyping system is the interface system. The interface system consists of PC-23 indexer, KS-drives and the a. c. brushless servomotors. They are located between the personal computer and the precision manipulator. Figure 3.3(a) illustrated the interface system schematically where the system is bounded by the thickest dotted line.

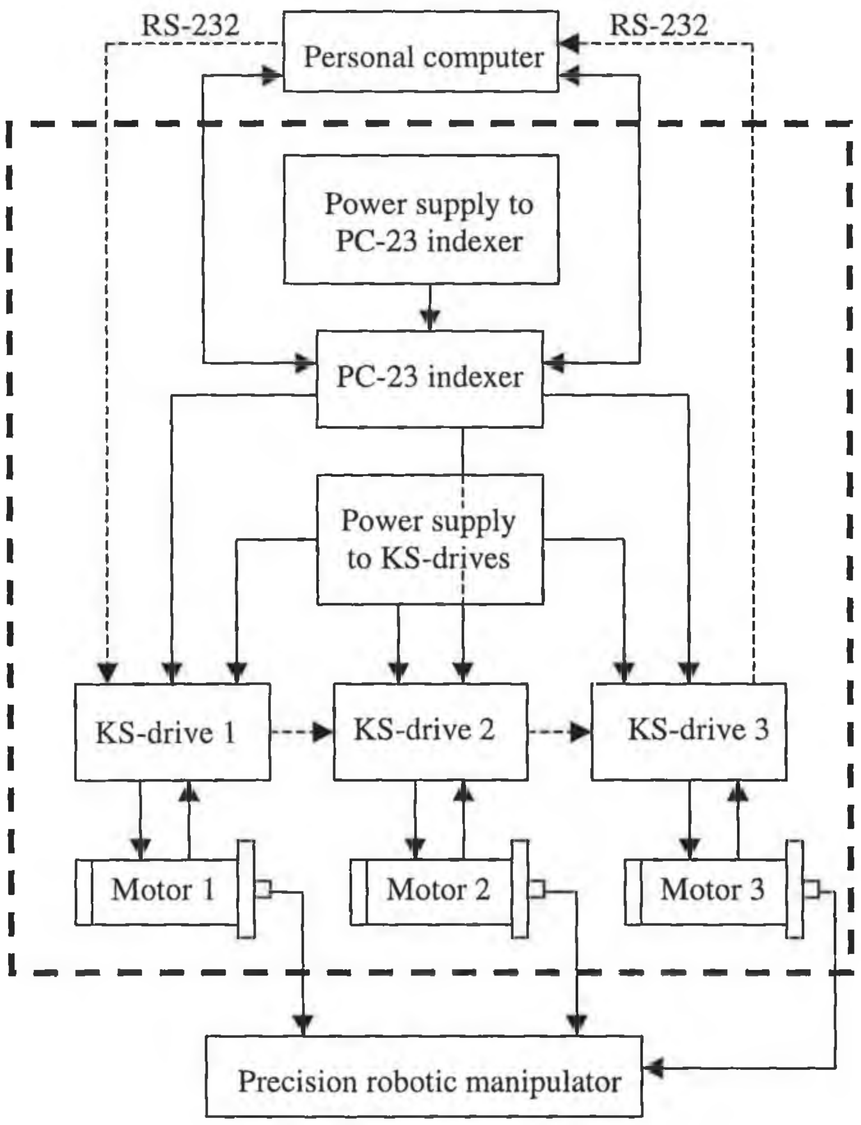


Figure 3.3(a) Interface System

As shown in Figure 3.3(a), a personal computer was used to send commands and receive responses from PC-23 indexer. The indexer in turn communicates with three

KS-drives. The KS-drives control three a. c. brushless servomotors that drive the precision robotic manipulator to produce the three-dimensional complex shaped object. For functional purposes, two external power supply modules are needed for the indexer and three drives. The KS-drives' performance can be tuned via the RS-232 link. Currently, it is still an open-loop system. It can be further improved by implementing a closed-loop control.

3.3.1 PC-23 Indexer

PC-23 indexer [93] uses a 16-bit microprocessor for controlling the motion of up to three motor axes, independently or simultaneously. The indexer is used with an IBM microcomputer (PC, XT or AT) or compatible and suitable for any kind of drive systems that can accept pulsed control signals. It is used for controlling velocity, distance and linear acceleration parameters. The indexer's performance, physical, environmental and electrical specifications are presented in Table 3.3(a).

The indexer receives acceleration, velocity, position and direction information in ASCII (American Standard Code for Information Interchange) characters from the personal computer's control program, and uses that information to produce motion profile command signals for the drive system. The drive commands are in the form of "step" pulses. They can be issued at controlled rates of up to 500000 steps per second to the drive system.

Table 3.3(a) PC-23 Indexer's Performance, Physical, Environmental and Electrical Specifications

Parameter	Value
Performance	
Stepping accuracy:	± 0 steps from preset total
Velocity accuracy:	± 0.02% of maximum rate above 0.01 revolutions/second
Velocity repeatability:	± 0.02% of maximum rate
Velocity range:	0.01 - 20.00 revolutions/second (25000 steps/revolution)
	0.01 - 100.00 revolutions/second (5000 steps/revolution)
Acceleration range:	0.01 - 999.9 revolutions/second^2 (25000 steps/revolution)
	0.05 - 4999.95 revolutions/second^2 (5000 steps/revolution)
Position range:	0 - 99999999 steps (all resolutions)
Physical	
Main Circuit Board	
Length:	≈ 338 mm
Width:	≈ 125 mm
Physical	
Adaptor Box	
Length:	≈ 323 mm
Width:	≈ 156 mm
Height:	≈ 29 mm
Net weight:	≈ 0.74 kg
Environmental	
Operating:	0 to 50°C
Humidity:	10 to 95% (non-condensing)
Storage:	-30 to 85°C
Electrical	
Power:	5 VDC (Bus)

The indexer commands enabling the precision control of motor rotation for up to three axes independently or simultaneously. The motors can be controlled to rotate to a certain precise position and stop; rotate at a constant velocity and/or acceleration; alternate back and forth between two angular positions; or use a sequential combination of such moves.

Velocity parameters are expressed as the number of revolutions per second. Acceleration parameters are expressed as the number of revolutions per second squared. Distance parameters are given in characters representing motor increments (or decrements) or steps. The indexer can control motors of virtually any resolution.

The PC-23 indexer consists of two parts, which are a main circuit board and an adaptor box. The main circuit board is incorporated with the personal computer (PC) via ISA slot in the motherboard. The cable harness with four flat cable connectors from the adaptor box run through the slot in the PC's access panel and plugged into the main circuit board. The adaptor box is external to the personal computer and connected to the KS-drives. Plate 3.3(a) shows the adaptor box with wire harness connecting to the main circuit board which is in turn situated in the personal computer's motherboard.

Plate 3.3(a) PC-23 Indexer

In order to communicate with the PC-23 indexer, the personal computer must know where to write instructions and read responses. As a result, the indexer must have an address that does not conflict with typical devices that reside on the PC's I/O bus such as graphics adaptors, disk drives and other peripheral devices. The address can be set to any number that the personal computer will recognise as valid.

The indexer address can be set with the DIP switches on the main circuit board. The switches consists of eight switches representing a binary number. The switches are "negative true". Any switch in the position marked "ON" has a binary value of zero. Switches that are at "OFF" positions are having a non-zero binary value. The sum of the binary values of switches 1 through 8 is the main circuit board's base address.

The base address for the PC-23 indexer is set as 300 Hex. The indexer will occupy four address locations. However, only two of the address locations are significant. The data register of the indexer is at the even address locations, namely 300 Hex. Whereas, the control and status register of the indexer is at the odd address location, which is 301 Hex. The values assigned to the eight switches of the main circuit board's switch package are shown in Table 3.3(b). The selected base (board) address for the PC-23 indexer is also shown in the table.

Table 3.3(b) Assigned Switches Values and Base Address Setting

Switch No.	**Address "Bit"**	**Binary Value (OFF)**		**PC-23 Indexer Setting**
		Decimal	**Hex**	
8	2	4	4	ON
7	3	8	8	ON
6	4	16	10	ON
5	5	32	20	ON
4	6	64	40	ON
3	7	128	80	ON
2	8	256	100	OFF
1	9	512	200	OFF

The adaptor box has three 25-pin connectors for each axis. Only the Motor Driver connection is absolutely required for controlling any axis. The Motor Driver will provide the output signals to the KS-drive to step the motor, change direction, de-energise the motor and so on. All signals connected to the adaptor box are optically isolated from the computer. As a result, power is required to drive the isolated portion of the circuit. An external fixed power supply of 5 VDC at 10 Amperes is connected to one of the Auxiliary connectors of the adaptor box. The positive terminal is wired to pin 23 and the negative terminal is connected to pin 21.

3.3.2 Programming of PC-23 Indexer

Any programming language can be used to program the PC-23 indexer's operations. The motion control program only need to have the capabilities of reading information from and writing digital data to the I/O bus of the personal computer. Motion control commands and responses are transferred through the Input Data Buffer (IDB) and Output Data Buffer (ODB) at the indexer's base address, 300 Hex. Interface control commands and status information are transferred through the Control Byte (CB) and Status Byte (SB) at one address location above the base address, 301 Hex.

ODB and SB are read-only registers, whereas IDB and CB are write-only registers. Indexer commands consists of "string" or sequences of ASCII (American Standard Code for Information Interchange) characters. A register or buffer is a temporary storage area for holding only one character (one 8-bit "byte") at any occasion. As a result, passing a commands to the indexer means transferring each character in the command one at a time.

Each character transfer requires that the sender notifies the receiver that a character is ready, and that the receiver notifies the sender that the character has been received. The notification process involves the 8-bit CB and SB registers. Each bit is a "flag" with a specific meaning. The CB will allow certain operating conditions to be set and the SB will report the others. Signalling the indexer involves setting or clearing (resetting) the control bits or flags, which means forcing them to a binary value of one or zero, respectively.

The available Status and Control Bytes' flags and their definitions are shown in Table 3.3(c) and Table 3.3(d) respectively. Thus, the notification process is making use of the single bit flags of the 8-bit SB and CB registers being set high (1) or low (0) to denote the ready or busy condition of the indexer.

Table 3.3(c) Status Byte Format

Bit (Flag)	Definition
0	If axis 2 stopped, this bit will be set.
1	When axis 1 stopped, this bit is set.
2	If this flag is set, axis 3 is stopped.
3	This bit is set when the ODB contains an output character for the host, signalling the host to read the information it contains.
4	This flag is set when the IDB is ready, telling the personal computer it may write a character to the IDB.
5	Setting this bit will inform the personal computer that the Watchdog Timer of the indexer has timed out, possibly indicating an internal failure from which it cannot recover. Resetting the indexer can clear this bit.
6	If this bit is set, the host will be informed that a conditional interrupt has been "armed" and that the condition has occurred.
7	Reserved

Table 3.3(d) Control Byte Format

Bit (Flag)	Definition
0	Setting this bit will indicate that the Binary Mode of data input for the TD mode of contouring is under way.
1	Unused
2	Setting this flag will cause the indexer's Watchdog Timer to time out and stop. It forces a reset of hardware. Cycling power or restarting the timer can clear the reset condition.
3	Setting this bit will tell the indexer that its interrupt signal to the personal computer has been noted and is no longer required.
4	Setting this flag will inform the indexer that a command character has been put into the IDB. The indexer then clears the bit 4 of the SB to indicate that the IDB is not available, reads the IDB's character, and then sets SB's bit 4 to indicate to the host that the IDB is ready for new character again.
5	It is for restarting the Watchdog Timer. First, it must be reset, then the timer will start up when the bit is set again. This bit should never be toggled unless the timer has timed out.
6	It is for resetting the hardware interrupt latch and thus the interrupt output. The interrupt output cannot be reset unless the interrupt is first acknowledged with bit 3 above. These bits should be cleared during reset or interrupt acknowledged.
7	Setting this bit will tell the indexer that the host has received a response character that was previously placed in the ODB by the indexer, and a new character may be placed in the ODB.

The PC-23 indexer is designed to operate motor axes in a fashion largely independent of the personal computer. It only requires a small number of high level commands and interaction. The interaction is almost exclusively in the form of strings and characters rather than numbers. Thus, knowledge of string handling of a particular programming language is needed when constructing the control program.

Regardless of what the control program's intended application, it must have the following subroutines (in order of importance):

(1) Reset the indexer.

(2) Send a command string to the indexer.

(3) Receive a character string from indexer.

The step by step procedures for resetting the PC-23 indexer is:

(1) Write 64 Hex to the Control Port.

(2) Read the Status Port until (the Status Byte AND 20 Hex) = 0.

(3) Write 40 Hex to the Control Byte.

(4) Write 60 Hex to the Control Byte.

(5) Read the Status Port until (the Status Byte AND 16 Hex) ≠ 16 Hex.

The step by step procedures for writing a character to the PC-23 indexer is:

(1) Read the Status Port until (the Status Byte AND 10 Hex) = 0.

(2) Write the ASCII character to the Data Port.

(3) Write 70 Hex to the Control Port.

(4) Read the Status Port until (the Status Byte AND 10 Hex) > 0.

(5) Write 60 Hex to the Control Byte.

(6) Read the Status Port until (the Status Byte AND 10 Hex) = 0.

The step by step procedures for reading a character from the PC-23 indexer is:

(1) Initialise the ASCII variable to null (0).

(2) Read the Status Port until (Status Byte AND 8 Hex) = 0.

(3) Read the Data Port into the ASCII variable.

(4) Write EO Hex to the Control Port.

(5) Read the Status Port until (the Status Byte AND 8 Hex) > 0.

(6) Write 60 Hex to the Control Port.

(Note: AND is the bitwise logical and.)

3.3.3 Personal Computer and PC-23 Indexer

As stated in section 3.3.1, the PC-23 indexer is meant for PC/XT, AT or any IBM compatible personal computers. What will happen if the personal computer (PC) is different from the above stated models? What is the effect of using high-end personal computer? What are the counter measures to be applied so that powerful PC can be used? The answers to the above questions are stated in the following paragraphs.

In this project, 3 kinds of PC were tried to link with the indexer and they were PC/XT, 486DX33 and Pentium II 300 PC. By following the manufacturer's advice, using PC/XT is the correct choice. But, its processing speed is only 4.77MHz. Further more, its memory (RAM) and hard disk capacity are only 640 Kbytes and 20 Mbytes respectively. As a result, the PC/XT's processing speed is very slow compare to the available PC on the market. Besides, such a low memory and hard disk capacity computer cannot accommodate other software like AutoCAD and Mechanical Desktop (computer-aided design package). As a result, a 486DX33 PC was used to replace the PC/XT computer.

Initially, using the 486DX33 PC to link with the indexer proved to be feasible in functionality and performance aspects. Also, software such as AutoCAD R11 (DOS version) can be used on this PC. But, further testing revealed the weakness of the communication between the 486DX33 PC and the indexer. The system began to deviate from its normal functionality. The motion axes were not moving in a synchronous manner and the whole system hanged in the middle of the process.

The cause of the problem is the different in microprocessor execution speed. The indexer main circuit board is using a MC68008P8 microprocessor. This microprocessor is too slow if compared with the 486DX33 microprocessor. As a result, the communication between the 486DX33 PC and the indexer is not smooth. The solution was to use a program to slow down the execution speed of the PC. Hence, the PC can be used for other means beside controlling the manipulator.

A 486DX33 PC was still not quite suitable for the project because of the limitation in speed, memory and hard disk capacity. It is much better to use a Pentium II 300MHz PC to control the precision manipulator since it has a better microprocessor and higher speed. Windows based software like Microsoft Office 97, AutoCAD R13, Mechanical Desktop and Borland C++ 4.5 Programming Language can also be used in the same computer because the PC has 128 Mbytes of RAM (random access memory) and about 5 Gigabytes of hard disk space.

However, the PC's higher execution speed cannot be slowed down using a program. As a result, the routines for sending commands to the indexer and receiving responses from the indexer must be changed to solve this problem. The initial routines had a particular number of loops for communicating with the indexer. The revised routines will continue to wait until the indexer gives a response. Hence, the latest PC was successfully integrated with a PC-23 indexer to control the precision manipulator.

3.3.4 Motion Control of PC-23 Indexer

All applications of an indexer axis are either movement of a motor to a precise position (number of motor steps) or movement of the motor at a prescribed velocity (steps per second). Output control for both position and velocity can be achieved with a high degree of precision without any additional external feedback. There are approximately 106 commands for specifying different conditions and operating modes within the motion control program. Better motion control and responses from the indexer lies in the selection of suitable commands for any particular set of motion sequence.

The standard motor resolution setting for all the axes on the PC-23 indexer is 25000 steps per revolution although it supports motor or drive resolutions for up to 50000 steps per revolution. The KS-drives for motors KS 210 and KS 220 are configured in the range of 1000 to 16384 steps per revolution. For accurate speed control, each indexer axis needs to know the resolution of its controlled motor and the settings of the motor resolution on the KS-drives and the PC-23 indexer must match. In this project, the KS-drives and the indexer motor resolution settings are kept at 5000 steps per revolution.

The MRn commands of the indexer can be used to set the motor resolution. n is an integer. The MRn commands also controls step pulse width and velocity range. Table 3.3(e) shows the MRn commands with their corresponding settings for motor resolution (in steps per revolution), velocity range and pulse width.

The PC-23 indexer has two principal modes of operation, namely the preset (normal) and continuous modes, for controlling the position and speed respectively. Alternating mode is the third mode that is a special case of the preset mode. The indexer commands of MN, MC and MA can be used to set the mode to normal, continuous and alternating respectively.

Table 3.3(e) MRn Commands with Corresponding Settings

PC-23 Command	Motor Resolution	Velocity Max. (RPS)	Pulse Width (μsec)
MR0	200	160	15
MR1	400	80	15
MR2	800	78	7.5
MR3	1000	100	4
MR4	1600	60	4
MR5	3200	78	2
MR6	5000	100	1
MR7	6400	78	1
MR8	10000	50	1
MR9	21600	23	1
MR10	25000	20	1
MR11	25400	19.5	1
MR12	36000	13.8	1
MR13	50000	10	1
MR15	4096	122	1
MR16	12800	39	1
MR17	25600	19.5	1
MR18	12500	40	1
MR19	16384	30	1
MR20	20000	25	1
MR21	25000	80	0.25
MR45	2000	50	4
MR46	4000	125	1

The continuous mode only requires acceleration and velocity values for moving the motor. Once accepted the G (go) command, the motor will rotate at a constant velocity until a new velocity (and a new acceleration if desired) are commanded. Issuing the commands like S (stop) can stop the motion. In alternating mode, the motor shaft will rotate to the commanded position corresponding to the value set by the D (motor steps) command upon receiving the G command. After reaching the destination, it will retrace its path back to the start position. The shaft will continue to rotate back and forth until a stop command like S is received. Then, the motor will complete the cycle and stop at the start position. Another G command will repeat the same motion pattern.

The normal mode is the default mode. It is also the author-selected operating mode for the indexer. In this mode, the indexer will drive the motor to a desired position at a specified velocity. It can be divided into the normal incremental mode, which is a default mode and the normal absolute mode. MPI command can be used to set the positioning mode to incremental. In this mode, all move distances are referenced to the starting position of each move. On the other hand, MPA command can be used to set the positioning mode to absolute. In this mode, all distances moved are referenced to the absolute zero position (home position). The normal incremental mode is selected due to the nature of the motion control program.

An example of constructing a motion command for linear movement is described below. The desired motion conditions are:

(1) Mode of motion = Normal Absolute

(2) Indexer and drive resolution = 5000 steps per revolution

(3) Motion direction = counter clockwise

(4) Motion distance = 10 mm

(5) Velocity = 1 revolution per second

(6) Acceleration = 0.2 revolution per second squared

(7) Pitch of lead screw = 2 mm

The motor step calculation is shown as below.

(10 mm x 5000 steps per revolution) / 2 mm = 25000 motor steps

The motion command is xMN xMPA xA0.2 xV1 xD-25000 xG

Note: x can be 1 or 2 or 3 for specifying the desired motion axis. If x is not specified, its default motion axis is 1. A "+" or "-" sign may precede the specified distance. "+" = clockwise motion, "-" = counter clockwise motion, referenced to the motor mounting face. If no sign precede the specified steps, its default direction is clockwise. The maximum number of character in a command is 1000. All the PC-23 indexer motion commands are listed in Appendix A.

Synchronised motion means all the motion axes move and stop at the same time. The motion of the precision robotic manipulator can be synchronised in two ways. One way is to use pause command, PS and continue command, C. As long as PS command is located before G command in a sequence of motion commands, then, the indexer will pause the execution of the motion. The indexer will initiate the motion once the C command is received.

The PC-23 indexer's command processor is constantly switching sequentially from one axis to another for handling command processing. The time sharing process switches every two milliseconds. Time difference between different motion axes will affect the accuracy and smoothness of a complex surface since the surface is define by a large number of motion commands. Thus, the author did not choose that method.

The method of using I command and G123 command is the author-preferred method since the time difference between each motion axis is only 150 microseconds (maximum). The I command will enable the function of pre-calculating of motion data by the indexer. The pre-calculated move data will be sent over to each motor axis buffer. The buffer can accommodate up to one thousand characters' command at one time. Then, sending the G123 command to the indexer will let each motor axis to start moving within 150 microseconds of one another.

3.3.5 KS-Drive

The second part of the interface system is the KS-drive [94]. It is a complete brushless servo positioning system. The system consists of a brushless servomotor, a brushless resolver feedback and a microprocessor based closed loop drive amplifier. The KS-drive accepts digital step and direction inputs from the PS-23 indexer for controlling the position and velocity. The onboard microprocessor monitors both the pulse inputs from the indexer and the resolver feedback from the brushless servomotor. Then, it will determine the proper current levels to apply to the motor. Table 3.3(f) shows the physical, environmental and electrical specifications of the KS-drive whereas Plate 3.3(b) illustrated the drive pictorially.

Table 3.3(f) KS-Drive's Physical, Environmental and Electrical Specifications

Parameter	Value
Physical	
Height:	≈ 241 mm
Width:	≈ 125 mm
Depth:	≈ 171 mm
Weight:	≈ 14.6 kg
Environmental	
Operating:	0 to 50°C (with adequate air flow)
Humidity:	0 to 95% (non-condensing)
Storage:	- 40 to 85°C
Electrical	
Input Power	
Voltage:	100 – 130 VAC, single phase
Frequency:	47 – 66 Hz
Current:	6.3 amps maximum continuous (RMS)
Output Power (to motor)	
Voltage:	170 VDC peak
Frequency:	20 kHz PWM
Current:	5.0 amps continuous per phase 8.0 amps per phase peak (at 50°C)

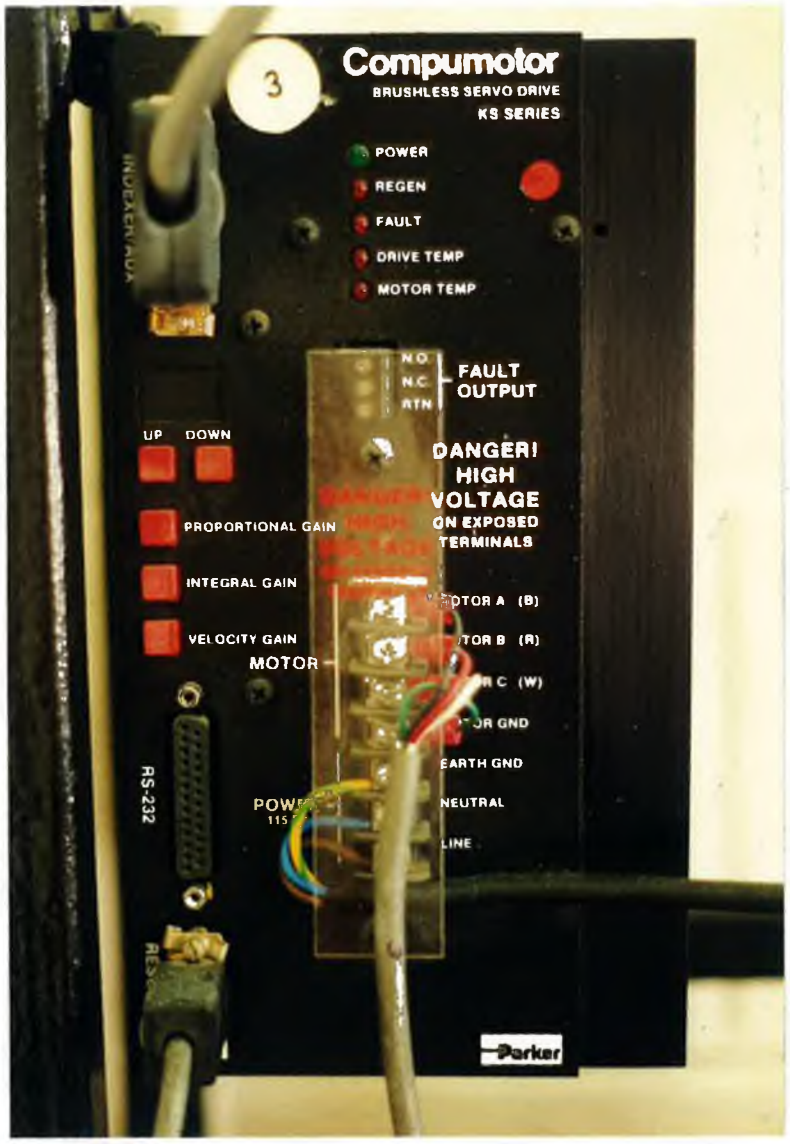


Plate 3.3(b) KS-Drive

Closed loop performance is simplified by the control of a microprocessor and a sophisticated servo algorithm. All servo performance parameters are stored in non-volatile EEPROM memory. As a result, a conventional system analogue potentiometer is not needed for adjustment purposes. The power amplifier section of the drive utilises a MOSFET 20 kHz pulse width modulation (PWM) current control. This design will improve the low speed smoothness and resulting in quiet operation. Other features of the KS-drive are listed as below:

(1) Up to 3000 revolution per minute in speed.

(2) RS-232 serial communication interface with the personal computer.

(3) User programmable resolution in steps per revolution via RS-232.

(4) Simple push button adjustment of servo compensation (Proportional, Integral, Velocity and Derivative gain).

(5) Adjustment of servo compensation can also be done through RS-232.

(6) High noise immunity due to optical isolation and brushless resolver technology.

(7) LED fault indicators (Power, Regen, Fault, Drive Temp, and Motor Temp).

3.3.6 Visual Indicators of KS-Drive

As shown in Plate 3.3(b), there are five LED indicators at the front panel of the KS-drive. The indicators are POWER, REGEN, FAULT, DRIVE TEMP and MOTOR TEMP. Their functions are described as below:

(1) POWER. This is a bicolour LED that will be green under normal operating conditions. If the microprocessor fails, this LED will be red. Turning the power of the drive on and off will clear the condition only if the problem is temporary. If the LED is off, then, it indicates a loss of the low voltage power supply.

(2) REGEN. This is a red LED that is normally off. It will be on when the motor is generating power that is being dissipated into the power dump resistor. If the LED is on for more than 5 percent of the time or 10 seconds in duration, then, the motor

sizing and duty cycle calculations should be considered. Adding external resistors to dissipate the generated heat, adding cooling facilities to the internal resistors, changing motor sizes, changing gear ratio or slowing down the duty cycle can rectify the problem.

(3) DRIVE TEMP. This red LED is normally off. When the drive is having over temperature condition, it will be on and causing the FAULT LED to light up too. Power down the drive for 30 minutes will rectify the problem. If the problem is recurring, then, using a fan kit is necessary.

(4) MOTOR TEMP. This LED is normally off. When lit, it will be red and indicating an over temperature condition in the motor. It is derived by the microprocessor based on the average current being sent to the motor. Lower the peak current setting can rectify the fault.

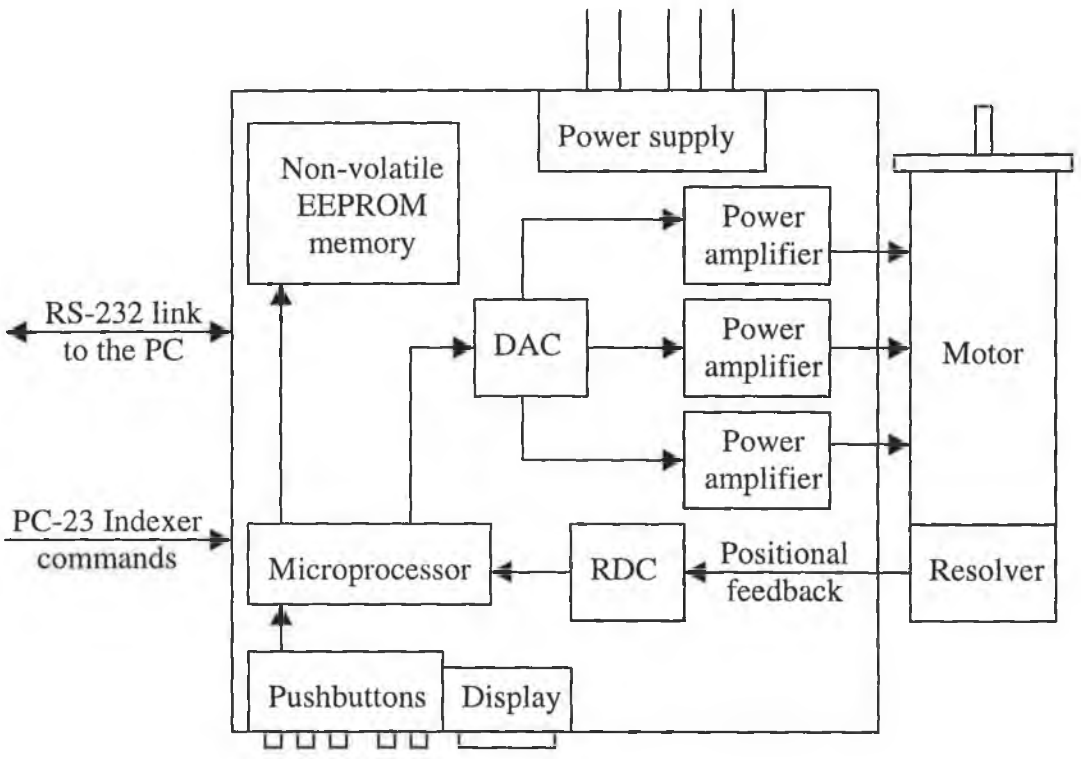
(5) FAULT. Under normal operating condition, this LED is off. The LED will be red if there is a microprocessor detectable error condition like under voltage, short circuit, over current, over temperature and so on. When the drive is having fault, a numerical error code will be display at the code display window above the push buttons. Table 3.3(f) shows the available KS-drive error codes and its conditions.

Table 3.3(g) KS-Drive's Error Codes with Conditions

Code	Condition
11	Over temperature
19	Short on motor circuit.
20	Following error exceeded.
21	Outside allowable deadband.
22	Maximum average current exceeded.
30	EEPROM checksum error.
40	Both limits engaged and indexer commands attempting move.
60	RS-232 command shutdown.
61	Incoming indexer pulses.

3.3.7 Servo System of KS-Drive

The KS-drive can be divided into the digital controller board and the analogue amplifier board. The controller board sends two digitised waveforms from its DAC (digital to analogue converter) to the analogue amplifier board. These waveforms represent two commanded motor phase currents. The analogue amplifier board generates its own third phase command and measures the actual motor current to determine the correct pulse width of voltage to apply to the motor windings. The KS-drive servo system is shown in Figure 3.3(b).



DAC = digital to analogue converter
RDC = resolver to digital converter

Figure 3.3(b) KS-Drive Servo System

The controller will command a “desired current” to the amplifier board. Then, the amplifier boards will attempt to generate that “desired current” in the motor windings. The resolver that is attached to the motor will sense the position of the motor shaft and send the information back to the controller. With the positional information, the controller will generate the “desired current” command to the amplifiers.

The generation of the current command to the amplifier by the controller is based on several quantities. They are:

(1) Position of the motor shaft from the resolver.

(2) Desired position of the PC-23 indexer command.

(3) Previous current commands of the amplifier.

An indexer will generate a stream of pulses that the controller collects with an up/down (i.e. clockwise/counter clockwise) counter. The resultant pulse count, at any given instant of time, is the desired position. The controller will subtract the motor's actual position from this desired position to determine the positional error. The positional error is the difference between where we want the motor to be and where it actually is. This positional error is put into a recursive equation, along with previous positional errors and previous commands to the amplifier, to generate the current command for the amplifier.

The recursive equation is a mathematical function that is evaluated at periodic time intervals. The recursive equation of the KS-drive is an approximation of an analogue, continuous-time PID network that is used quite often in stabilising conventional servo systems. The drive’s recursive equation is the discrete-time equivalent to a continuous-time PID network.

It is called a discrete-time PID network because it operates on sampled data and not on continuous data. The sampling rate of the drive controller is the rate at which the recursive equation is evaluated and the rate at which the current command to the amplifier is changed. The sample-rate of the drive controller is 512 microseconds. Such a fast rate can produce excellent dynamic response.

The digital controller board handles all the positioning compensation like proportional, integral, derivative and velocity gains. The effects of the PID and V (proportional, integral, derivative and velocity gain) to the system's response are:

(1) *Proportional Gain.* It will affect the system stiffness and accuracy. The influence of the feedback signal becomes greater if the gain is adjusted higher. If the gain is too high, the system will oscillate. That is because very small resolver changes are amplified into very large error signals. The mechanical inertia of the motor and load will not allow the system to follow the electronic commands fast enough. The system's lag time will finally reach a point where the feedback and the command signals are in phase, then, oscillation occurs.

(2) *Integral Gain.* It allows the system to compensate for positional errors in static position. It also works to reduce the velocity ripple. It does that by slowing down the electronic response time so that it can be more closely resembles the response of the mechanical components of the loop.

(3) *Derivative Gain.* It will add damping effects to the system. Increasing the gain will reduce the ringing if the system is oscillating at the end of a move or around a change in velocity. The derivative gain will add phase lead to compensate for the system natural phase lag.

(4) *Velocity Gain.* This gain is used to affect the overall responsiveness of the system. If the system is too sluggish, then the velocity gain can be increased. If the system is overshooting badly or there is excessive ringing that the derivative term is not able to adequately compensate, then, reducing the velocity gain will work.

The most important aspect of a servo system is setting the controller's "gains". The "gains" of the controller are the constant coefficients of the recursive equation. The form of the recursive equation will determine how many of these "gains" should be adjusted in order to stabilise the system. The methods of adjusting the KS-drive's servo compensation network are

(1) The five pushbuttons on the KS-drive front panel.

(2) The RS-232 serial communication port.

3.3.8 Pushbutton Tuning of KS-Drive

The KS-drive has five pushbuttons on the front panel that provide a simple pushbutton method of fine tuning the systems performance to a specific attached load. Before trying to set the gains of the controller, it is important to observe the response of the system to commands from the indexer and the stiffness of the system at rest.

With the motor at rest, if attempt is made to turn the shaft, it should not be easily turned from its rest position. If it feels soft, the system gains may need to be increased since a soft system will not respond very quickly to the motion commands. If it feels stiff, the system should be checked so that it is not vibrating. The gain may be too high if there is vibration. Vibration will cause the drive to provide excess current and can shorten the life of mechanical components. In extreme situations, the vibration will grow in amplitude producing ever more violent motion until the drive faults or something breaks. As a result, tuning the KS-drive should be done with some caution.

On the front panel of the KS-drive, there are five red colour pushbuttons and a two-digit LED display. The buttons are labeled UP, DOWN, PROPORTIONAL GAIN, INTEGRAL GAIN and VELOCITY GAIN. Holding down one of the gain buttons will cause the display to light up and indicate the present value for the selected term. Holding down a specific term button and pressing the UP button once will increase the term value by one count. On the other hand, holding down a term button and pressing the DOWN button once will decrease the term value by one count. As long as the term button is held down, repeated pushes on the UP or DOWN button will cause the term to continue to increase or decrease by one unit for each push of the button.

After the completion of the pushbutton tuning procedure, it will be necessary to save the selected term values into the non-volatile EEPROM memory. Pressing all three term buttons and releasing them at the same time will save the setting memory. The pushbutton tuning procedures and its conditions are shown in Table 3.3(g).

Table 3.3(h) Pushbutton Tuning Procedures and Conditions

UP	DOWN	P	I	V	Condition
0	0	0	0	0	Display off/unless error LED is on.
0	0	+	0	0	Display current PROPORTIONAL value.
+	0	+	0	0	Increase PROPORTIONAL value.
0	+	+	0	0	Decrease PROPORTIONAL value.
0	0	0	+	0	Display current INTEGRAL value.
+	0	0	+	0	Increase INTEGRAL value.
0	+	0	+	0	Decrease INTEGRAL value.
0	0	0	0	+	Display current VELOCITY value.
+	0	0	0	+	Increase VELOCITY value.
0	+	0	0	+	Decrease VELOCITY value.
0	0	0	+	+	Display current DIFFERENTIAL value.
+	0	0	+	+	Increase DIFFERENTIAL value.
0	+	0	+	+	Decrease DIFFERENTIAL value.
0	0	+	0	+	Display current device address.
+	0	+	0	+	Increase device address (maximum is 15).
0	+	+	0	+	Decrease device address (minimum is 1).
0	0	+	+	0	Return to factory default setting.
0	0	+	+	+	Save the tuning values.
+	+	0	0	0	Reset.

Note: P = Proportional Gain, I = Integral Gain, V = Velocity Gain, + = Pushed, 0 = Not pushed. Other combinations are ignored.

3.3.9 RS-232 Interface of KS-Drive

The tuning process of the front panel pushbuttons can be duplicated through the RS-232 serial communication port with an interface program. The KS-drive's RS-232 connector is a standard 25-pin "D" connector. It has a three-wire implementation of this interface and provides Receive Data (pin 2), Transmit Data (pin 3) and Ground (pin 7). No handshaking is required for the interface. The interface program enables the communication between a personal computer and 1 or multiple KS-drives. The command and response are strings of characters. It is assumed that:

(1) The personal computer's serial card is set as either COM1 (3F8 Hex) or COM2 (2F8 Hex).

(2) The communication protocol is configured as:

(a) Baud rate = 9600

(b) Data bits = 8

(c) Parity = None

(d) Stop bits = 1

(3) The KS-drive/s is connected to the COM1 or COM2 of the personal computer.

(4) Each KS-drive's device address is set.

Note: In Windows95, the COM1 or/and COM2 communication protocol can be set via Control Panel's Modem setting.

The connection between the personal computer and the KS-drive is important in order to make sure that the communication is smooth. The RS-232 connector pinouts for most computers are:

25 Pin "D" Connector -------- 2 (Transmit Data), 3 (Receive Data), 7 (Ground)

9 Pin "D" Connector ---------- 2 (Transmit Data), 3 (Receive Data), 5 (Ground)

As a result, the computer serial port pin 2 must be connected to the pin 3 of the drive serial port. The drive serial port pin 2 must be linked to the pin 3 of the computer serial port. The ground of the two serial ports must be connected. As more than one KS-drive is used, it is necessary to construct a daisy chain cable to connect all the KS-drives to the personal computer. Figure 3.3(c) shows the daisy chain wiring.

The interface program is a DOS platform software. Upon opening the program, press Alt-T to access the Terminal menu. The Settings command under the Terminal menu allows us to verify and set the communication protocol (baud rate, data bit, stop bit, and parity), the selected serial port (COM1 or COM2) and others. Once the settings are correct, then select Connect command under the Terminal menu. The Connect command is for checking the RS-232 connection. If the connection is valid, the Terminal window will be brought up. If an error is detected, a message such as "Device not ready or echo off" will be displayed.

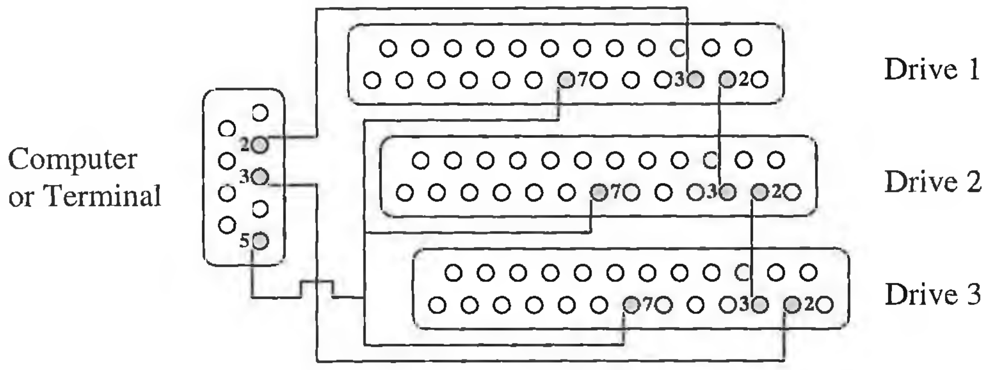


Figure 3.3(c) Daisy Chain Wiring

After the Terminal window is opened, typing E command will enable the RS-232 link and disable the pushbuttons functions (except the reset function). Typing F command will return tuning control to the pushbuttons. Any command that will cause the drive to transmit information to the RS-232 port must be prefixed with a device address. This is to prevent several drives from transmitting at the same time in daisy chain wiring.

Responses and reports from the drive will have an asterisk (*) as a leading character to prevent the response from being interpreted as a command by other devices on the communication link. Invalid commands will be ignored by the drives. Upper or lower case command characters are accepted by the drive but the Echoed characters from the drive will always be upper case. After the setting has been changed, typing SV command will save the new value into the non-volatile EEPROM memory.

An example of checking and changing the motor resolution is shown as below.

2E (enable the RS-232 link between the PC and the KS-drive 2)

2CMR (checking the current motor resolution of the KS-drive 2)

*MOTOR_RESOLUTION=16384_STEPS/REV (response from the KS-drive 2)

2CMR5000 (changing the KS-drive 2's motor resolution to 5000 steps per revolution)

2SV (saving the new motor resolution value into KS-drive 2's non-volatile EEPROM)

2CMR (checking the latest motor resolution of KS-drive 2)

*MOTOR_RESOLUTION=5000_STEPS/REV (response from the KS-drive 2)

1F (return to the pushbutton tuning control)

The motor resolution of KS-drive 2 has been changed from 16384 steps per revolution to 5000 steps per revolution by following the above steps. The motor resolution for all the involved motors is set at 5000 steps per revolution. There are about 45 commands available for a KS-drive. All the KS-drive commands are listed in Appendix B. The commands are categorised into:

(1) *General Commands.* E, F, SV and so on.

(2) *Configuration Commands.* CMR (configure motor resolution), FMCA (find motor commutation angle) and others.

(3) *Tuning Commands.* CVG, CIG (tuning integral gain), CPG (tuning proportional gain) and so on.

(4) *Display/Report Commands.* DCA (periodically displays/reports current in amperes), DCP (periodically displays/reports peak current) and others.

3.3.10 A. C. Brushless Servomotors

Four a. c. brushless servomotors drive the four degrees of freedom precision manipulator. Three of the motors are KS-220 and another one is KS-210. Two KS-220 motors are used to drive the linear motion along x and y-axis. Another KS-220 motor is used in driving the rotary motion around y-axis. The KS-210 motor is used for driving the rotary motion around x-axis. The specifications of the motors are shown in Table 3.3(i).

Table 3.3(i) A. C. Brushless Servomotors' Specifications

Description	KS-210 Motor	KS-220 Motor
Static torque (continuous)	0.23 Nm	0.43 Nm
Static torque (peak)	0.70 Nm	1.29 Nm
Top speed	50 revolutions per second	50 revolutions per second
Rotor inertia	$7.4\text{x}10^{-6}$ kgm^2	$13.2\text{x}10^{-6}$ kgm^2
Weight	1 kg	1.32 kg
Motor Resolution	1000 to 16384 steps per revolution (programmable)	
Repeatability	+/- 0.033 degrees (unloaded at 20°C)	
Accuracy	+/- 0.35 degrees (unloaded)	
Relative accuracy	+/- 0.35 degrees (any load)	
Operating temperature	130°C (maximum)	
Storage temperature	-40 to 85°C	
Humidity	0 to 95% (non-condensing)	

An a. c. brushless servomotor will rotate when the rotor magnetic field tries to follow the stator turning magnetic field created by the three phase a. c. current. By changing the three phase current frequency, the motor will achieve different velocities. Step pulses applied first slowly, and then more quickly have the effect of accelerating the motor. The advantages of a brushless motor [95] are:

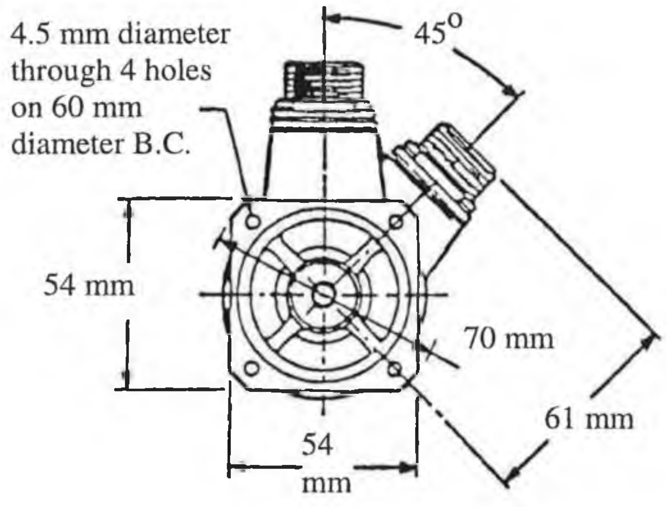
(1) Reduced maintenace.

(2) Increased torque/volume ratio.

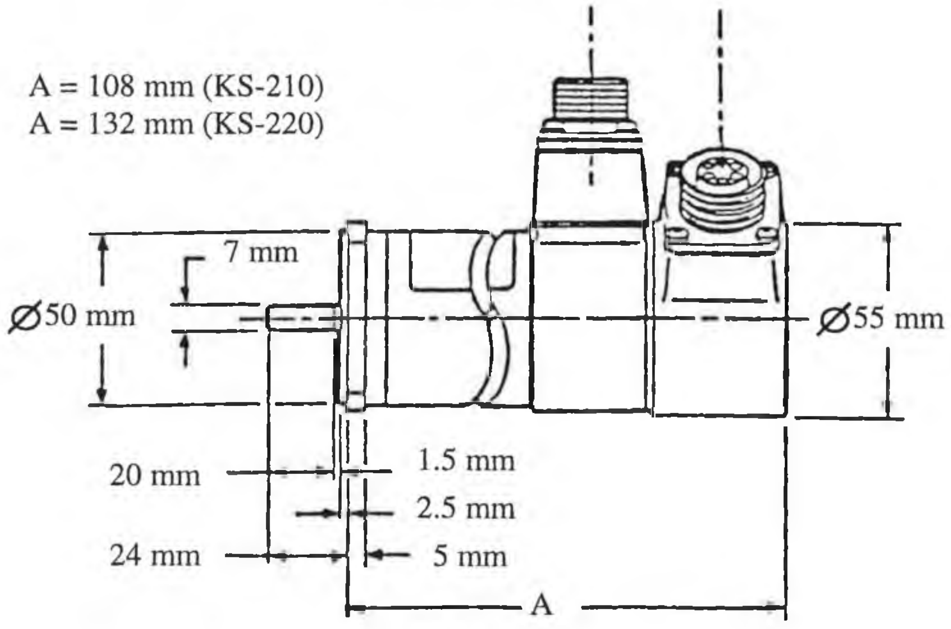
(3) Increased torque at high speed.

(4) Simplified in protection compare with more conventional motors.

The front and side views of the KS-210 and KS-220 a. c. brushless servomotors are shown in Figure 3.3(d) with dimensions.



Front View



Side View

Figure 3.3(d) A. C. Brushless Servomotors

3.4 SUBTRACTIVE PROTOTYPING EQUIPMENT

The subtractive prototyping process is a ball nosed end milling cutter. A ball nosed cutter has cutting edges at the end and around the cutter. As a result, single point cutting can be accomplished by using the cutting edge at the end of the cutter. The cutting edges at the periphery of the cutter enable multiple cutting operations to happen at different interval of time.

The size of the milling chips is relatively small compared to other machining processes. Hence, the produced surfaces are smoother. Ball nosed end milling can produce virtually any kind of surface compared to other kinds of milling processes. Ball nosed end milling has the advantage of making holes compared to the conventional end milling. The equipment is shown in Plate 3.4(a).

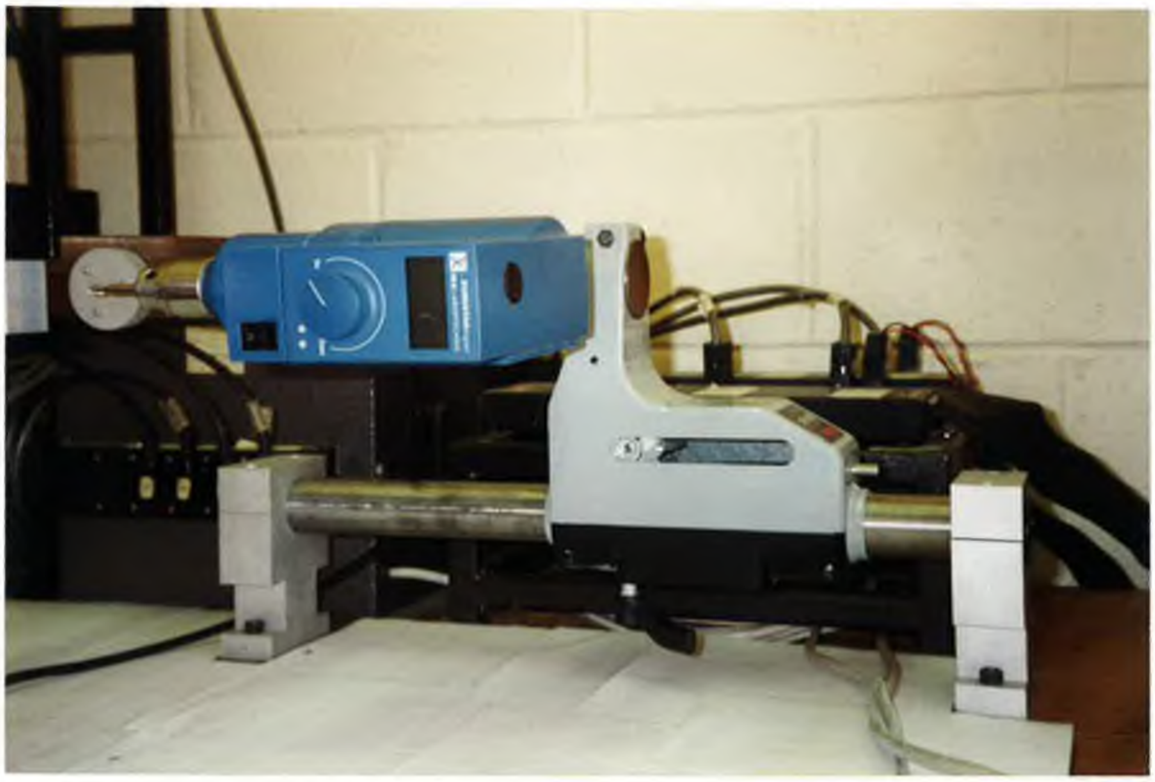


Plate 3.4(a) Subtractive Prototyping Equipment

As shown in Plate 3.4(a), the subtractive prototyping equipment consists of the following components:

(1) Ball nosed cutter.

(2) Digital drive.

(3) Drive holder and support.

Ball nosed slot drill was selected due to its versatility in producing various kinds of surfaces and features. The available cutters are

(1) 3 mm diameter high strength steel screwed shank standard ball nosed slot drill (Sherwood, CTL 061 5952C).

(2) 6 mm diameter 8 percent Co screwed shank standard ball nosed slot drill (Sherwood, CTL 061 5955F).

The ball nosed cutter was mounted onto the clamping chuck of a EUROSTAR digital d. c. motor [96]. The maximum tool diameter allowed is 10 mm. The ball bearing equipped d. c. motor has a quiet synchronous belt drive. The motor is controlled via a computer-controlled speed regulator using pulse width modulated voltage (PWM). The whole drive unit is maintenance free and suitable for continuous operation. The motor current is electronically limited and has anti-stall as well as anti-overload system. The front panel of the drive is shown in Figure 3.4(a).

In normal operation, green signal light at the front panel will be on. If a fault occurs, a safety circuit immediately switches off the motor permanently through a relay and indicated with a yellow signal light at the front panel. At the same time, a fault code will be shown in the LCD display at the front panel. Two of the error codes are:

(1) *ER3 – Internal temperature too high.* It can only occur when the permitted environmental temperature is exceeded the limit.

(2) *ER4 – Speed fault.* Indicating the output shaft is locked or the speed was higher than permitted. If there are jerky loads that exceed three times the nominal torque, the machine will switch off as a safety precaution.

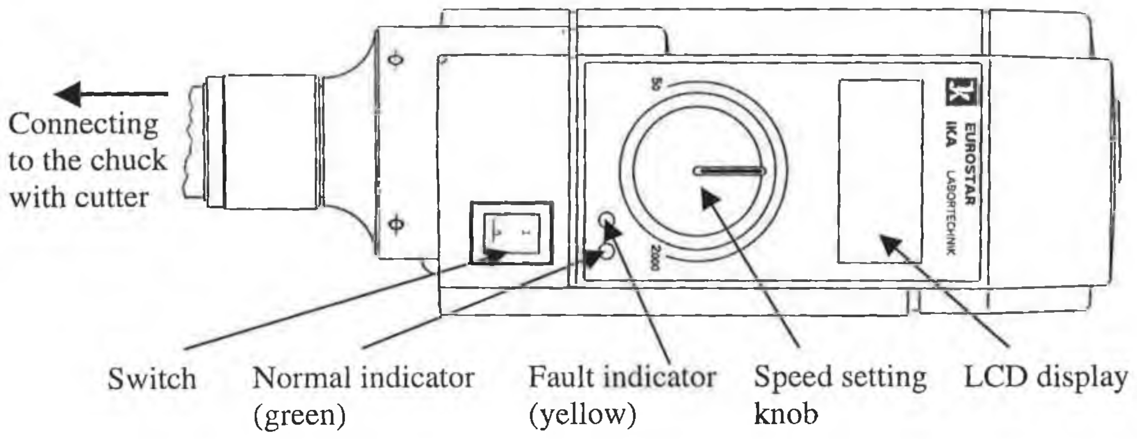


Figure 3.4(a) Milling Drive Front Panel

The nominal speed value is constantly compared with the actual speed value of the output shaft and the variations will be corrected. This will ensure a constant speed during the milling process. The speed is set with the front knob. The actual value is indicated directly in rpm (1/min) on the LCD display. The nominal value set corresponds to the actual value.

In overload operation, the drive can deliver doubled output for a short time to even out load peaks which could, for instance, occur if the milling material is not homogenous throughout the whole cross section. The possible speed is continually adapted to operating conditions to ensure that the speed is as close as possible to the nominal speed set. The technical data of the drive is shown in Table 3.4(a).

Table 3.4(a) Milling Drive Technical Data

Parameter	Value
Speed range	50 – 2000 rpm (revolution per minute)
Permitted on-time	100%
Speed indicator	Liquid crystal display (LCD)
Frequency	50/60 Hz
Input power	75 W
Output power	53 W
Overall efficiency	71%
Ambient temperature	5 to 40°C
Ambient humidity	80%

The entire drive holder unit and half of the support component are from the Bosch BS45 holder and support equipment. The drive holder is 210 mm in length and sits on a 535 mm long hollow metal tube. The tube is in turn supported by two aluminium blocks. The aluminium blocks and the whole unit of precision robotice manipulator are mount on a 70 mm by 140 mm by 10 mm aluminium base. Part of the supporting unit is shown in Figure 3.4(b) below.

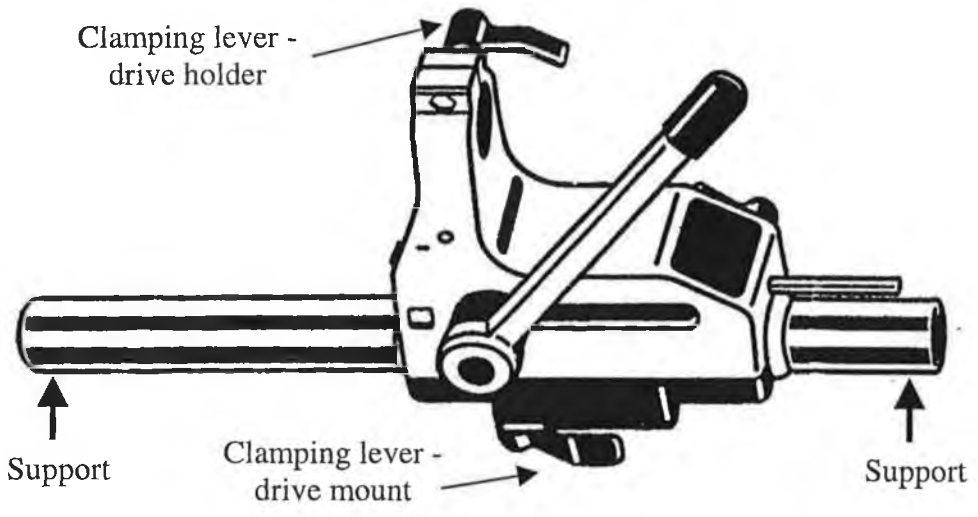


Figure 3.4(b) Drive Holder and Support

The milling material used in the project is a kind of blue colour extruded polystyrene [97]. The material is found to be a good choice for milling because minute chips can be removed during the prototyping process and smooth surface can be created. Its properties are not the same as the normal white colour polystyrene used to protect electrical appliance in the packaging industry. The material's minimum density is 32 kg/m^3. Its thermal conductivity is 0.028 W/mK (measured at 10^{o}C). The compressive strength is 300 kN/m^2.

CHAPTER 4: CAD/CAM

4.1 INTRODUCTION

Computer-aided design and manufacturing (CAD/CAM) play an important role in this project. The CAD software used in the project was AutoSurf [98], which is part of the Mechanical Desktop (product of Autodesk, Inc.) package. The CAM programs were built by the author using ANSI (American National Standards Institute) C programming language [99].

In the subtractive prototyping approach, the CAD software was used to create surface models. The models were sectioned and cut into multiple cross sectional layers. Then, the section cut models were converted from graphic files into non-graphic files (neutral format files) for further processing by the CAM programs.

In the additive prototyping approach, the CAD software was used to create solid models with internal cavities (parent model). Multiple smaller diameter solid models were then derived from the parent model. The various diameter solid models were converted into surface models. The surface models were then section cut and later changed into neutral format files. The various diameter solid models were also used in the graphic simulation process. AutoLISP [100, 101], the AutoSurf programming language was used in developing the graphic simulation program.

The CAM programs of the subtractive and additive processes are specially used for extracting surface co-ordinates from the neutral format files, converting the data into different co-ordinate system, sorting, creating motion parameters, communicating with PC-23 indexer and controlling the precision robotic manipulator.

4.2 CAD TOOL

AutoSurf is a personal computer based, two and three-dimensional mechanical design and drafting software. Geometric shapes and figures can be created and modified for engineering purposes. A reduced instruction set processor (RISC), with a limited number of instructions is built into the processor to reduce the response time for running some applications on the software development system [6].

Crosshairs and a computer mouse are used to locate geometric shapes within the work area. An X-Y construction plane is used for the two-dimensional mode that uses a three-point origin placed by the user, known as the user co-ordinate system (UCS). In default setting, the Z-axis is perpendicular to the personal computer screen and pointing directly to the user.

AutoSurf has an open architecture for easy customisation of menus. The screen menu is the main menu, which includes the drawing editor, configuration, plot, file utility, and operating parameters menus. A dialogue box appears when selected item is chosen from the pull-down menus to assist the user. Besides of using the pull-down menus, the user can type in the commands into the command prompt to call up the functions.

The software commands are path dependent. For example, the 'undo' command will remove the screen image and any previous drawing layers up to the earlier drawing level. AutoLISP is the AutoSurf programming language that enhances the drawing and editing commands. It is an interpretive system, with instructions being read, interpreted, validated, and then executed in sequence. It can also be used to simulate the material processing process.

<u>4.3 CAD FOR SUBTRACTIVE PROTOTYPING</u>

It is important to know the physical limitation of the manipulator and the capability of the ball nosed end milling process before attempting to create the surface models and the subsequent models modification. The criteria that must be followed in designing the surface models are:

(1) *Shape*. Manipulation along the x-axis, y-axis and around the x-axis are used in this project. Hence, the shapes of all the surface models are in cylindrical forms.

(2) *Length*. The three-pin grippers of the manipulator can hold a polystyrene cylinder block with the size ranging from 120 mm to 125 mm in length and 40 mm to 150 mm in diameter. But, due to the pins of the grippers, the length of the surface model should be less than 110 mm.

(3) *Milling Depth*. Maximum milling depth is determined by the length of the cutting edges. The length of the cutting edges along the cutter axis is about 15 mm. As a result, the distance between the surface model's highest and lowest co-ordinates (from the axis of rotation) should be less than 15 mm.

(4) *Reference Point*. The model is a cylindrical surface form with a rotating axis at the (100, 100, 0). The x = 100, y = 100 and z = 0 is an important reference point for the CAM programs in the later stage. Motion parameters are produced based on that reference point.

4.3.1 Surface Modelling and Manipulation

Some of the important modelling configurations of AutoSurf need to be checked and set before creating the surface models. The configurations are:

(1) *Units*. It is for selecting co-ordinates and angle display formats and precision. The command can be found under the pull down menu of Data or by typing 'units' at the command prompt. The selected type of unit is Decimal and the precision is up to four decimal point. The selected type of angle is Decimal Degrees and up to 0^o precision. Direction of rotation can be set under this command. East has been set as 0.0^o and the direction of rotation is counter clockwise.

(2) *Drawing Limits*. It is for setting and controlling the two and three-dimensional drawing boundaries. The command is under the Data pull down menu too and can be called up by typing 'limits' at the command prompt. Currently, the lower left corner of the drawing limit is set at (0.0000, 0.0000) and upper right corner of the limit is set to (297.0000, 210.0000). A4 size's drawing limit was set.

(3) *Layers*. This command is also under the Data pull down menu and can be called up by typing 'layer' at the command prompt. Under this command, the user can set as many line types and line colours as possible so that they can be used in the subsequent modelling.

(4) *Drawing Aids*. This command is under the pull down menu of Options and can be called up by typing 'ddrmodes' at the command prompt. The command will enable the setting of Grid spacing and Snap spacing that will make the modelling easier.

(5) *Preferences*. It is for customising the AutoSurf settings. It can be called up by typing 'preferences' at the command prompt or under the Options pull down menu. It can

be used to set the types of digitizer input, font type, font size, background colour and others.

AutoSurf R3.2 is a window based CAD software. It is part of the Mechanical Desktop package from Autodesk, Inc.. If compare to AutoCAD, AutoSurf has more features and more user-friendly. Besides, only AutoSurf can be used to modify the created surface models in the later stage. As a result, AutoSurf was chosen by the author for the surface modelling process. The AutoSurf modelling system is based on NURBS (non-uniform rational B-spline) curves. Technically, there is only one type of surface in AutoSurf, a NURBS surface.

Models can be created and modified by using the commands from the toolbars like Draw, Modify and Surface Create. Draw and Modify toolbars are under the Toolbars menu that is in turn under the Tools menu. The Draw toolbar enable the creation of line, polyline, arc, circle, ellipse, polygon and point. The Modify toolbar has commands like move, copy object, offset, mirror, array, rotate, scale, trim, extend, edit polyline, chamfer, fillet, union, subtract, intersection, erase and so on.

The Surface Create toolbar is under the Mechanical Toolbars pull down menu. Mechanical Toolbars is under the Tools pull down menu. There are various commands under the Surface Create toolbar. There are four different types of surfaces in terms of the methods used to construct them:

(1) *Surface Primitives*. Created directly by the AutoSurf. Examples are cone and cylinder surfaces.

(2) *Motion-based Surfaces*. Produced by moving wires through space. Examples are revolved, extruded, tubular and swept surfaces.

(3) *Skin Surfaces*. Constructed by applying over a wireframe such as ruled surface.

(4) *Derived Surfaces*. Generated from the existing surfaces like blended surface.

Primitive surface models do not require a wireframe for their construction but are instead directly created using the user-specified values. A full or partial primitive cone surface model can be created by using the Cone Surface command at the Surface Create toolbar. Typing 'amprimsf' and then selecting Cone at the command prompt has the same effect of activating the modelling task.

A full cone surface model was built by using AutoSurf. The base centre point was (100, 100, 0). The radius of the cone base was 16 mm. The radius of the cone top was 22 mm. The height of the cone was 50 mm. The start angle was 0° (default setting). The included angle was 360° (default angle is a full circle). The primitive cone surface model is shown in Figure 4.3(a).

The surface model can be verified and viewed at different angles by using the Rotate View command at the Desktop View toolbar. The Desktop View toolbar is under the Mechanical Toolbars menu. After creating the surface model, the following step was to cut it into multiple cross sectional layers. Only the neutral format file (DXF entities file) of the section cut surface model contains the useful data for subtractive prototyping process.

The section cut surface model can be created by using the Section Cuts command at the Surface Create toolbar. The author needed to change the view of the primitive model into the Front View (from Desktop View toolbar) before activating the Section Cuts command. Typing 'amsection' at the command prompt can also call up the Section Cuts command.

Upon selecting the primitive surface model, a dialogue box will appear. In order to cut the surface model, the following information must be provided.

(1) *Section type*. It is for selecting Single, Parallel or Radial sectioning cut.

(2) *Initial plane*. It is for specifying the initial cutting plane. It can be either from the user View Direction or UCS (user co-ordinate system) plane.

(3) *Multiple cuts*. When Parallel or Radial section type is selected, the user needs to specify the Stop position of the last cut and the Step of each cut (step over).

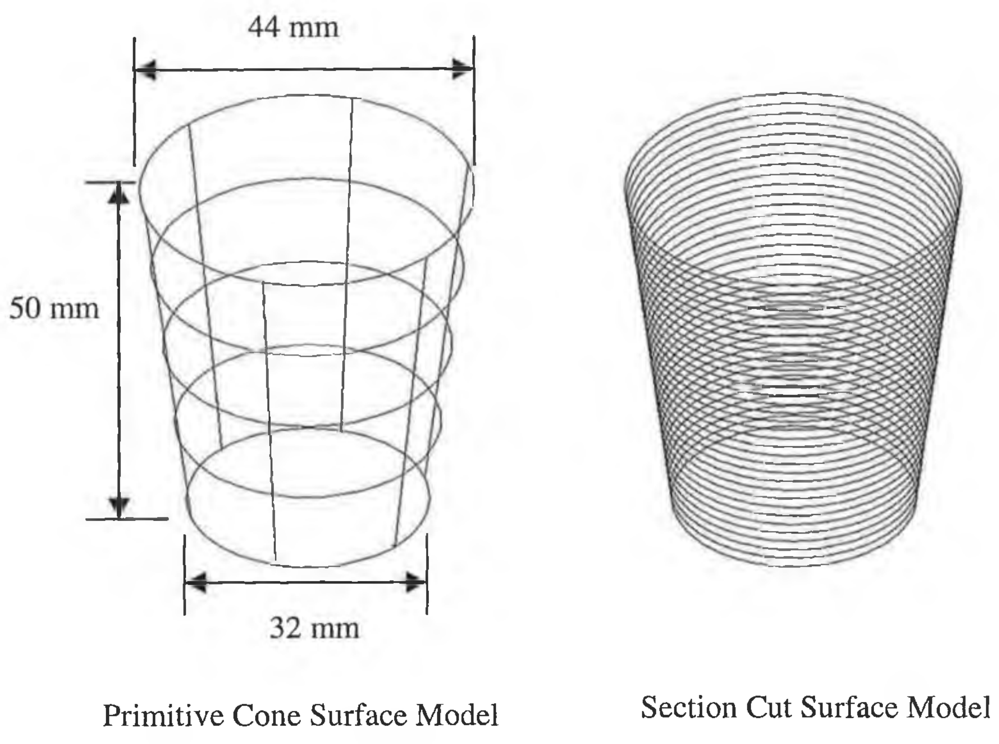
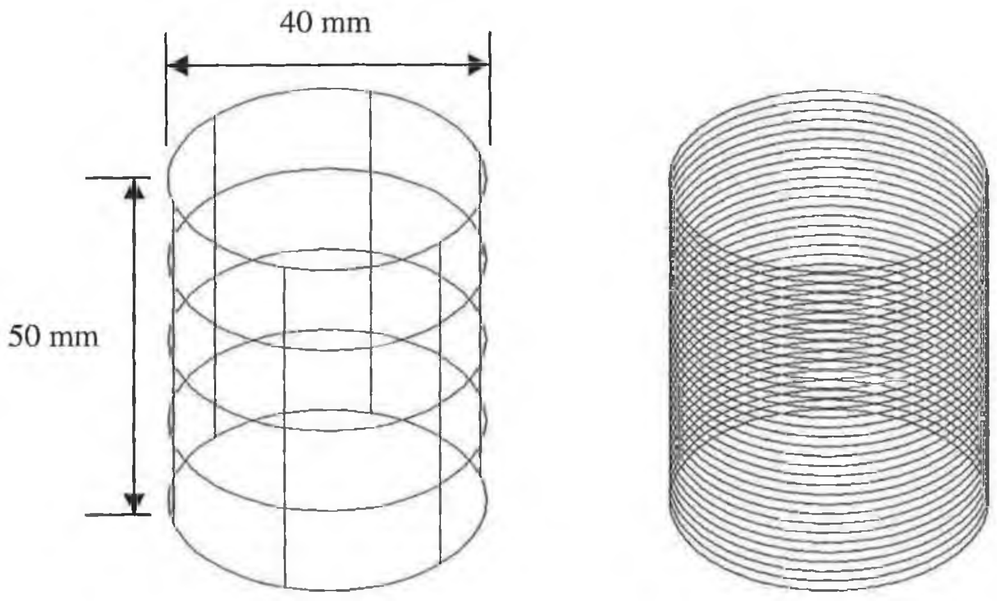


Figure 4.3 (a) Primitive Cone and Section Cut Surface Models

A primitive cone surface model was cut into multiple layers by using Section Cuts command. The section cut surface model is displayed in Figure 4.3(a) too. Parallel section type was selected for creating the model. It started to section cut the model from the UCS plane. The section cut stopped at 49.5 mm and with a step over of 1.5 mm. The total section cut layers are 34. The step over distance is one of the parameters that will affect the final product surface smoothness.

A full primitive cylinder surface model can be generated by using the Cylinder Surface command at the Surface Create toolbar. Typing 'amprimsf' and then selecting Cylinder at the command prompt can also activate the function. A primitive cylinder surface model was built by using (100, 100, 0) as based centre point, radius of 20 mm, height of 50 mm, start angle was 0^o and included angle was 360^o. A cylinder section cut model was produced by using the same method as the cone section cut model described above. Figure 4.3(b) shows both models.



Primitive Cylinder Surface Model   Section Cut Surface Model

Figure 4.3(b) Primitive Cylinder and Section Cut Surface Model

Motion-based surface models are created based on the three-dimensional motion of wires through space. A revolved surface is one of the motion-based surfaces. Rotating any number of path curves or profiles, around a selected axis creates surfaces of revolution. The path curve can be line, arc, spline or polyline. An example of creating the revolved surface model by using a polyline is presented in the following paragraphs.

Before creating the revolved surface model, a polyline has to be drawn. The Polyline command is under the Draw toolbar. Typing 'pline' at the command prompt has the same effect as calling up the function. The polyline should be drawn within the boundary of (75, 100, 0), (85, 100, 0), (85, 150, 0) and (75, 150, 0). This is to ensure that the revolved model will have a minimum diameter of not less than 30 mm and the maximum diameter of not more than 50 mm.

After the polyline was drawn, the polyline has to be smoothened. Using the Edit Polyline command on the Modify toolbar or typing 'pedit' at the command prompt will turn sharp edges of the polyline into smooth corners. There are a few options to choose from the Edit Polyline command such as Fit, Spline and others. Spline was chosen so that the polyline will become a B-spline curve.

A completely revolved surface model was built by rotating a polyline around the axis of the model. The Revolved Surface command is under the Surface Create toolbar and can also be activated by typing 'amrevolvesf' at the command prompt. The 'Start point of axis' was (100, 100, 0) and the 'End point of axis' was (100, 150, 0). The start angle was 0^{o} (default setting). The included angle was 360^{o} (default angle is a full circle). As a result, a revolved surface model that is 50 mm long was created.

For section cut purposes, the revolved surface model has to be rotated about a three-dimensional axis. The command for rotation is 3D Rotate that is under the Modify toolbar. Typing 'rotate3d' at the command prompt serve the same purpose. The rotating axis has to be specified as x. The point on x-axis that the model will use as the rotation based point was (100, 100, 0). The rotation angle was 90^{o}.

Changing the view to Front View and using the Section Cuts command can cut the revolved surface model into multiple layers. The section cut configurations were same as the primitive cone surface model cutting operation. The revolved surface with its section cut models are shown in Figure 4.3(c). The total section cut layer is 34.

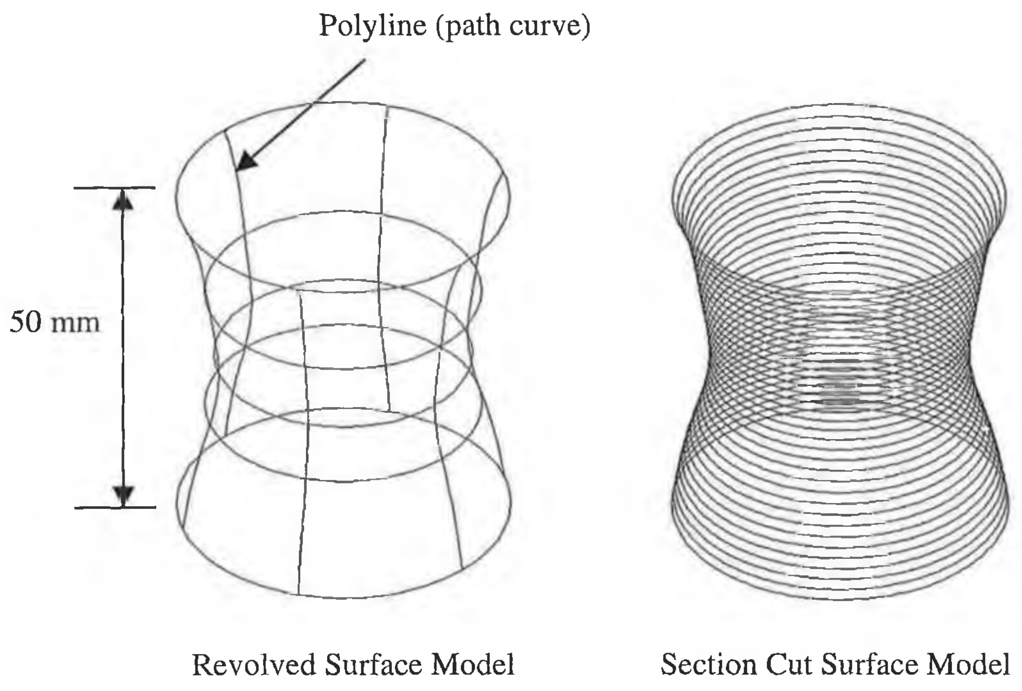


Figure 4.3(c) Revolved and Section Cut Surface Models

An extruded surface model is one of the motion-based surfaces. It can be created by moving any three-dimensional wire shape along a straight line. The direction can be specified by a view and giving a value for the total amount of movement, or supplying a wire to specify both the direction and the distance required for creating the surface. The command is Extruded Surface that is under Surface Create toolbar. Typing 'amextrudesf' at the command prompt can also activate the function.

A polyline was used to create an extruded surface model. The direction of extrusion was Z, that is normal to the personal computer screen. The extrusion distance was 50 mm with 0° taper angle. The extruded surface model then underwent the section cut process just like the previous illustrated models. Figure 4.3(d) shows the extruded surface and section cut surface models.

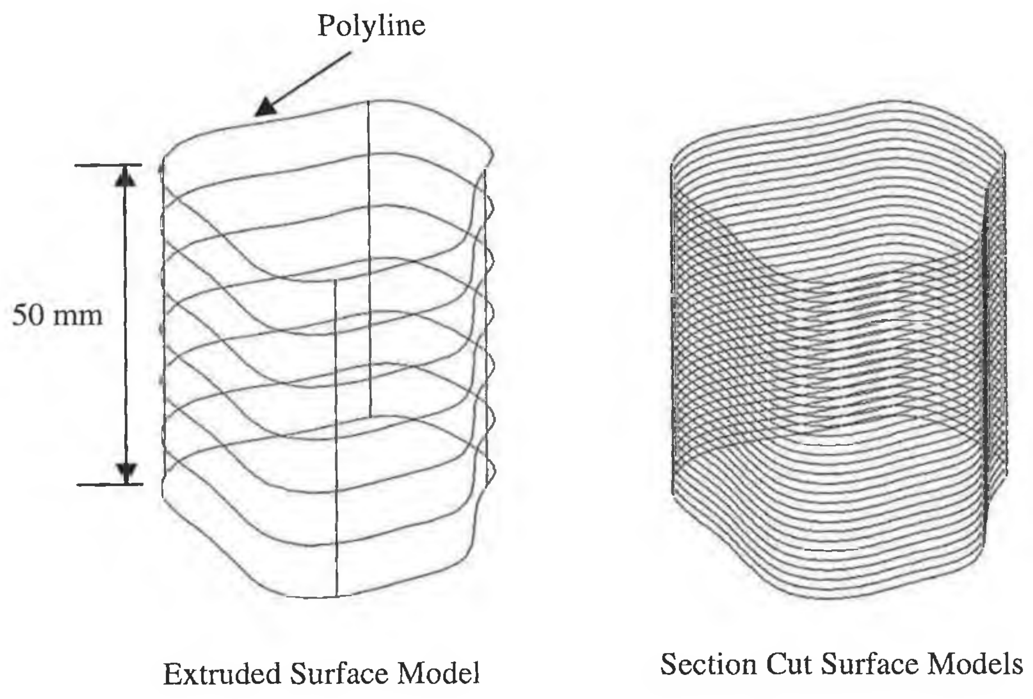
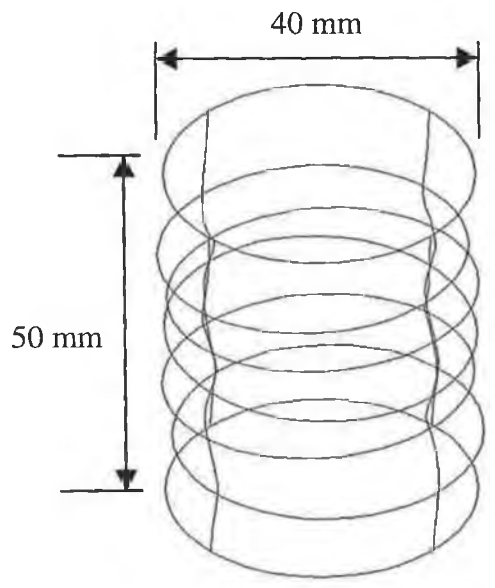


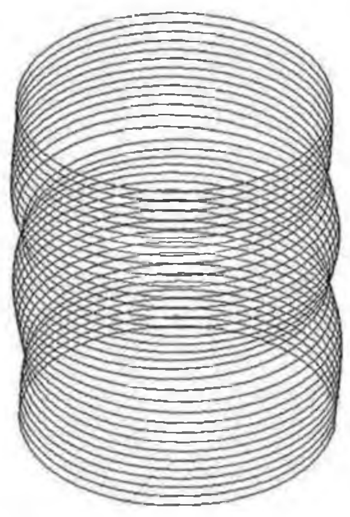
Figure 4.3(d) Extruded and Section Cut Surface Models

One of the motion-based surfaces is the tubular surface model. The command to create the Tubular Surface is under the Surface Create toolbar. Typing 'amtube' at the command prompt can activate the function too. It is used to create a tubular surface around a selected wire that becomes the axis of the tube.

The information needed for the construction of the tubular surface model is the tube diameter and the specified wire. The wire can be line, arc, polyline and spline. Figure 4.3(e) shows the tubular surface and its section cut surface models. The section cut process is same as the previous models. The model was built by using a three-dimensional rotated polyline and the tube diameter was 40 mm.



Tubular Surface Model



Section Cut Surface Model

Figure 4.3(e) Tubular and Section Cut Surface Models

Swept surface model is one of the motion-based surfaces. The command is Swept Surface that is under Surface Create toolbar. Typing 'amsweepsf' at the command prompt can activate the function too. It is used to create a surface by sweeping cross sections along a rail. Using a polyline cross section and a polyline as a rail, a swept surface model can be created. The model is shown in Figure 4.3(f) along with its section cut surface model. The section cut configuration was same all the previous models.

Skin surface models are created by "skinning over" a wireframe shape. Skin surface models can be visualised as a surface draped over, or skinned, across an existing wire structure. Ruled surface model is one of the skin surface models. Ruled surface models are constructed from only two path curves. The path curves can be line, arc, closed or open polylines. The way of creating a ruled surface model by using two closed polylines is shown in the following paragraphs.

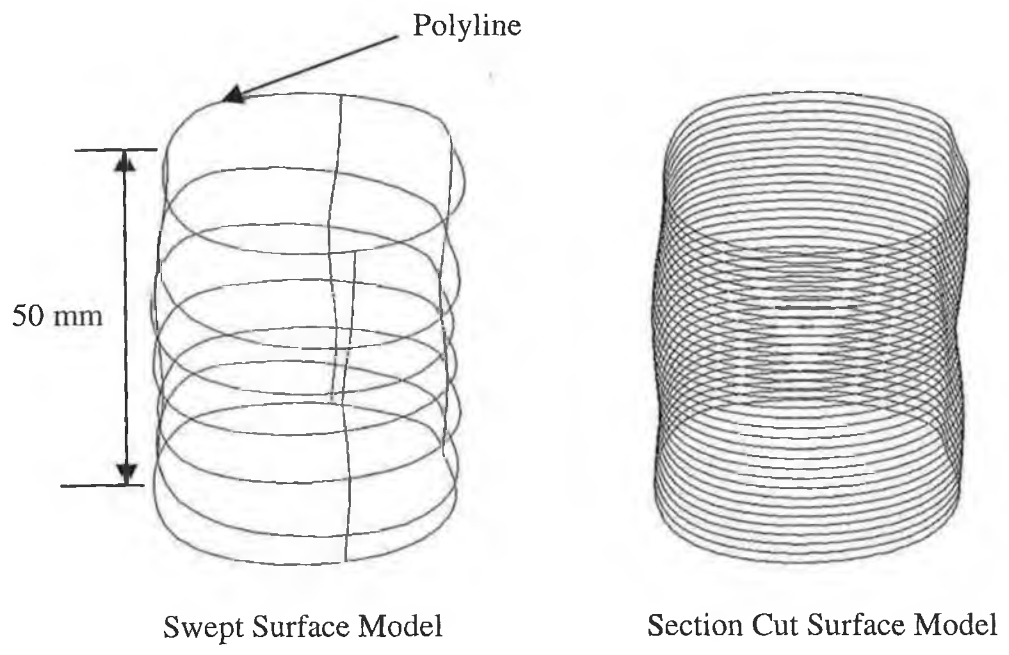


Figure 4.3(f) Swept and Section Cut Surface Models

First of all, using the Polyline command at the Draw toolbar for drawing up two closed polylines. Then, the polylines were edited by using Edit Polyline at the Modify toolbar so that it became a spline. Then, using Move command at the Modify toolbar or typing 'move' at the command prompt to separate the closed polylines so that the distance between them is 50 mm.

After that, typing 'amrule' at the command prompt or clicking the Ruled Surface at the Surface Create toolbar to call up the ruled surface creation function. Then, selecting the two smooth polylines as the path curves. A ruled surface will be created like the one in Figure 4.3(g). The ruled surface model was later taken through the section cut operation to produce the multi-layer model. The section cut operation was the same as the previous described models. The section cut model is illustrated in Figure 4.3(g) too.

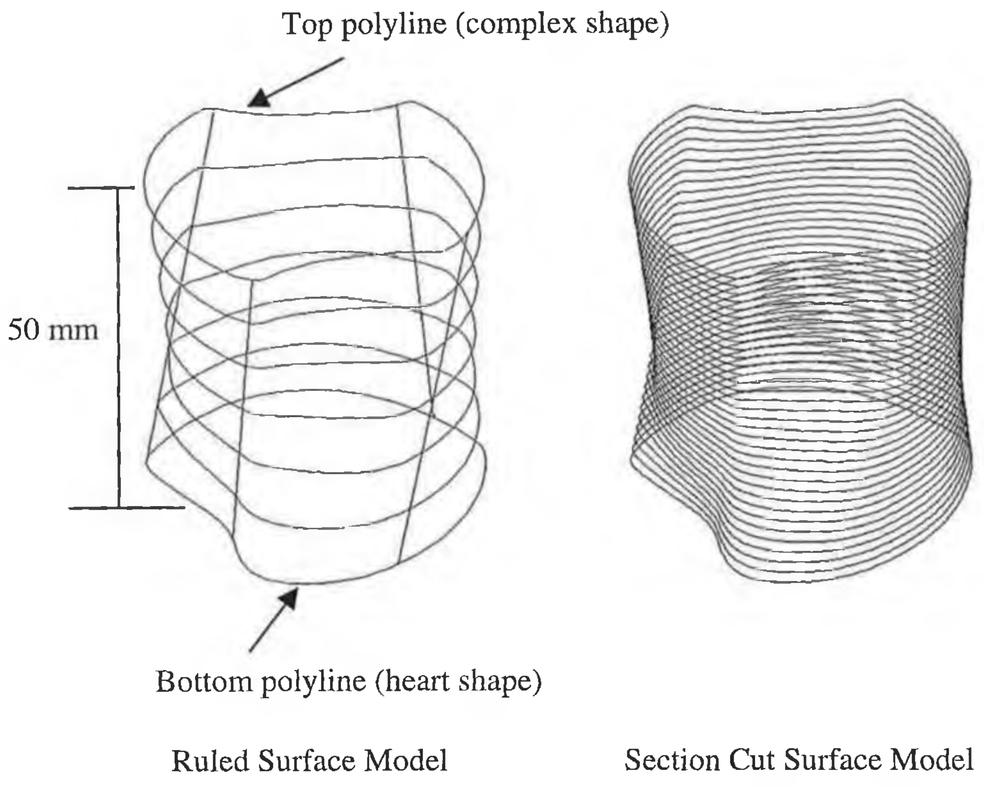
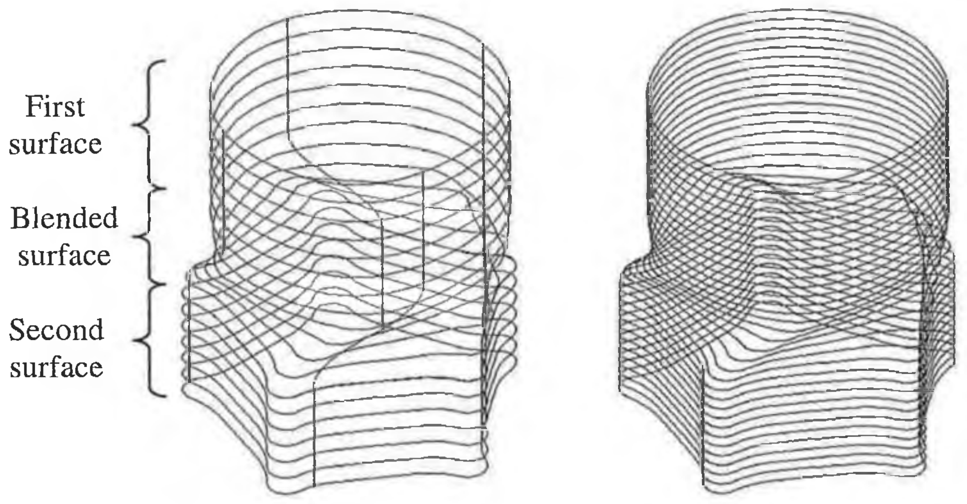

Top polyline (complex shape)
50 mm
Bottom polyline (heart shape)
Ruled Surface Model
Section Cut Surface Model

Figure 4.3(g) Ruled and Section Cut Surface Models

A blended surface is one of the derived surfaces. It is created between two, three, or even four other surfaces. The blended surfaces is tangent to the surfaces from which it is created. Blended surfaces may also be created between wires or between combinations of wires and surfaces. The Blended Surface command is under the Surface Create toolbar. Typing ‘amblend’ at the command prompt can also call up the function. Selecting the desired surfaces for the creation of blended surface will construct the model. Figure 4.3(h) shows the blended surface model. First and second surfaces were used to derive the blended surface. Section cut model is also shown in the figure. The section cut model was created by using the same procedures as the previous models.



Blended Surface Model Section Cut Surface Model

Figure 4.3(h) Blended and Section Cut Surface Model

Of all the surface models described above, only the ruled surface model was selected for the project due to its complexity and versatility. Cone, cylinder and revolved surface models are symmetrical model. Although extruded, tubular and swept surface models are not symmetrical, but they are simple. Blended surface model is not suitable since the distance between the highest and lowest point (from the axis of rotation) of the surface is unpredictable. The milling depth might need to be more than 10 mm in some cases. Ruled surface model is the best method for creating complex surface model because its models are asymmetric, complex and have predictable milling depth. As a result, more ruled surface models were created. The models are shown from

Figure 4.3(i) to Figure 4.3(w) with their section cut models. Two or more polylines were used to create the ruled surface models. The shapes of the polylines are circle, heart, complex, star, pentagon, cross and square.

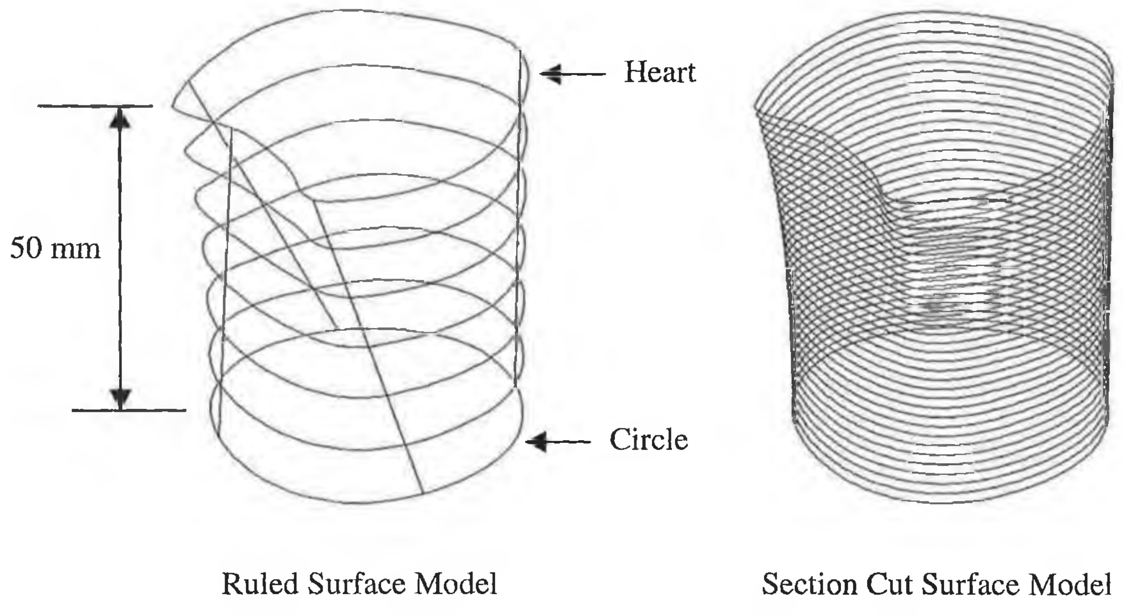


Figure 4.3(i) Circle to Heart Model

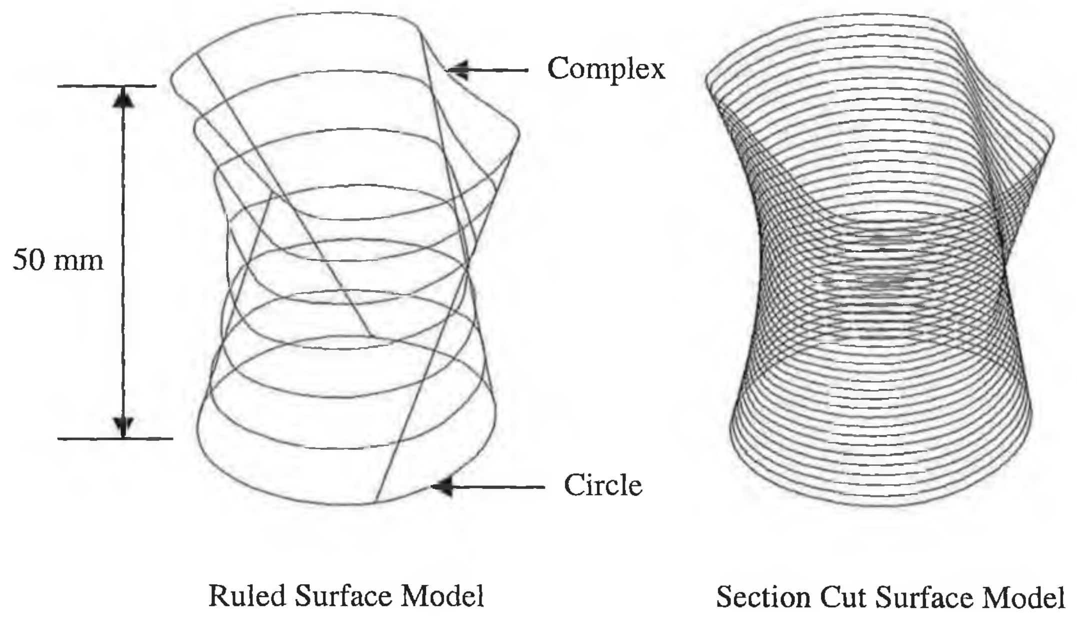


Figure 4.3(j) Circle to Complex Model

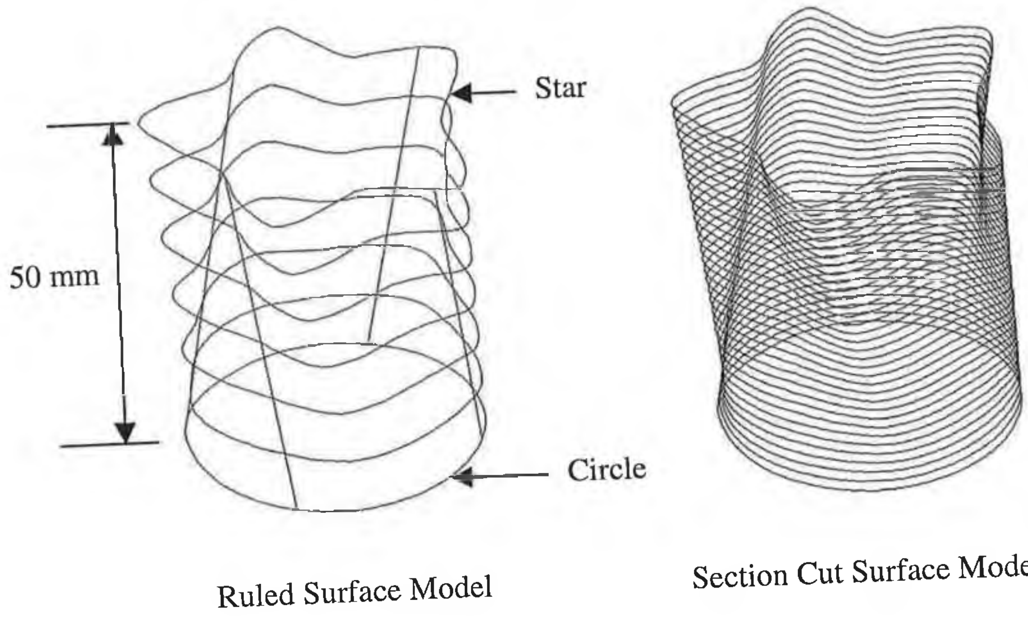


Figure 4.3(k) Circle to Star Model

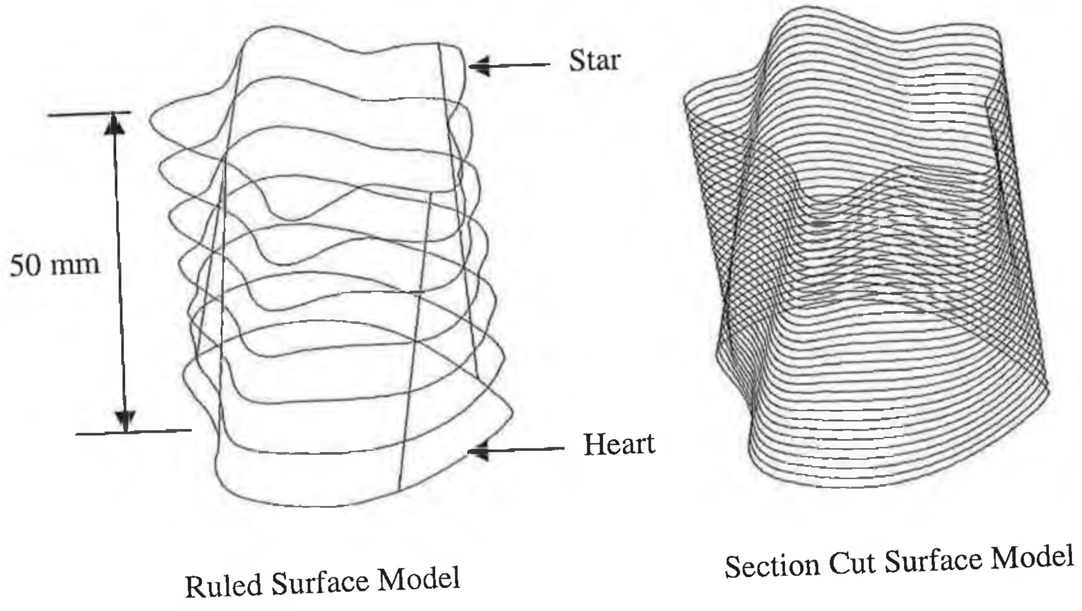


Figure 4.3(l) Heart to Star Model

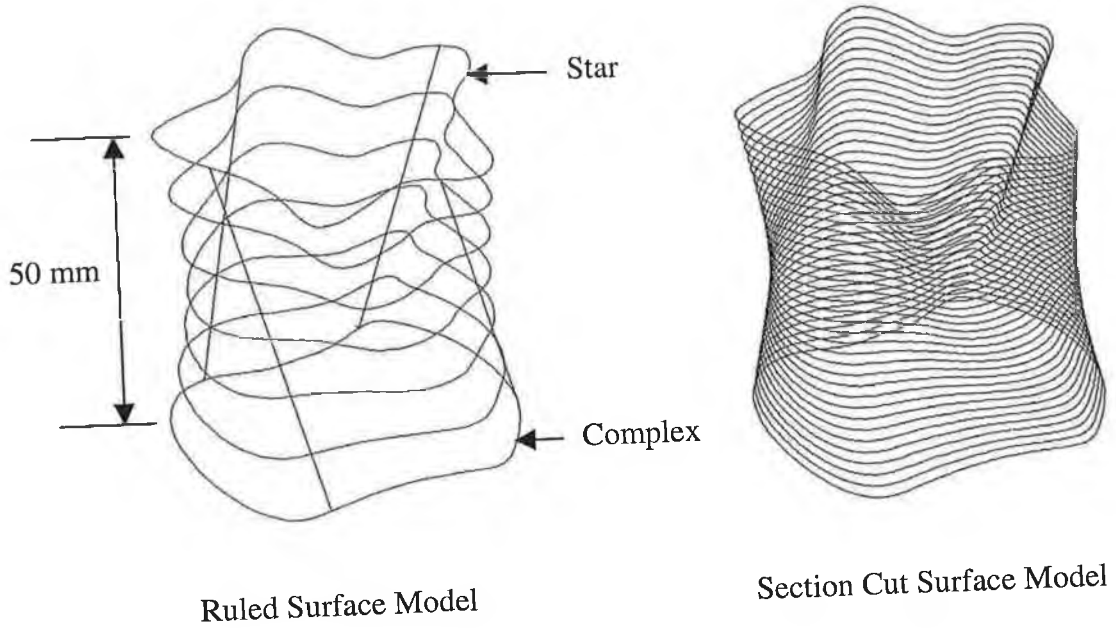


Figure 4.3(m) Complex to Star Model

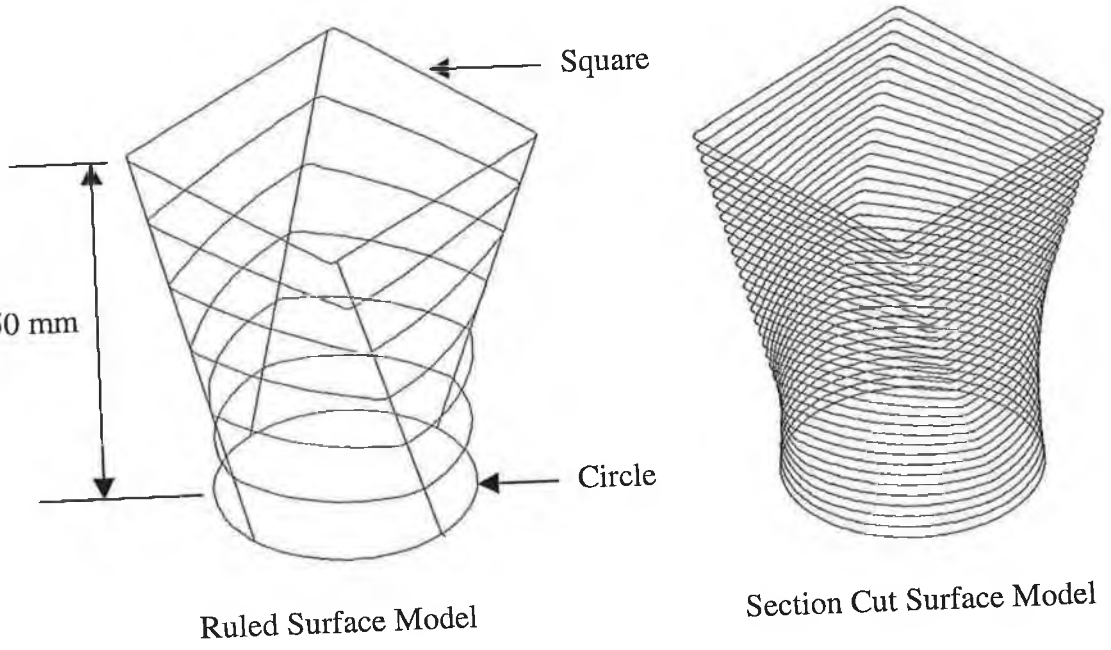


Figure 4.3(n) Circle to Square Model

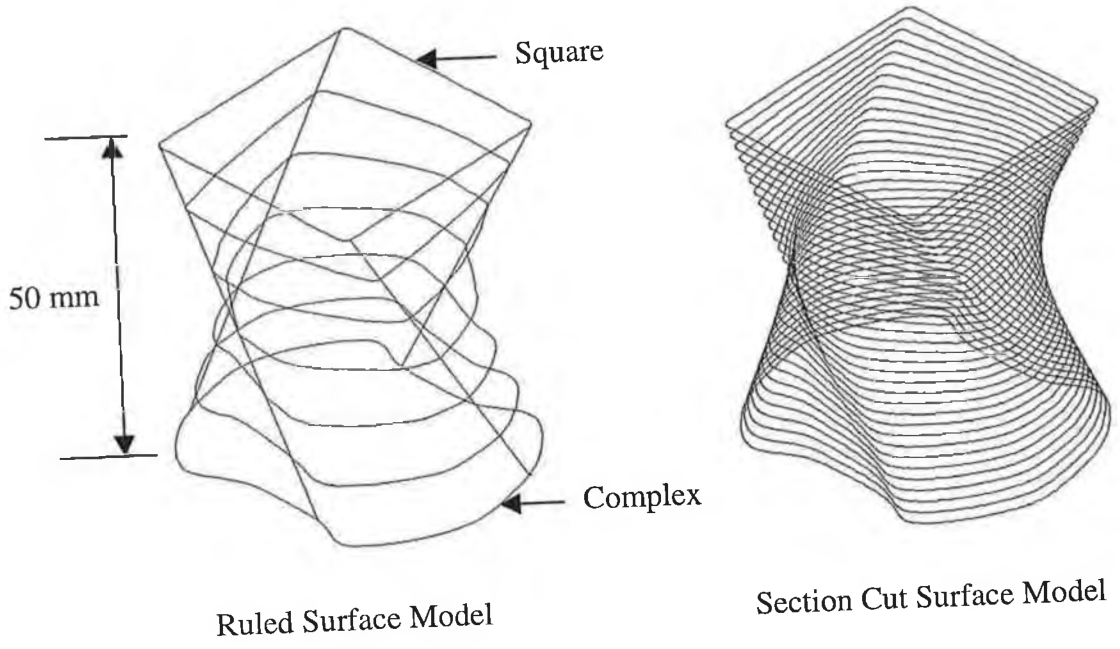


Figure 4.3(o) Complex to Square Model

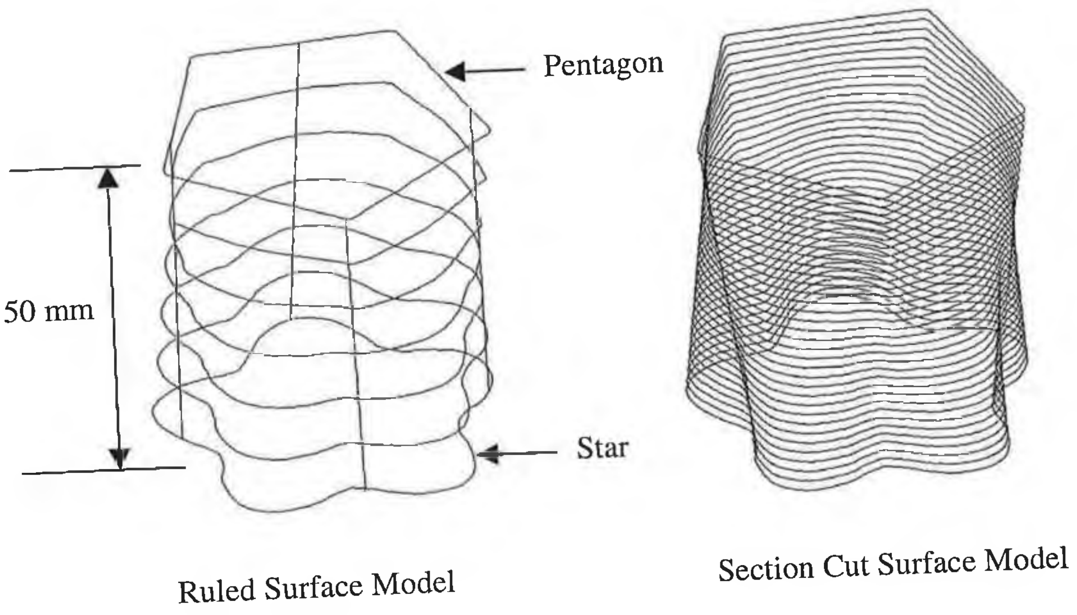


Figure 4.3(p) Star to Pentagon Model

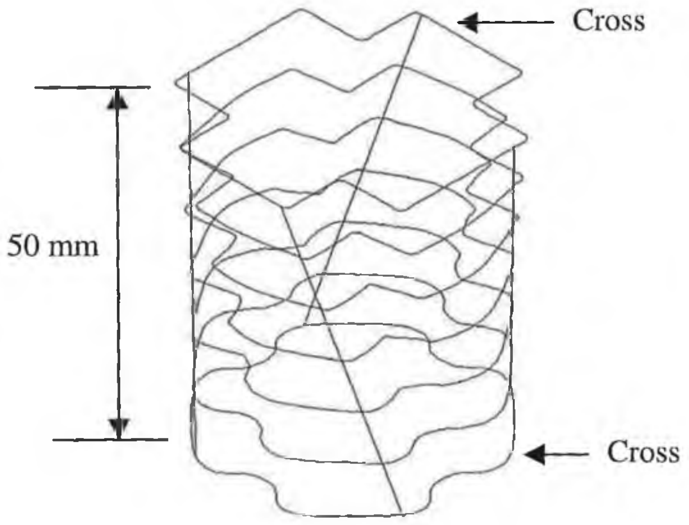


Figure 4.3(q) Cross to 45° Rotated Cross Model

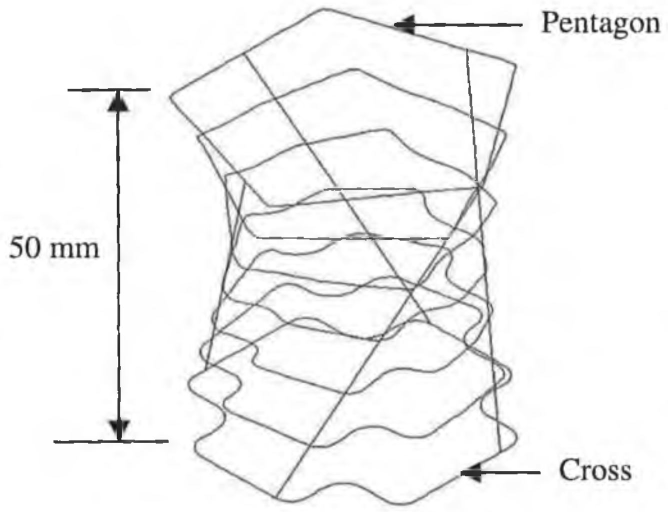


Figure 4.3(r) Cross to Pentagon Model

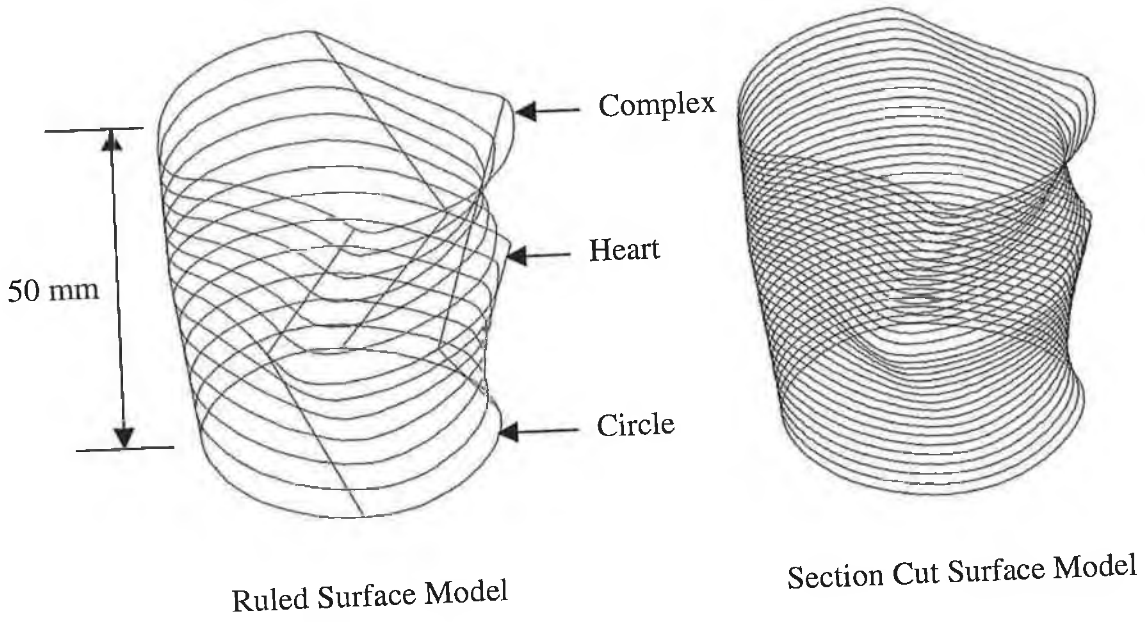


Figure 4.3(s) Circle to Heart to Complex Model

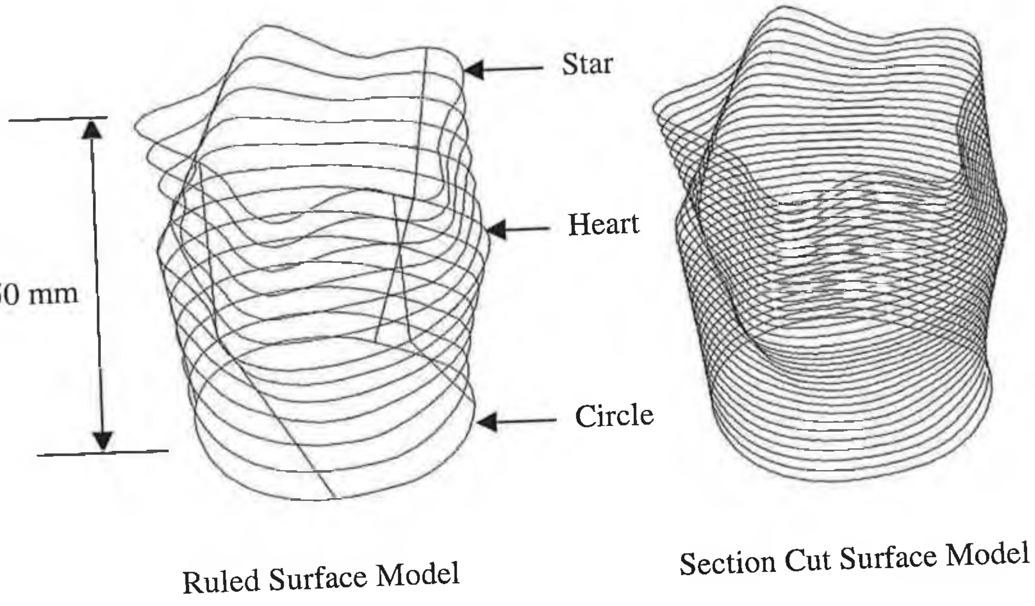


Figure 4.3(t) Circle to Heart to Star Model

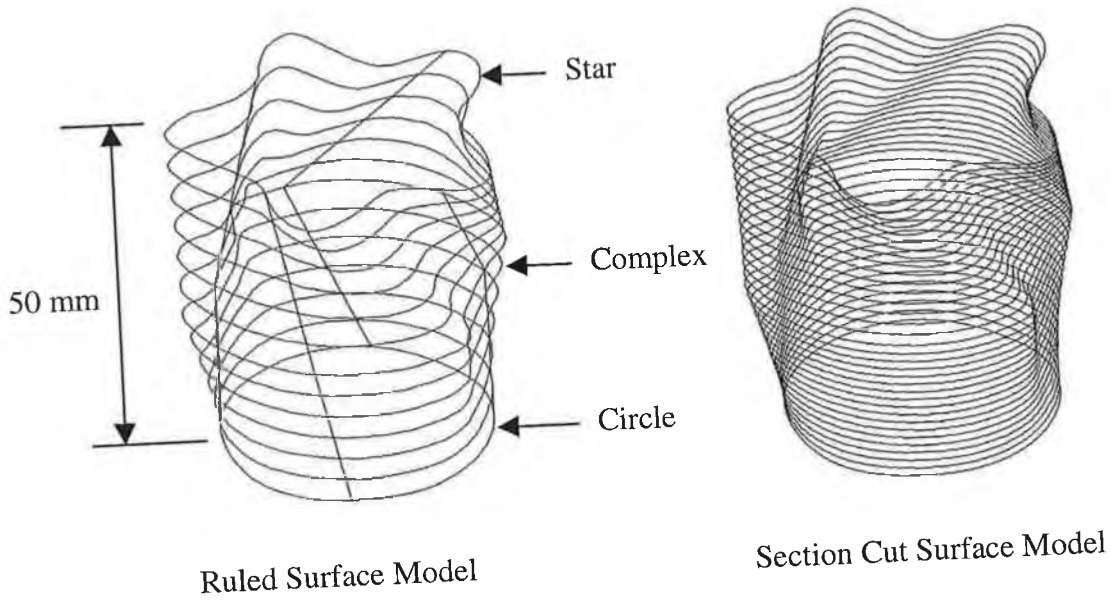


Figure 4.3(u) Circle to Complex to Star Model

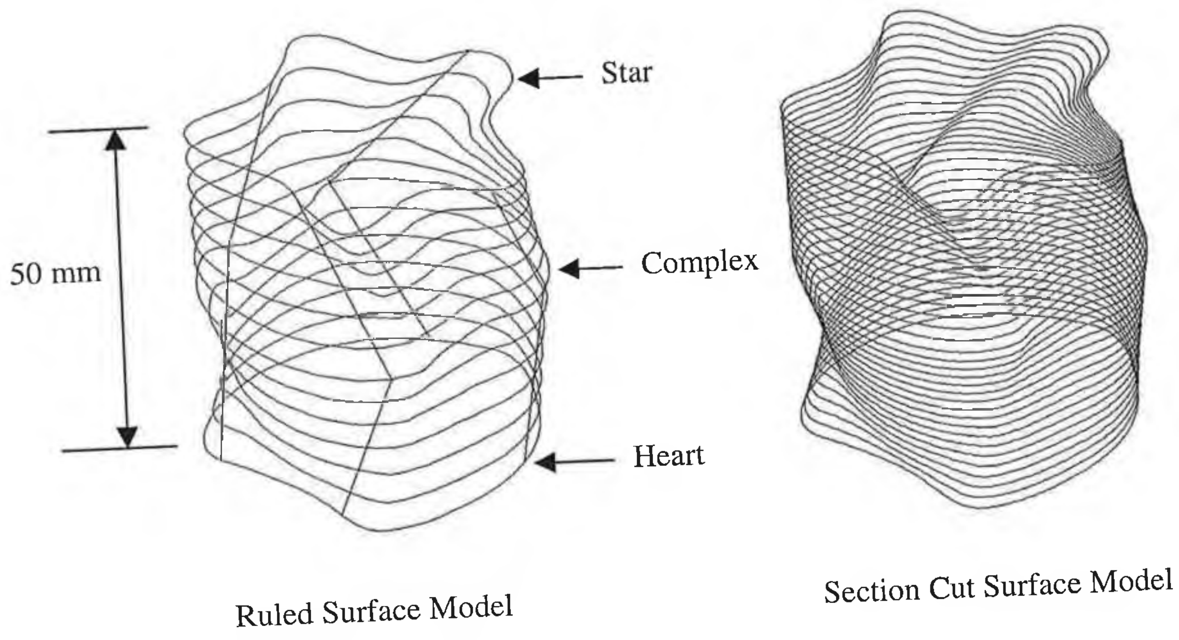


Figure 4.3(v) Heart to Complex to Star Model

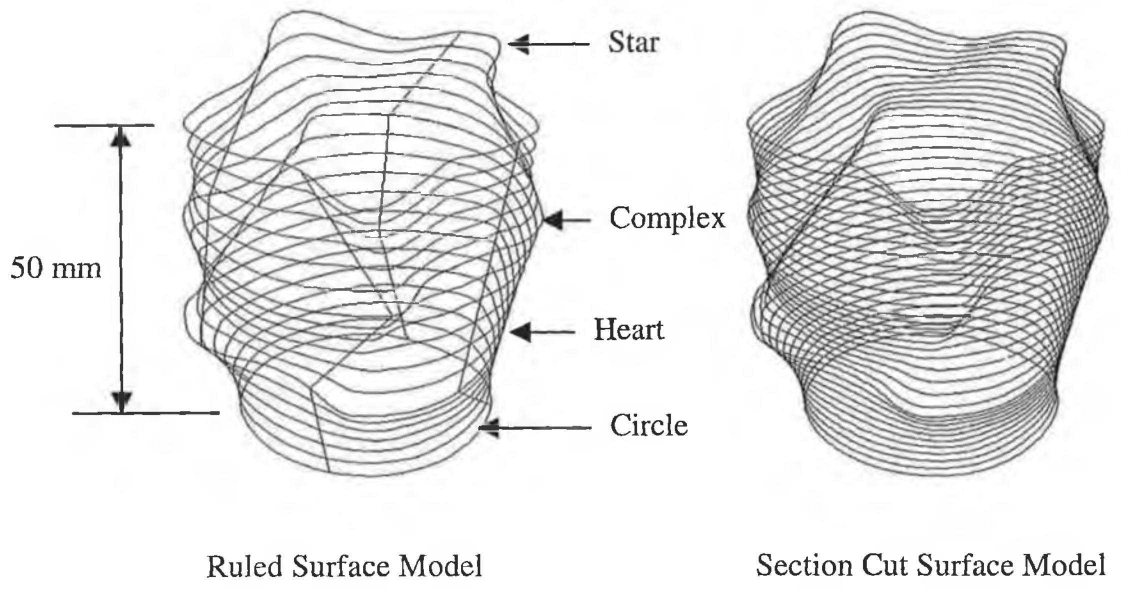


Figure 4.3(w) Circle to Heart to Complex to Star Model

4.4 CAD FOR ADDITIVE PROTOTYPING

It is important to know the physical limitation of the manipulator and the proposed additive prototyping equipment before attempting to create the solid model and the subsequent models manipulation. The criteria that must be followed in designing the solid model are:

(1) *Shape*. The proposed additive prototyping equipment is a vertical semi-liquid deposition device. The semi-liquid material will be deposited onto the rotating cylindrical core material held by the grippers. As a result, only manipulations along and around x-axis are used in handling the model. Hence, the shapes of the solid model should be in cylindrical form.

(2) *Length.* The three-pin grippers of the manipulator can hold a cylindrical block with the size ranging from 120 mm to 125 mm in length and 40 mm to 150 mm in diameter. But, due to the pins of the grippers, the length of the surface model should be less than 110 mm.

(3) *Reference Point.* The model is a cylindrical surface form with a rotating axis at the (100, 100, 0). The x = 100, y = 100 and z = 0 is an important reference point for the CAM programs in the later stage. Motion parameters are produced based on that reference point.

4.4.1 Object Modelling and Manipulation

Some of the important modelling configurations of AutoSurf need to be checked and set before creating the solid model. The configurations are:

(1) *Units.* It is for selecting co-ordinates and angle display formats and precision. The command can be found under the pull down menu of Data or by typing 'units' at the command prompt. Direction of rotation can be set under this command.

(2) *Drawing Limits.* It is for setting and controlling the two and three-dimensional drawing boundaries. The command is under the Data pull down menu too and can be called up by typing 'limits' at the command prompt. Drawing limits has been set to A4 size.

(3) *Layers.* This command is also under the Data pull down menu and can be called up by typing 'layer' at the command prompt. Under this command, the user can set as

many line types and line colours as possible so that they can be used in the subsequent modelling.

(4) *Drawing Aids.* This command is under the pull down menu of Options and can be called up by typing 'ddrmodes' at the command prompt. The command will enable the setting of Grid spacing and Snap spacing that will make the modelling easier.

(5) *Preferences.* It is for customising the AutoSurf settings. It can be called up by typing 'preferences' at the command prompt or under the Options pull down menu. It can be used to set the types of digitizer input, font type, font size, background colour and others.

A solid cone model was created by using AutoSurf. Then, four internal cavities were created in the solid cone by using the Boolean operator's subtract command. The solid cone with four internal cavities is called the parent model. Later, multiple smaller diameter solid models were derived from the parent model by using the Boolean operator's intersection command. The smaller diameter solid models were then changed into surface models. The surface models went through the section cut process to become section cut models. Only the section cut models can be convert into useful neutral format files. The parent model is shown in Figure 4.4(a) in wire frame format.

There is no one step command in creating a solid cone. A solid cone model can be created by revolving a closed polyline. Four points were used to create the close polyline. The co-ordinates of the points were (100, 100, 0), (100, 200, 0), (50, 200, 0) and (60, 100, 0). The axis of revolution started from (100, 100, 0) and ended at (100, 200, 0). The angle of revolution was a full circle. The revolving command that is under the Solids toolbar can be called up by accessing the Tools pull down menu's Toolbars. Typing 'revolve' at the command prompt can also activate the function.

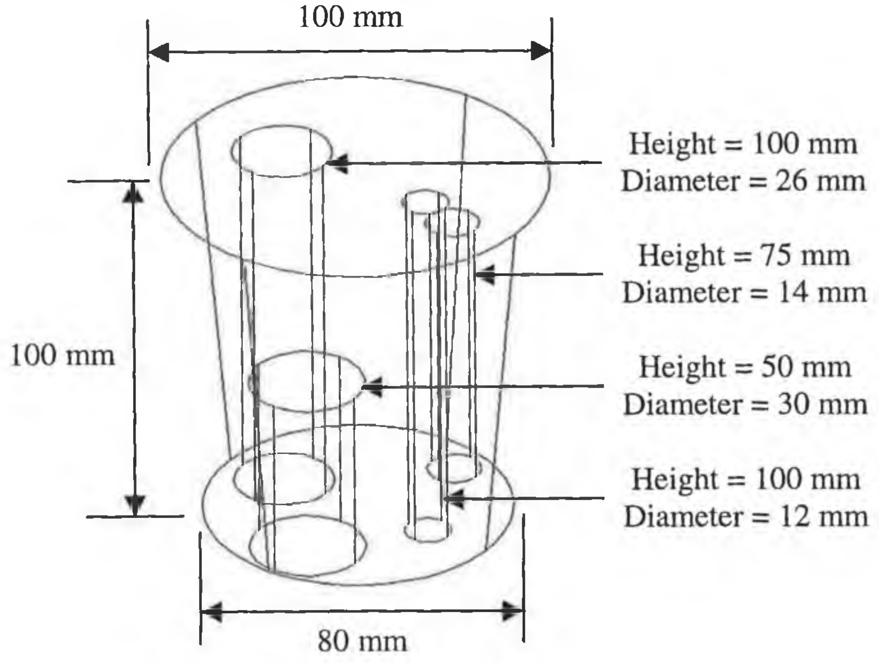


Figure 4.4(a) Solid Model in Wireframe Representation

The solid cone model was then rotated in three-dimensional around x-axis at (100, 100, 0) with 90 degree rotation angle. After that, four solid cylinders were built inside the solid cone model as shown in Figure 4.4(a). Subtract command was used to subtract the cylinders from the solid cone model. In the end, the solid cone model was having four internal cylindrical cavities. The Boolean operators of Union, Intersection and Subtract can be accessed from the Modify toolbar. Typing 'union' or 'intersect' or 'subtract' at the command prompt will give the same effect.

Eight smaller diameter solid models were derived from the parent model. The eight derived models' maximum diameters are 20 mm, 30 mm, 40 mm, 50 mm, 60 mm, 70 mm, 80 mm and 90 mm. Intersecting various diameter solid cylinders with the parent model will create the eight models. Once again, Boolean operator of Intersection was used in deriving the smaller models.

The derived solid models and the parent model were then be converted into surface models. The Convert All command in the Surface Edit toolbar was used to convert all the solid faces into surfaces. Typing 'am2sf' at the command prompt can activate the function as well. After obtaining the surface models, the top and bottom surfaces of each model were deleted.

The surface models were then went through the section cut operation. The section cut type was parallel, UCS was the initial plane, the step over distance is 3 mm and it will stop at the height of 99 mm. So, every surface model was section cut into 34 layers. The surface models and their respective section cut models are shown in the following diagrams from Figure 4.4(b) to Figure 4.4(j).

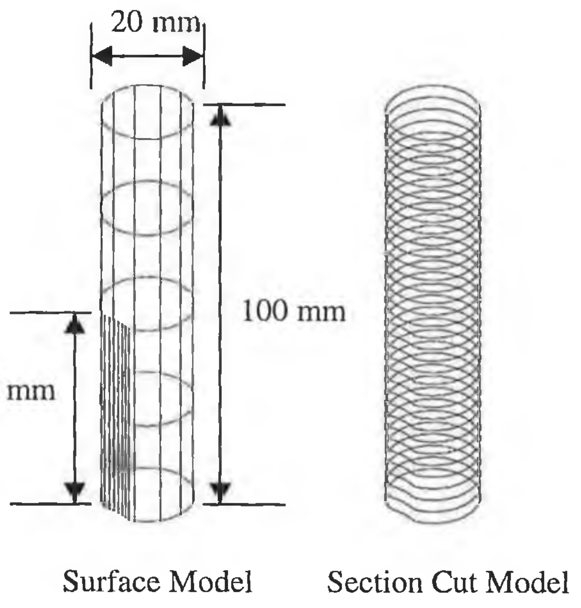


Figure 4.4(b) 20 mm Diameter Surface and Section Cut Models

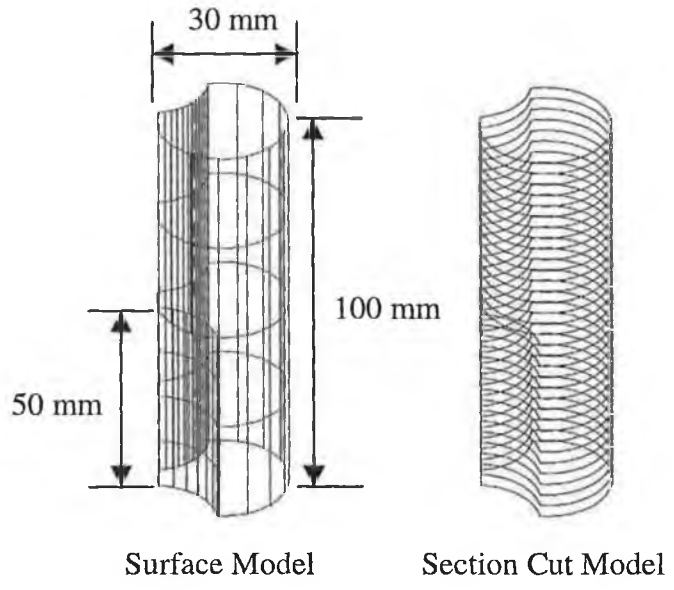


Figure 4.4(c) 30 mm Diameter Surface and Section Cut Models

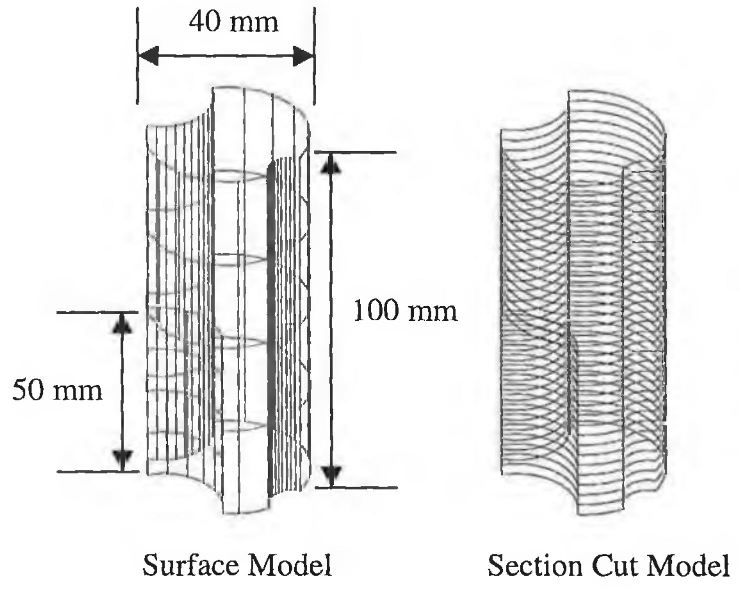


Figure 4.4(d) 40 mm Diameter Surface and Section Cut Models

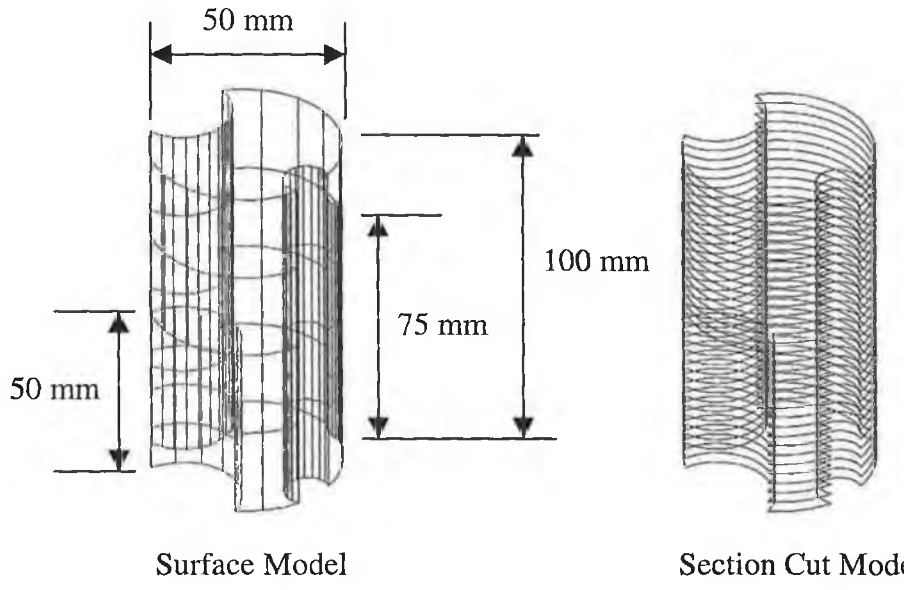


Figure 4.4(e) 50 mm Diameter Surface and Section Cut Models

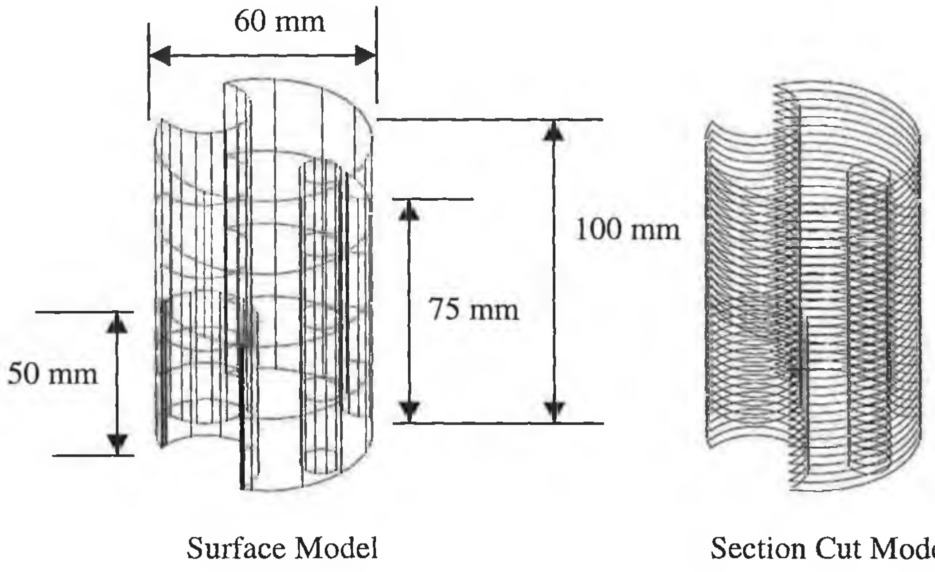
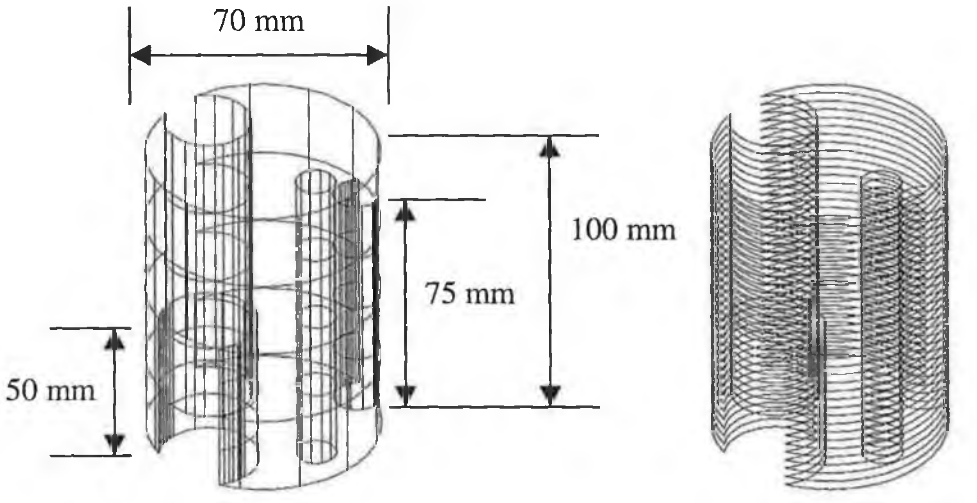
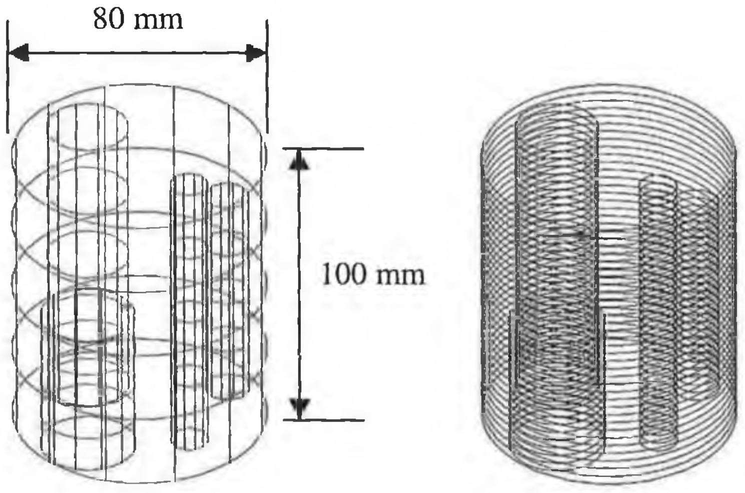


Figure 4.4(f) 60 mm Diameter Surface and Section Cut Models



Surface Model Section Cut Model

Figure 4.4(g) 70 mm Diameter Surface and Section Cut Models



Surface Model Section Cut Model

Figure 4.4(h) 80 mm Diameter Surface and Section Cut Models

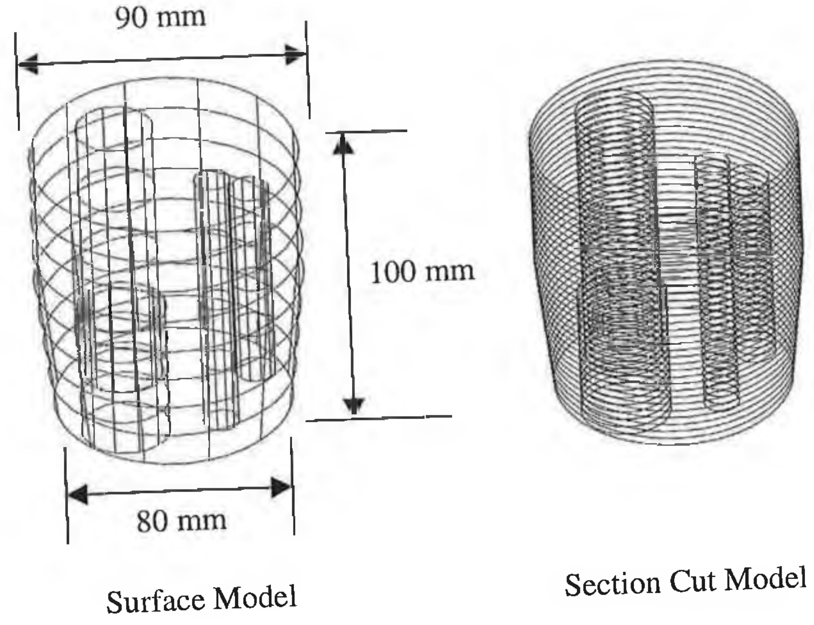


Figure 4.4(i) 90 mm Diameter Surface and Section Cut Models

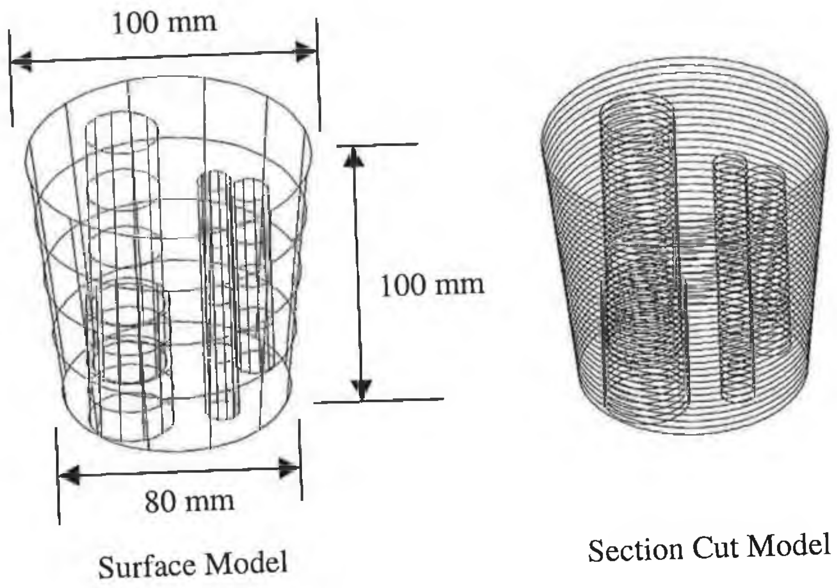


Figure 4.4(j) Complete Surface and Section Cut Models

4.4.2 Graphical Simulation

The additive prototyping process was simulated graphically by a program which was built by the author using AutoLISP programming language [101]. AutoLISP was created by the developers of the AutoSurf [100]. It is an integral part of the AutoSurf package. It is a small subset of the Common LISP programming language. It adheres closely to the same syntax and conventions, but has many additional functions specific to AutoSurf.

AutoSurf has a built-in LISP interpreter that the user uses to enter AutoLISP code at the command line or to load AutoLISP code from external files. AutoLISP applications or routines can interact with the CAD software in many ways such as prompting the user for input, access built-in AutoSurf commands directly, and modify or create objects in the drawing database.

Since no compiling is required, AutoSurf can be used to read the AutoLISP code directly. The results are shown immediately after typing the code at the command line. AutoLISP can be used to write macro programs and functions in a powerful, high-level language suited to graphics applications. Its applications are stored in ASCII (American Standard Code for Information Interchange) text files.

The making of the solid model in Figure 4.4(a) was simulated graphically by using AutoLISP in the AutoSurf environment. In the simulation, a simple deposition tool was created together with nine solid models from Figure 4.4(a) to Figure 4.4(i).

The complete model was first created. The graphical and non-graphical entity definition data of the model was immediately extracted from the drawing by using AutoLISP commands like 'setq', 'prin1', 'entget' and 'entlast'. All the other solid models were created based on the intersection of various diameter solid cylinders with the complete model. The entity definition data of the models were extracted immediately upon the creation of each solid model.

The entity definition data are listed in pairs of "key/value" that define the drawing. The "key" for any of the pairs in the definition data is always an integer, whose value dictated the interpretation of the "data" part of the pair. The definition data of a circle entity within the system is shown and explain in Table 4.4(a).

Table 4.4(a) Entity Definition Data

Key/Value Pair	Explanation
((-1 . <Entity name: d13d50>)	; -1 indicates internal identifier for this circle object
(0 . "CIRCLE")	; 0 indicates type of object
(330 . <Entity name: d13cc8>)	; 330 indicates internal identifier of the circle's container
(5 . "52")	; 5 indicates the "handle" of the object
(100 . "AcDbEntity")	; first 100 indicates the most ancestral class of the object
(67 . 0)	; 67 indicates the circle's colour
(8 . "0")	; 8 indicates the circle's layer
(100 . "AcDbCircle")	; subsequent 100 indicates more derived class of object
(10 3.0 5.0 0.0)	; 10 indicates centre of circle
(40 . 1.6)	; 40 indicates circle radius
(210 0.0 0.0 1.0))	; 210 indicates vector normal to plane of circle

The AutoLISP command of 'entmake' is used to create and display the solid models. The entity definition data has to be modified - the first key/value pair of the data file has to be omitted. The data files can also be changed so that the models have different sizes, colours and so on. The basic picture frames of the graphic simulation process are shown from Figure 4.4(k) to Figure 4.4(t).

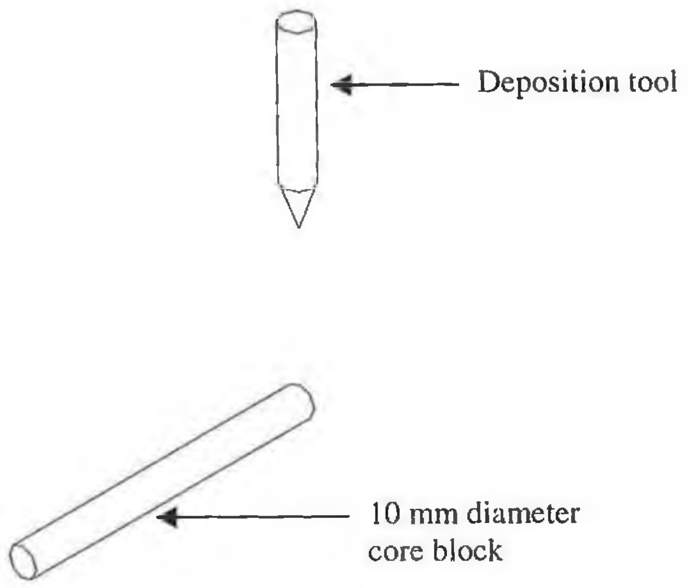


Figure 4.4(k) 10 mm Diameter Core Block with Deposition Tool

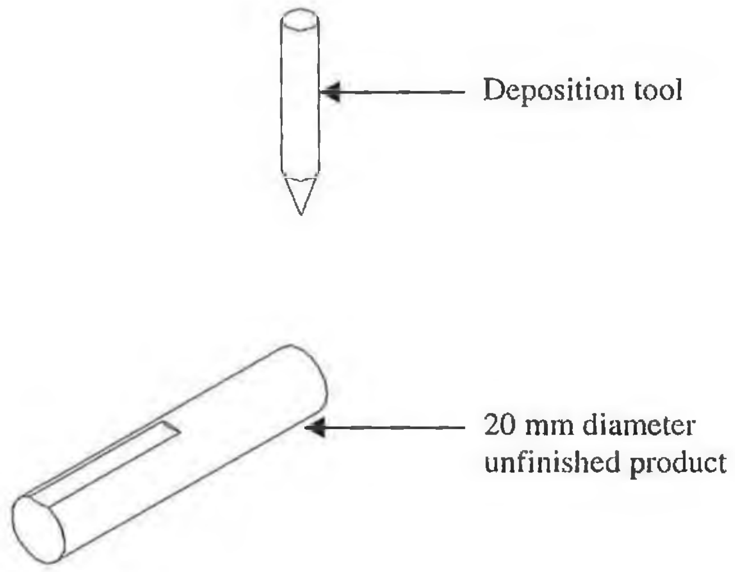


Figure 4.4(l) 20 mm Diameter Product with Deposition Tool

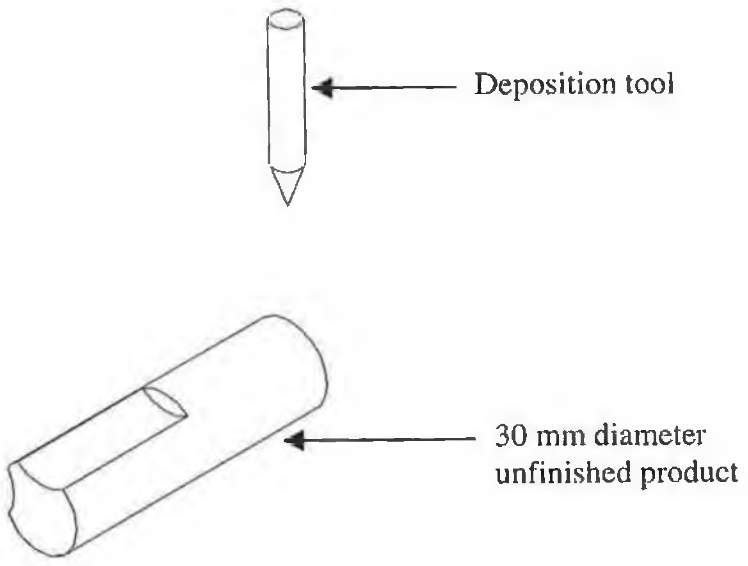


Figure 4.4(m) 30 mm Diameter Product with Deposition Tool

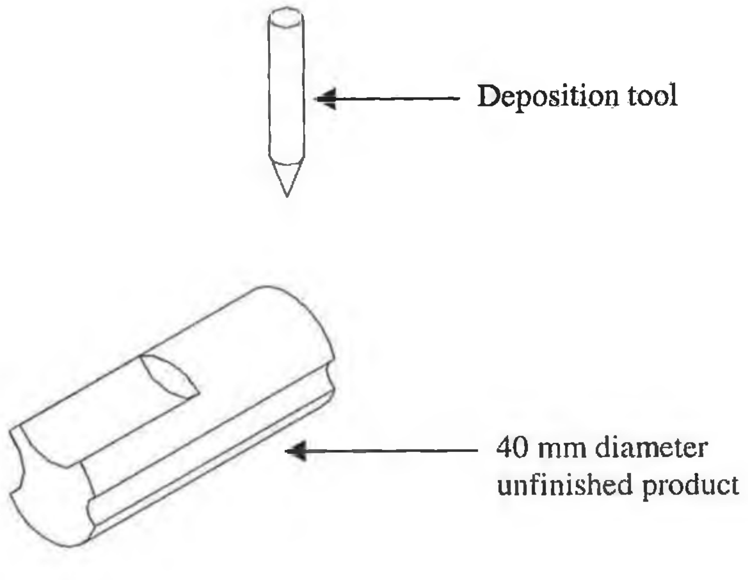


Figure 4.4(n) 40 mm Diameter Product with Deposition Tool

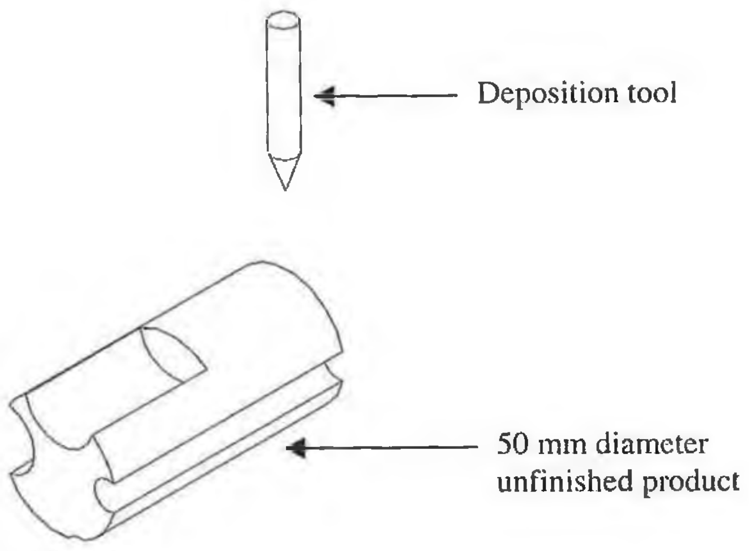


Figure 4.4(o) 50 mm Diameter Product with Deposition Tool

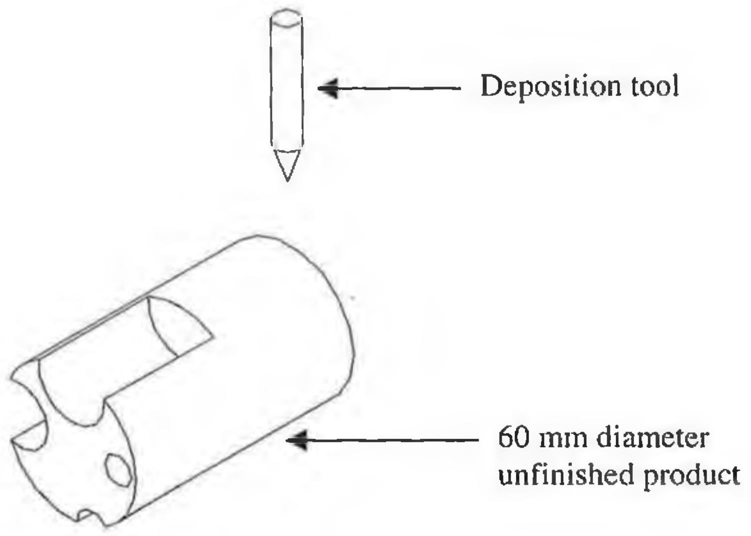


Figure 4.4(p) 60 mm Diameter Product with Deposition Tool

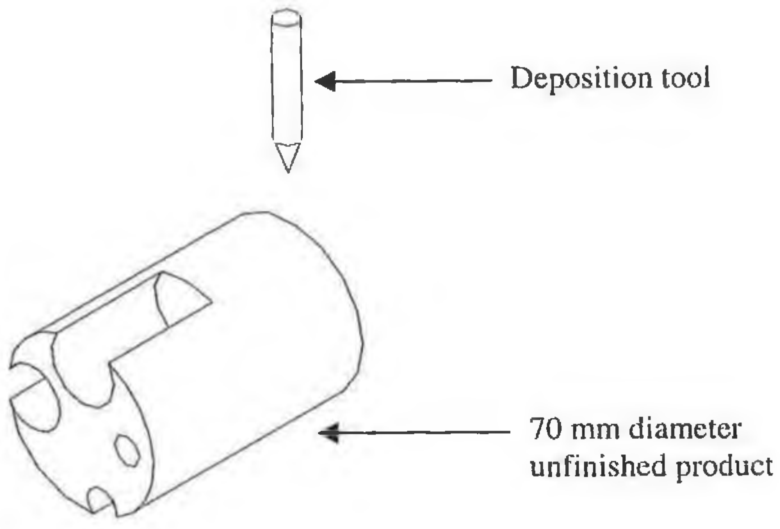


Figure 4.4(q) 70 mm Diameter Product with Deposition Tool

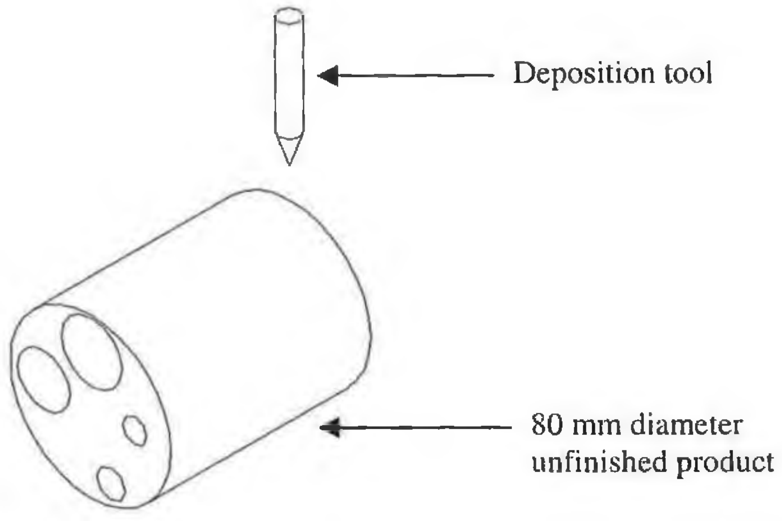


Figure 4.4(r) 80 mm Diameter Product with Deposition Tool

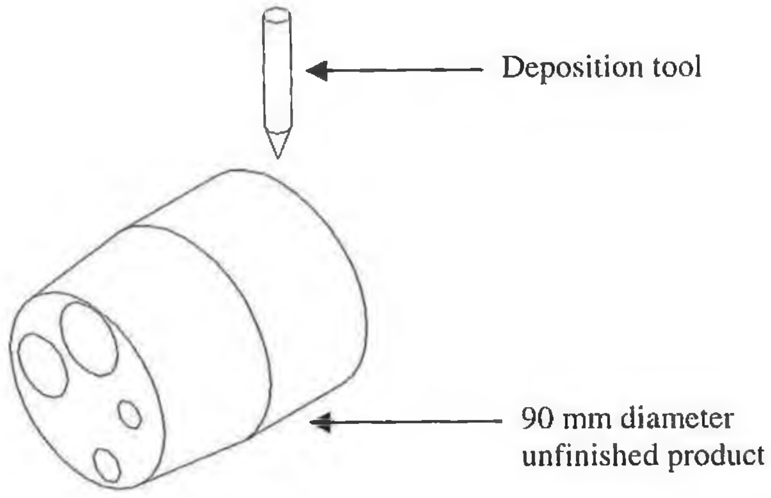


Figure 4.4(s) 90 mm Diameter Product with Deposition Tool

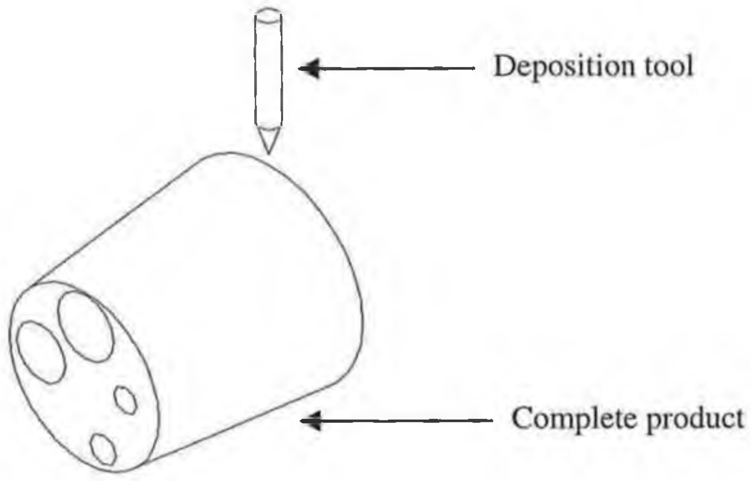


Figure 4.4(t) Complete Product with Deposition Tool

The simulation program is illustrated graphically in Figure 4.4(u).

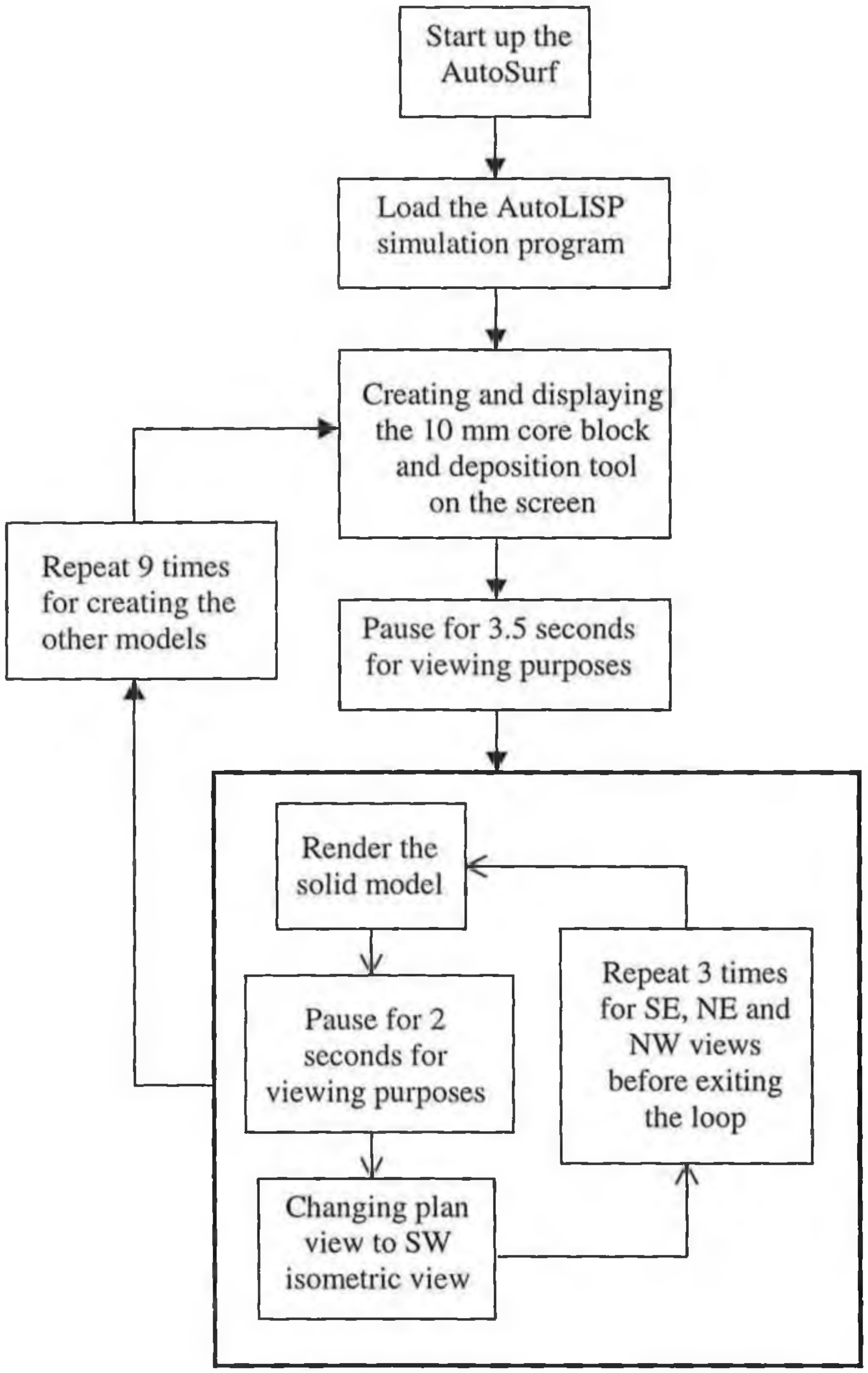

Start up the AutoSurf
Load the AutoLISP simulation program
Creating and displaying the 10 mm core block and deposition tool on the screen
Repeat 9 times for creating the other models
Pause for 3.5 seconds for viewing purposes
Render the solid model
Pause for 2 seconds for viewing purposes
Changing plan view to SW isometric view
Repeat 3 times for SE, NE and NW views before exiting the loop

Figure 4.4(u) Graphic Simulation Program Flow

The AutoLISP graphic simulation program is listed in Appendix C. The program utilised AutoLISP commands like 'entmake', 'setq', 'while', 'princ', 'command' and AutoSurf commands like 'avrender' and 'vpoint'.

The command 'entmake' is for creating and displaying the solid model on the screen. 'setq' is for setting the program variable, 'while' is for constructing the time delay function and 'avrender' is for creating a realistically shaded image of a three-dimensional solid model. The command 'vpoint' is for setting the viewing direction for a three-dimensional visualisation of the drawing. Four kinds of viewing direction were provided, namely Southeast (SE), Southwest (SW), Northeast (NE) and Northwest (NW).

4.5 DRAWING INTERCHANGE FORMAT (DXF)

In this project, only the section cut models in the previous sections can be converted into useful neutral format files. The subsequent CAM programs can be used to analyse the neutral format files and generate useful data for subtractive and additive prototyping processes. Drawing interchange format (DXF) file has been adopted as the neutral format file in this project.

DXF is written in human-readable character codes - ASCII (American Standard Code for Information Interchange) text. It contains all the necessary and important geometry and graphics entities of a model [102]. DXF was originally proposed by Autodesk Inc. as a method to allow for transferring data between different versions of

AutoCAD [43]. It also allows drawings to be exchanged between AutoCAD or AutoSurf on different types of computer [36].

AutoSurf's DXF file is the best neutral format file in this project because of the following reasons:

(1) It is easy to be understood since it is quite verbose. It contains strings, integers and floating-point numbers only.

(2) Its model information is arranged. It uses one line for each data item. It is perfect for data filtering and extraction in programming.

(3) It is relatively simple and has enormous future prospect since most personal computer-based CAD software will read and write DXF formatted files.

(4) Its file size is very small because specific entities (drawing objects) can be selected by the user for producing the neutral format file.

(5) Its data accuracy can be determined by the user up to 16 decimal places.

DXF object file size is larger than the DXF entities file size. It is very hard to decode the DXF object file. DXF binary file is more compact than the entities file but it is not written in a human-readable form. Only entities contain the needed data in this project. As a result, the neutral format file should contain the entities data only.

For producing DXF entities file by using the AutoSurf, the author needed to type 'dxfout' at the command prompt. Before producing the DXF file, a statement will be displayed at the command prompt like this:

"Enter decimal places of accuracy (0 – 16)/Objects/Binary<6>"

The author needed to type 'entities' at the end of the statement. Then, the author will need to select the entities from the drawing file; only those selected will be transferred into the DXF entities file. The default accuracy is six decimal places.

Essentially a DXF file is composed of pairs of codes and the associated values. The codes, known as group codes, indicate the type of value that follows. Using these group code and value pairs, a DXF entities file is organised into sections, which are composed of records, which in turn are composed of a group code and a data item. Each group code and value are on its own line in the DXF file.

Each section starts with a group code 0 followed by the string, SECTION. This is followed by a group code 2 and a string indicating the name of the section (for example, ENTITIES). Each section is composed of group codes and values that define its elements. A section ends with a 0 followed by the string ENDSEC. For example, a complete DXF entities file definition of a circle is listed as below (comments in brackets).

```
0
SECTION                  (a new section)
  2
ENTITIES                 (drawing entities section)
  0
CIRCLE                   (circle is the entity)
  5
6F                       (handle)
100                      (subclass marker)
AcDbEntity
  8
0                        (name of the layer)
100                      (subclass marker)
AcDbCircle
 10
100.0                    (x co-ordinate of the centre of the circle)
 20
100.0                    (y co-ordinate of the centre of the circle)
 30
0.0                      (z co-ordinate of the centre of the circle)
 40
25.0                     (radius of the circle)
  0
ENDSEC                   (end of entities section)
  0
EOF                      (end of ASCII text file)
```

4.6 CAM TOOL

All computer-aided manufacturing (CAM) programs in this project were written by the author using American National Standards Institute (ANSI) C programming language. The source codes were compiled in Borland C++ version 4.5.

C language was created by Dennis Ritchie at the Bell Telephone Laboratories in 1972. Initially, It was used for designing UNIX operating system. Then, programmers around the world started to use C language for other purposes since it is so powerful and flexible. Different organisations created their own version of C, and subtle differences between implementation started to give programmers headaches. In response to this problem, the American National Standards Institute (ANSI) formed a committee in 1983 to establish a standard definition of C, which became known as ANSI Standard C. With few exceptions, every modern C compiler has the ability to adhere to this standard [103].

ANSI C programming language was used in this project because of the following reasons [103]:

(1) It is powerful and flexible. The language itself places no constraints on the user.

(2) It is a portable language. A C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification.

(3) It is a language of few words. It contains a handful of terms, called keywords, which serve as the base on which the language's functionality is built.

(4) It is modular. C code can be written in routines called functions. These functions can be reused in other applications or programs.

4.7 CAM FOR SUBTRACTIVE PROTOTYPING

Five CAM programs were created by the author using ANSI C programming language for the subtractive prototyping process. With the programs available, the data processing procedures are computerised, path generation process is automated and so on. As a result, computer-aided manufacturing in subtractive prototyping (ball nosed end milling) can be materialised. The programs are listed sequential as below.

(1) *Extract.c.* It is for extracting all the surface co-ordinates from the DXF entities files. The DXF entities files were transferred from the section cut models.

(2) *Convert.c.* It is for converting all the data from Cartesian co-ordinate system to Cylindrical co-ordinate system to suit the precision robotic manipulator.

(3) *Sortadd.c.* It is for sorting all the co-ordinates according to the height of the model, angles and creating the first and last point for each machining section (layer).

(4) *Vd.c.* It is for converting all the distance from millimetre and degree to motor step. It is also calculating all the synchronised velocity for each motion axes.

(5) *VsvMt.c.* It is the ultimate motion control program. Its functions include machining time estimation, command construction, communication with the PC-23 indexer and so on.

Figure 4.7(a) illustrates the programs sequence that a data file has to go through in order to produce a three-dimensional polystyrene model. Further details of the programs will be explained in the following sections.

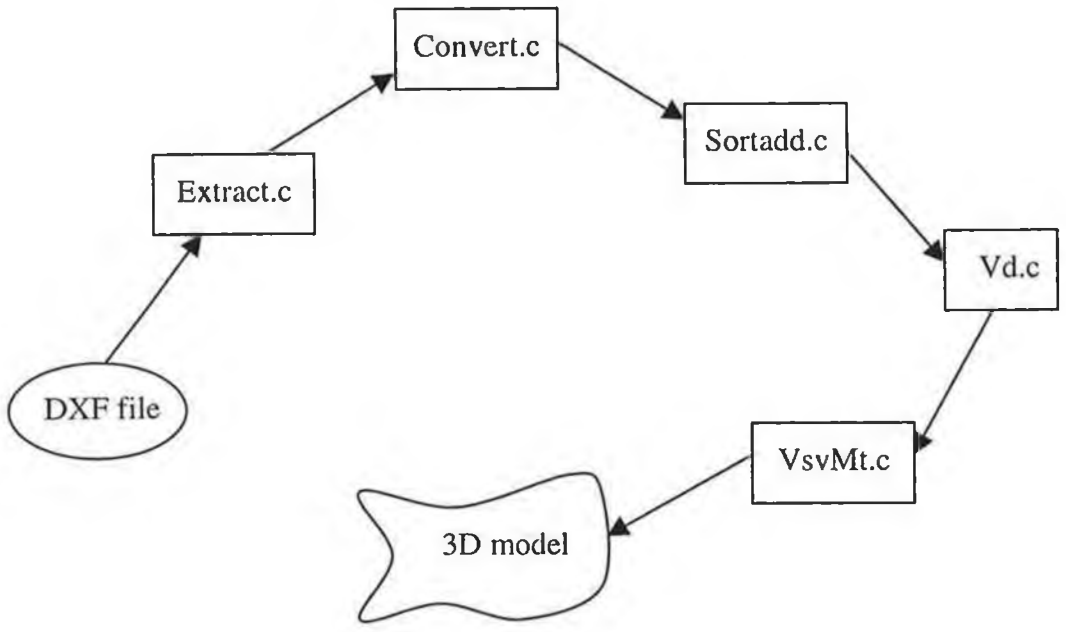


Figure 4.7(a) Subtractive Prototyping CAM Programs Flow

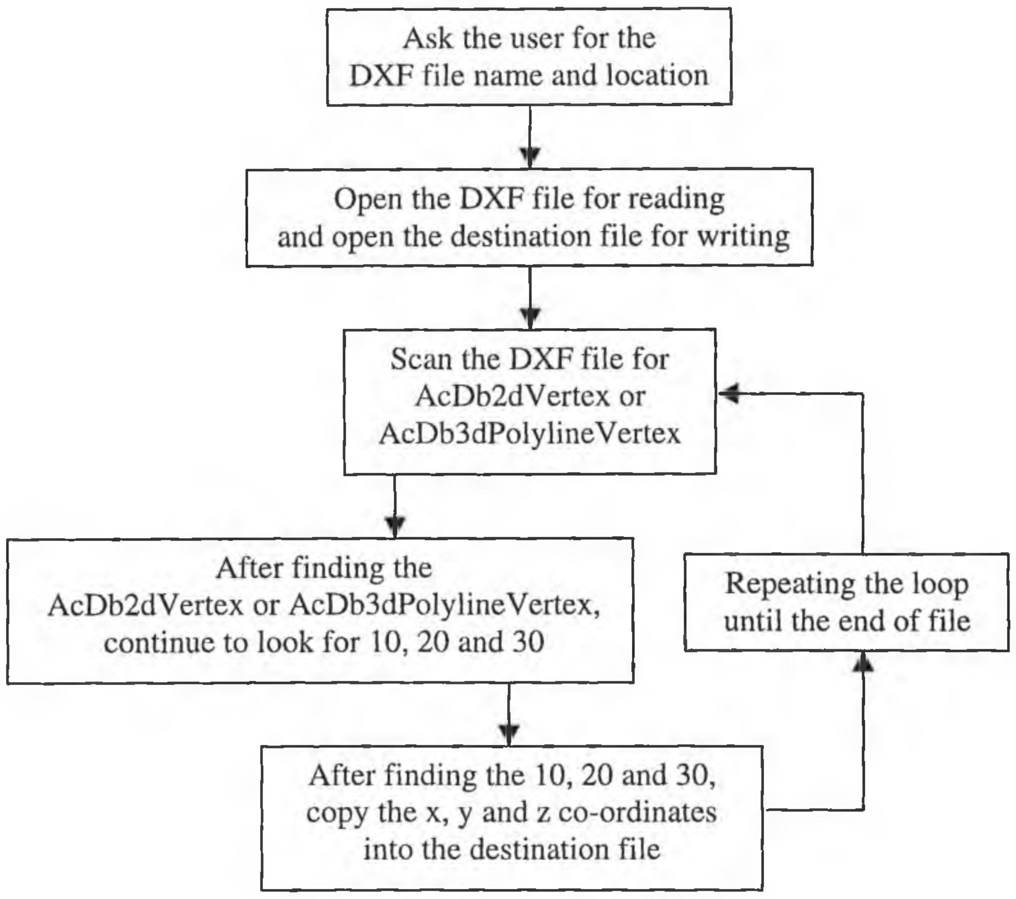
4.7.1 Extract.c

The first CAM program is the Extract.c. It is for extracting the x, y and z co-ordinates from the DXF files. Part of a DXF file of a section cut model is shown as below (comments in brackets).

```
0
VERTEX                  (vertex section begin)
  5
217                     (handle)
100                     (subclass marker)
AcDbEntity
  8
0                       (name of the layer)
100                     (subclass marker)
AcDbVertex
100                     (subclass marker)
AcDb2dVertex
```

```
 10
89.35717          (x co-ordinate of the vertex)
 20
116.386332        (y co-ordinate of the vertex)
 30
0.0               (z co-ordinate of the vertex)
```

The program flow is illustrated in Figure 4.7(b).



Ask the user for the DXF file name and location
Open the DXF file for reading and open the destination file for writing
Scan the DXF file for AcDb2dVertex or AcDb3dPolylineVertex
After finding the AcDb2dVertex or AcDb3dPolylineVertex, continue to look for 10, 20 and 30
Repeating the loop until the end of file
After finding the 10, 20 and 30, copy the x, y and z co-ordinates into the destination file

Figure 4.7(b) Extract.c Program Flow

The Extract.c program will begin by asking the user to input the DXF filename with the correct path and file extension. DXF file will have '.dxf' as the file extension. Then, the program opens the source file (*.dxf) for reading and opens the destination file (*.txt) for writing. The program will scan through the source file in searching for the subclass markers of 'AcDb2dVertex' or 'AcDb3dPolylineVertex'.

Once the subclass markers are available, the program will continue to look for group codes 10, 20 and 30. The data item for group code 10 is the x co-ordinate of the vertex. The y co-ordinate of the vertex will be paired with group code 20, while group code 30 will pair with the z co-ordinate of the vertex. Once obtaining the vertex co-ordinates, the program will write them into the destination file.

The program will continue to read through the source file in searching for the vertex co-ordinates until the end of file. When the program finish, the destination file will contain all the necessary surface data of a section cut model. The surface data are arranged according to the height of the model (step over). The Extract.c program source codes are listed in Appendix D.

4.7.2 Convert.c

Convert.c is the second CAM program. Its function is to convert the data from Cartesian co-ordinate system to Cylindrical co-ordinate system. The source file for Convert.c is the product file of Extract.c. Part of a source file is listed as below.

x	y	z
89.357170	116.386330	0.000000
89.578201	116.536995	0.000000
89.801727	116.685623	0.000000
90.027626	116.832169	0.000000
90.255753	116.976593	0.000000

The left column of the source file has x co-ordinates. The middle column of the file has y co-ordinates. The third column of the file has of z co-ordinates.

Initially, the program will ask the user to specify the source file and location. Then, the program will open the source file for reading and open the destination file for writing when necessary. The program will convert the data row by row. The base centre point of each section cut model is (100, 100, 0). As a result, the reference point for calculation is x = 100 and y = 100.

Only x and y co-ordinates are needed for calculating the radius (r mm) and angle (Rx degree) from the horizontal axis. Figure 4.7(c) shows the conversion method from Cartesian co-ordinate system to Cylindrical co-ordinate system in a graphical form.

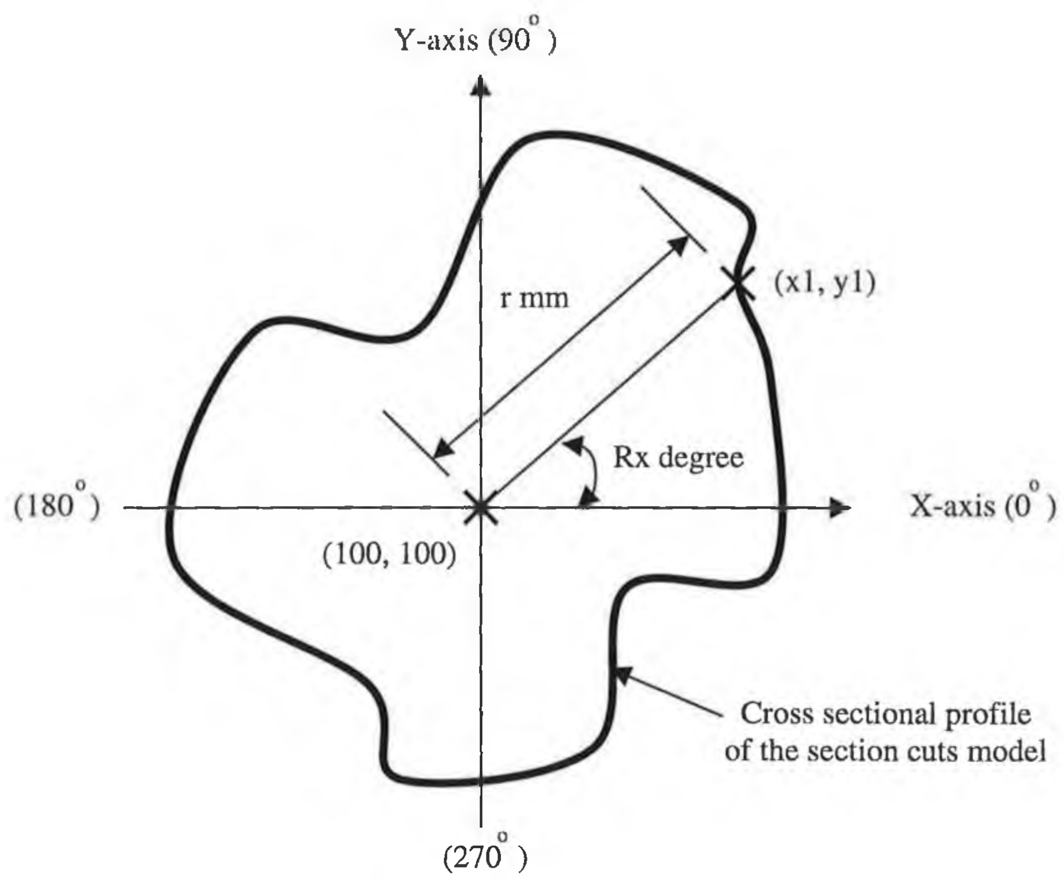


Figure 4.7(c) Cartesian System to Cylindrical System

By using (100, 100) and (x1, y1), the program can calculate r mm and Rx degree. The angle will start from the horizontal x-axis. The z value of the source file will be kept as it is in the destination file except that the locations of the data are rearranged.

The left column of the destination file consists of the height (z) of the model. The middle column of the file consists of the angle (Rx degree) values. The right column of the file consists of the radius (r mm) of the cross sectional profile. Part of a destination file is listed as below.

z	Rx	r
0.0	123.003510	19.539232
0.0	122.219589	19.547022
0.0	121.433334	19.555429
0.0	120.645020	19.564512
0.0	119.855026	19.574347

The Convert.c program source codes are listed in Appendix E.

4.7.3 Sortadd.c

Sortadd.c is the third CAM program. The source file of Sortadd.c will be the product file of the Convert.c. Soradd.c program functions are:

(1) Rearranging the data in each cross sectional profile (layer) so that the smallest angle ($\approx 0^o$) will be placed at the beginning of the layer and the largest angle ($\approx 360^o$) will be placed at the end.

(2) Calculating the radius (r mm) of each cross sectional profile (layer) at 0^o and 360^o. The points will be added as the first and last points of each layer.

Part of a source file is listed as below (comments in brackets).

z	Rx	r	
0.0	123.003510	19.539232	(beginning part of the first layer)
0.0	122.219589	19.547022	
0.0	121.433334	19.555429	
.	.	.	
0.0	1.295079	19.428715	(middle part of the same layer)
0.0	0.335376	19.427402	
0.0	359.377563	19.427805	
.	.	.	
0.0	125.297729	19.519014	(end part of the same layer)
0.0	124.532516	19.525387	
0.0	123.767746	19.532116	

The left column of the source file consists of the height (z or step over distance) of the model. The middle column consists of the angles (Rx). The third column consists of the radius (r) of the cross sectional profile of the model.

Once activated, the program will ask the user for the source filename and location. The program will then open the source file for reading and open the destination file for writing (appending) when necessary. The program will sort the data layer by layer. It will also calculate the radius of the profile at 0° and 360° by using linear interpolation.

Part of a destination file is listed as below (comments in brackets). The left column of the file consists of the z values of the model. The middle column consists of Rx degree. The third column consists of the r values.

z	Rx	r	
0.000000	0.000000	19.427544	(beginning part of the first layer)
0.000000	0.335376	19.427402	
0.000000	1.295079	19.428715	
.	.	.	
0.000000	179.815506	18.472027	(middle part of the same layer)
0.000000	180.804916	18.459417	
0.000000	181.798599	18.448023	
.	.	.	
0.000000	358.422180	19.429905	(end part of the same layer)
0.000000	359.377563	19.427805	
0.000000	360.000000	19.427544	

The Sortadd.c program source codes are listed in Appendix F.

4.7.4 Vd.c

Vd.c is the fourth CAM program. The source file for the Vd.c is the product file of the Sortadd.c program. Vd.c program has the following functions.

(1) Converting the height (step over distance), angle and radius values of the source file into motor steps based on the raw material block radius.

(2) Calculating the motor velocity based on the user's desired feed rate.

(3) Calculating the velocity of each motion axis so as to have a synchronous motion profile.

Once activated, the program will ask the user to provide the source filename, its location and the radius of the raw material block. Then, it will open the source file for reading and open the destination file for writing (appending). The data will be calculated row by row. The calculated motor steps are either positive (increment) or negative (decrement) values.

Part of the beginning, middle and the end of the first layer of a destination file is listed as below. The feed rate chosen by the author is 0.375 mm/sec (manipulation along x or y-axis).

Velocity along x-axis	Motor step along x-axis	Velocity around x-axis	Motor step around x-axis	Velocity along y-axis	Motor step along y-axis
0.2500	0	0.2500	18575	0.2500	0
0.2500	0	0.0003	0	0.2500	466
0.2500	0	0.0008	-4	0.2500	1333
.	.	.	.	.	.
0.2500	0	0.0084	46	0.2500	1368
0.2500	0	0.0076	42	0.2500	1374
0.2500	0	0.0069	38	0.2500	1380
.	.	.	.	.	.
0.2500	0	0.0024	13	0.2500	1323
0.2500	0	0.0013	7	0.2500	1327
0.2500	0	0.0003	1	0.2500	864

The first column from the left of the product file consists of the velocities (revolution/sec) of the motion axis along x-axis (refer section 3.2.3). The second column from the left of the file contains all the motor steps of the motion axis along x-axis. The third column from the left of the product file are the velocities of the motion axis around x-axis (refer section 3.2.1). The fourth column from the left of the file contains all the motor steps for the motion axis around x-axis. The fifth column from the left of the product file are the velocities of the motion axis along y-axis (refer section 3.2.2). The final column contains all the motor steps for the motion axis along y-axis. The Vd.c program source codes are listed in Appendix G.

<u>4.7.5 VsvMt.c</u>

VsvMt.c is the last CAM program of the subtractive prototyping programs. The needed source file is the product file of Vd.c. VsvMt.c program functions are:

(1) Calculating the total production time (program execution time and actual machining time) of the milling process.

(2) Calculating the individual machining time and the motion commands execution time.

(3) Calculating the remaining production time of the milling process.

(4) Initialising the PC-23 indexer and set the motor resolution of the indexer to match the KS-drives' motor resolution.

(5) Constructing motion commands and sending them to the PC-23 indexer.

(6) Communicating with the PC-23 indexer so as to write the commands to the indexer and read the responses from the indexer.

Once activated, the program will ask the user for the source file and location. Then, it will open the source file for reading. The following task is to calculate the overall production time of the model. Later, the PC-23 indexer is initialised (refer section 3.3.2 for the procedures of resetting the indexer). The motor resolution of the indexer is set by the program to match the KS-drives' motor resolution too.

The following step of the program is to calculate the production time of a single command and display it together with the remaining production time. Then, the program will construct the motion command based on the data of the source file. The complete command will then be sent over to the PC-23 indexer. The command will be written to the PC-23 indexer one character at a time (refer section 3.3.2 for the procedures of sending a command string to the indexer).

When all the motion axes stop, the program will decode the response (refer section 3.3.2 for the procedures of receiving a character string from indexer) of the indexer and display the incremental (or decrement) motor steps of each axis. An example of commands is listed as below.

```
1VS0.25 1V0.25 1D0 1I 2VS0.0084 2V0.084 2D46 2I 3VS0.25 3V0.25 3D1368 3I G123
```

VS command will let the motion axis start and stop at the specified velocity. V command will set the velocity of the motion axis during the move. The motion will be more accurate with VS and V are having same value. D command is for setting the motor steps to be moved. I command allows the indexer to pre-calculate the move data. G123 command will synchronise axis 1, 2 and 3 to start moving together. 1 or 2 or 3 is prefixing the VS, V, D and I commands for specifying the motion axis (refer section 3.3.4 and Appendix A for further detail).

The program flow of the VsvMt.c is shown in Figure 4.7(d). The program source codes of VsvMt.c are listed in Appendix H.

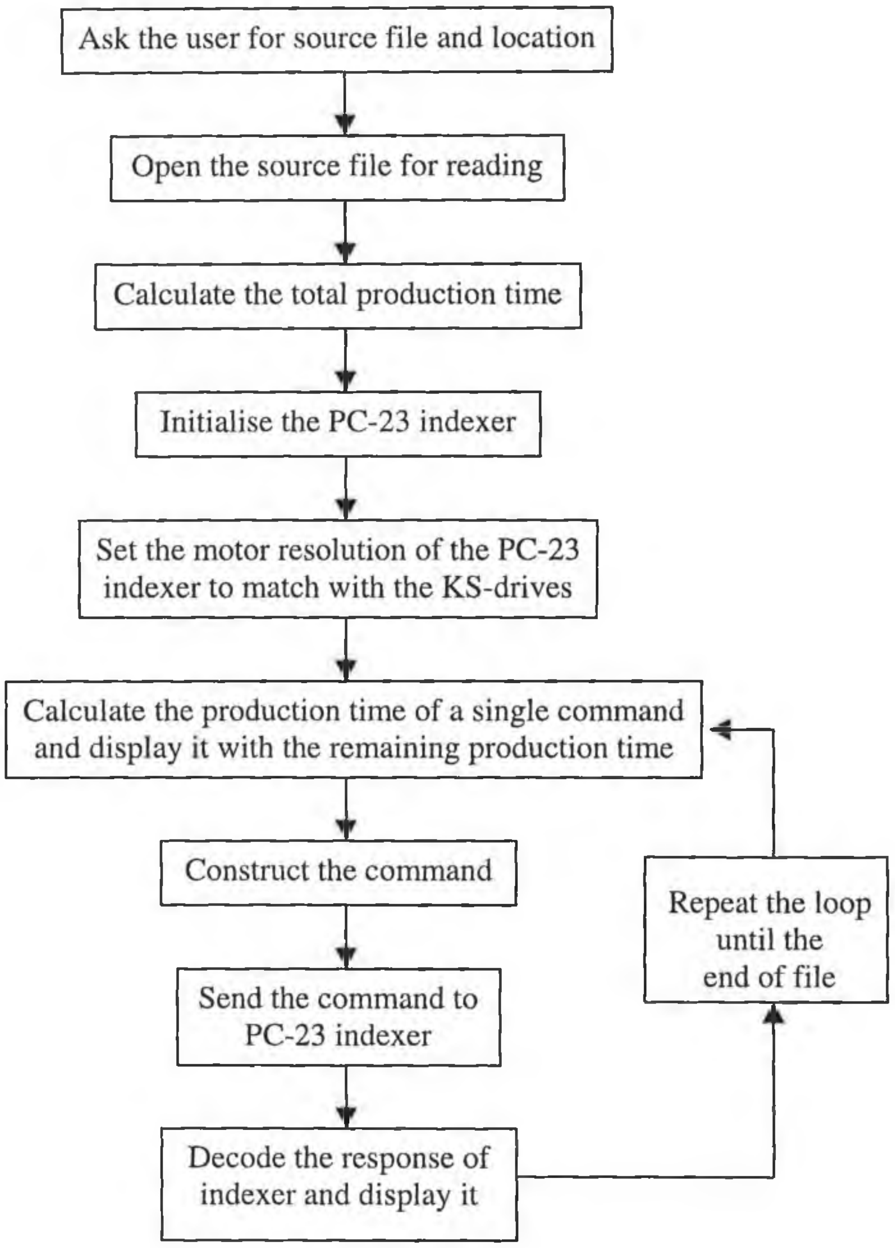

Ask the user for source file and location
Open the source file for reading
Calculate the total production time
Initialise the PC-23 indexer
Set the motor resolution of the PC-23 indexer to match with the KS-drives
Calculate the production time of a single command and display it with the remaining production time
Construct the command
Send the command to PC-23 indexer
Decode the response of indexer and display it
Repeat the loop until the end of file

Figure 4.7(d) VsvMt.c Program Flow

4.8 CAM FOR ADDITIVE PROTOTYPING

Seven CAM programs have been created by the author using ANSI C programming language to facilitate the additive prototyping process. The programs are listed sequentially as below:

(1) *ExWax.c.* It is for extracting all the surface co-ordinates from the DXF entities files. The DXF entities files were transferred from the section cut models.

(2) *Group.c.* It is for regrouping the data according to the height of the models.

(3) *ConWax.c.* It is for converting the data from Cartesian co-ordinate system to Cylindrical co-ordinate system.

(4) *SaWax.c.* It is for sorting the data according to the angles in each cross sectional profile (layer) and creating the first and last point of each layer.

(5) *Select.c.* It is for selecting the data that match with the user-specified radius.

(6) *Deposit.c.* It is the final control program that controls the motion axes, communicating with the PC-23 indexer, constructing motion commands and so on.

(7) *DpsSlp.c.* It is almost same as the Deposit.c. The only difference is the program did not control the additive prototyping equipment as in Deposit.c.

Figure 4.7(e) illustrates the programs sequence that a data file has to go through in order to produce a three-dimensional polystyrene model with internal cavities. ExWax.c, ConWax.c and SaWax.c programs are almost same as Extract.c, Convert.c and Sortadd.c programs respectively. As a result, only Group.c, Select.c, Deposit.c and DpsSlp.c programs will be elaborated further in the following sections. Appendix I, K and L listed the program source codes for ExWax.c, ConWax.c and SaWax.c respectively.

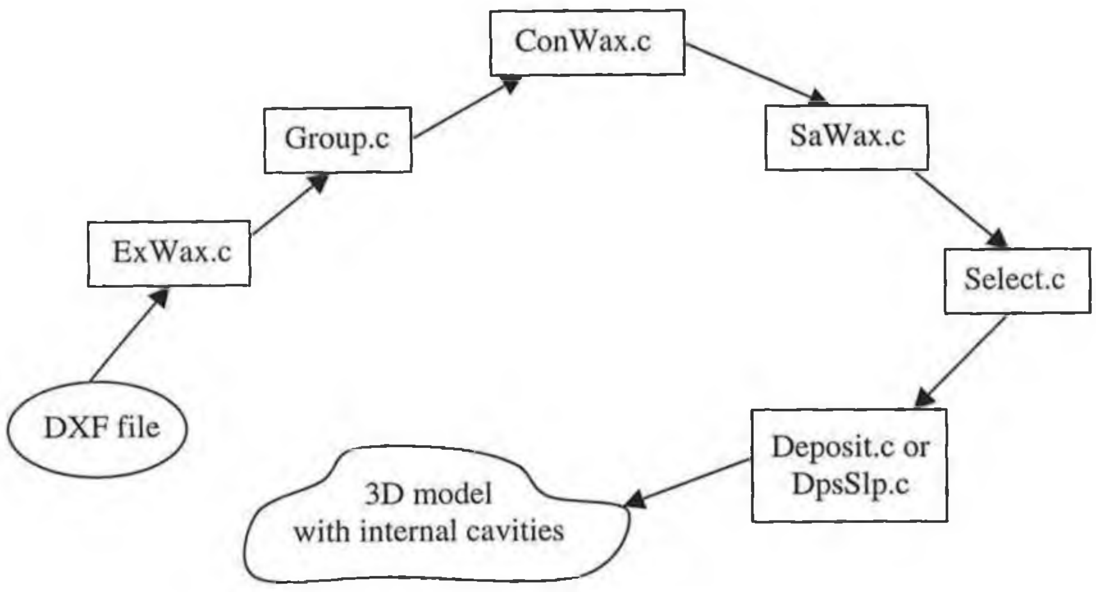


Figure 4.7(e) Additive Prototyping CAM Programs Flow

4.8.1 Group.c and Select.c

Group.c is the second program of the additive prototyping CAM programs. Group.c is a program that rearranges the data of the source file according to the height of the models. The source file is the product file of the ExWax.c program. The data file of a three-dimensional model with internal cavities is different from the data file of a surface model in the subtractive prototyping process. The co-ordinates of the data file in the additive prototyping process are not arranged according to the height of the models. As a result, Group.c is used to produce the desired data file for the following data processing stages.

The program has two assumptions. The assumptions are:

(1) The maximum height of the model is 99 mm.

(2) The step over distance is 3 mm.

Group.c will produce the data file for ConWax.c. ConWax.c will change the data file so that SaWax.c can utilise it. SaWax.c will in turn provide the data file for Select.c.

Select.c is the fifth program of the additive prototyping CAM programs. Select.c is a program that selects the data based on the user-specified maximum radius. The user will need to know the maximum radius of the model. The radius values of the source file can be checked by opening the SaWax.c program's product file. The radius values are stored in the third column from the left of the file. Select.c will produce the source file for Deposit.c or DpsSlp.c so that a three-dimensional model with internal cavities can be produced by using additive prototyping equipment. The program source codes for Group.c and Select.c are listed in Appendix J and M respectively.

4.8.2 Deposit.c and DpsSlp.c

Deposit.c and DpsSlp.c are the last programs of the additive prototyping CAM programs. Both programs have slight differences in their basic functions due to the difference in assumptions.

The functions of the Deposit.c program are:

(1) Calculating the incremental motor steps to be travelled by the manipulation units along and around x-axis.

(2) Reset the PC23 indexer. Assuring that the indexer is ready to accept commands from the author program.

(3) Constructing and sending command string of all motion axes to the PC23 indexer, getting position response string and display it on the screen.

(4) The model will be transferred back to its original position once it reached 99 mm in height (total step over distance).

(5) Communicating with the PC-23 indexer so as to write the commands to the indexer and read the response from the indexer.

The Deposit.c program has a few assumptions. The assumptions are:

(1) The motor of the manipulation unit along y-axis is used to push the semi-liquid material or wax onto the core cylinder block. The rotational motion of the motor will be converted to linear motion of the shaft which push the wax or semi-liquid material. Thus, the motor step is directly proportional to the quantity of the deposition material.

(2) There is no stoppage or delay time between all the motion axes. Stoppage time will have to be calculated once the actual processing equipment is ready.

(3) The total step over distance or the height of the model is 99 mm

Once the Deposit.c is activated, it will ask for the filename and location. It will also ask the number of motor steps required for pushing one drop of semi-liquid material or wax from the deposition tool. The user will need to calculate the motor steps by referring to the additive prototyping equipment. Then, it will open and read the source file.

The program will set the motor resolution of the PC-23 indexer to match with the KS-drive setting. The program will calculate the required motor step to be travelled by all the motion axes. Then, the commands for all the axes will be constructed and send over to the PC-23 indexer for motion control. The response will be decoded and displayed on the personal computer screen. Figure 4.7(f) shows the Deposit.c program flow.

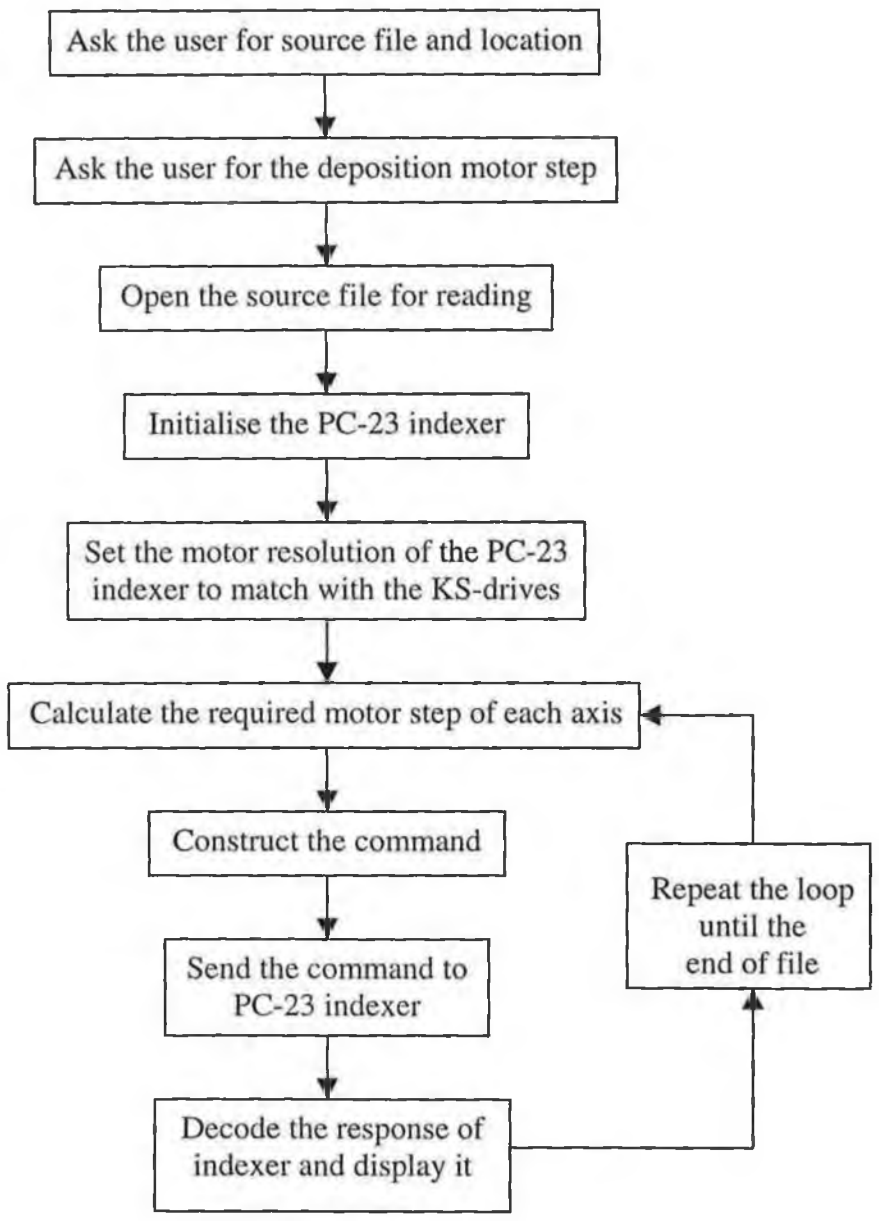

Ask the user for source file and location
Ask the user for the deposition motor step
Open the source file for reading
Initialise the PC-23 indexer
Set the motor resolution of the PC-23 indexer to match with the KS-drives
Calculate the required motor step of each axis
Construct the command
Send the command to PC-23 indexer
Decode the response of indexer and display it
Repeat the loop until the end of file

Figure 4.7(f) Deposit.c Program Flow

The format of the motion commands in Deposit.c is almost same as VsvMt.c program. The only difference is that Deposit.c uses 1G or 2G or 3G to activate the motion instead of G123 command in VsvMt.c program. As a result, Deposit.c created the step by step motion. The motion axes will move one after another instead of start and stop moving at the same time. Appendix N shows the program source codes of Deposit.c

The DpsSlp.c program is having the same functions as the Deposit.c except that it did not calculate the motor step to be moved by the deposition equipment. The assumptions of the programs are almost same as the Deposit.c program except that the DpsSlp.c assumes that the deposition equipment is controlled by another external controller. The motion axes of the robotic manipulator will have one-second stoppage time to accommodate the deposition operation. The format of the motion commands of DpsSlp.c is the same as Deposit.c program. Figure 4.7(g) shows the program flow of DpsSlp.c program. Its' program source codes are listed in Appendix O.

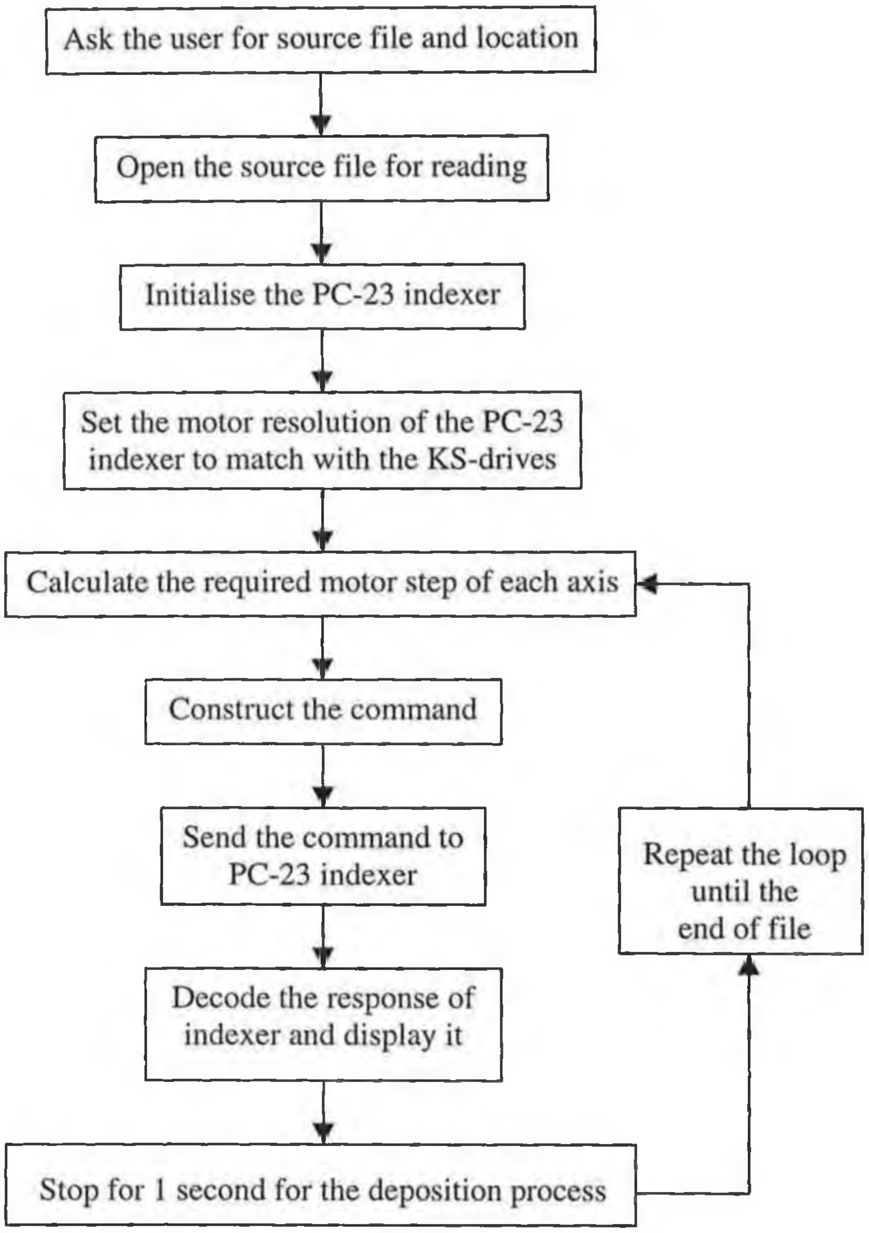

Ask the user for source file and location
Open the source file for reading
Initialise the PC-23 indexer
Set the motor resolution of the PC-23 indexer to match with the KS-drives
Calculate the required motor step of each axis
Construct the command
Send the command to PC-23 indexer
Decode the response of indexer and display it
Stop for 1 second for the deposition process
Repeat the loop until the end of file

Figure 4.7(g) DpsSlp.c Program Flow

CHAPTER 5: RESULTS AND DISCUSSION

5.1 INTRODUCTION

A rapid prototyping system using a precision robotic manipulator was built following the descriptions in the previous chapters.

A latest model personal computer was connected electronically to the PC-23 indexer. The PC-23 indexer was in turn connecting and controlling the KS-drives. The KS-drives were connected to the a. c. brushless servomotors. The motors were either connected mechanically to the lead screws or gearboxes of the manipulator to generate motions. A subtractive prototyping equipment was also built to simulate the ball nosed end milling process.

The AutoSurf software loaded onto the personal computer was used to produce the desired three-dimensional ruled surface models for subtractive prototyping process. The surface models were section cut into multiple cross sectional layers by using AutoSurf. The AutoSurf was also used for producing solid models with internal cavities for the proposed additive prototyping process and the graphic simulation program.

The graphic files of the section cut models were converted into DXF entities files. Then, the DXF files were processed into machining data files by the CAM programs that were developed by the author. The data files were then fed into the final control program for controlling the precision manipulator to produce the three-dimensional ruled surface models. Polystyrene cylindrical blocks were used as the model material of the milling process. The milled models were based on the ruled surface models in section 4.3.1.

CAM programs were also developed for the additive prototyping process. The final control programs of the additive process were catered for two situations. The first situation is – the wax or semi-liquid deposition equipment is controlled by the same system. The second situation is – the deposition equipment is controlled by an external system. The additive prototyping process was simulated graphically by using the simulation program which was built by the author using AutoLISP – the AutoSurf programming language.

The actual polystyrene products are shown in the following section. The results of the graphic simulation of the additive prototyping process are also shown in the later sections. The last few sections of this chapter will be devoted to discussing:

(1) The effectiveness of the personal computer, interfacing system and the manipulator.

(2) The effectiveness of the CAD/CAM programs.

(3) The effects of the model shapes, size and feed rate.

(4) The effectiveness of the graphic simulation programming language.

5.2 RESULTS OF THE SUBTRACTIVE PROTOTYPING PROCESS

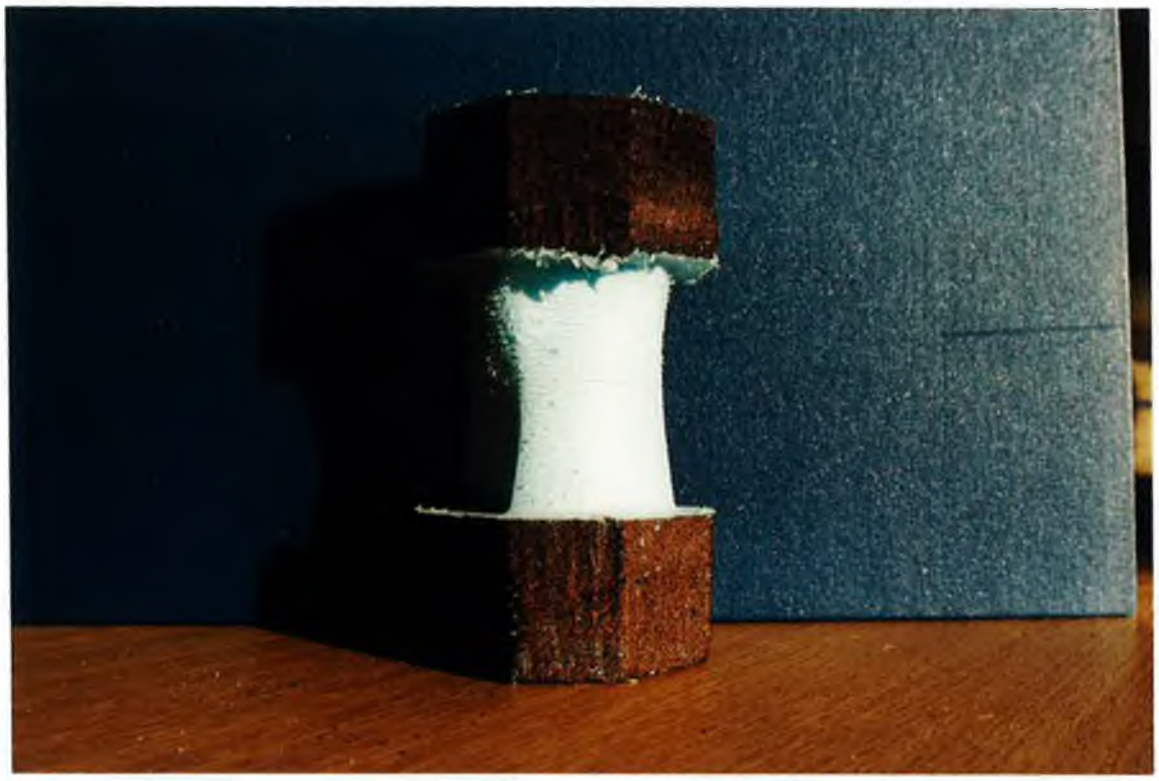


Plate 5.2(a) Result for Heart to Complex Shaped Model

The milled model in Plate 5.2(a) was based on the section cut surface model from Figure 4.3(g). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.69 hours

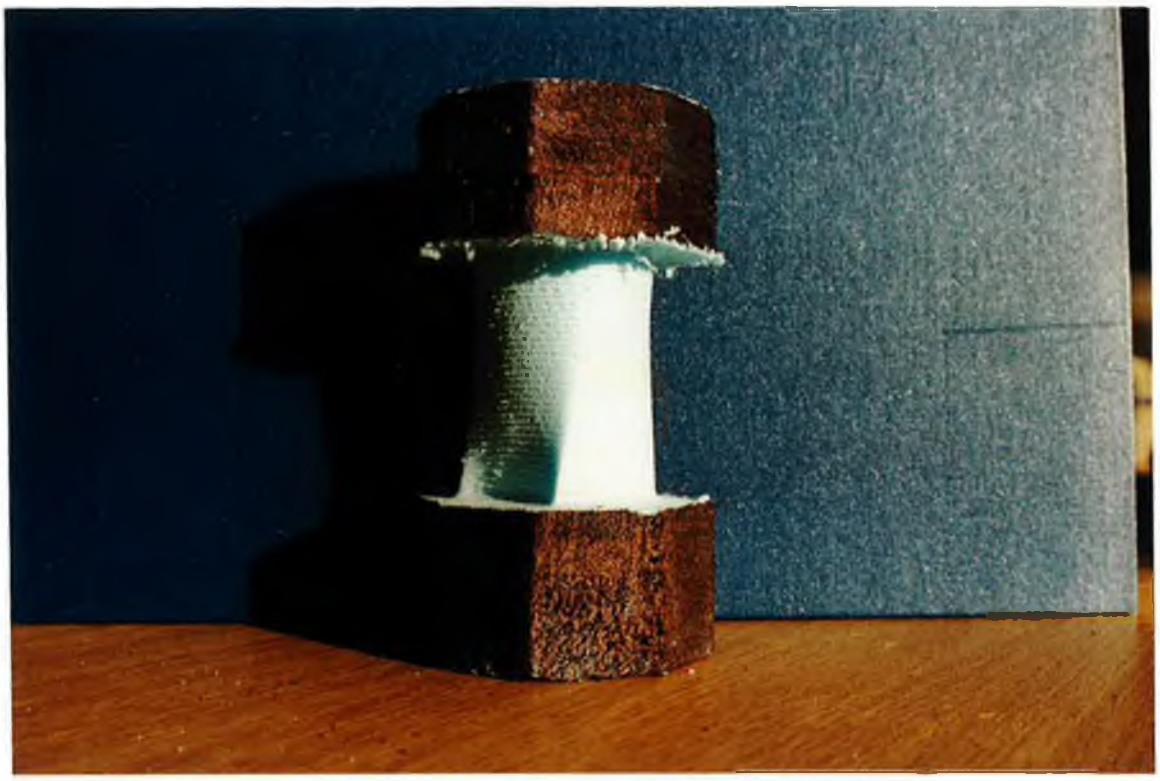


Plate 5.2(b) Result for Circle to Heart Shaped Model

The milled model in Plate 5.2(b) was based on the section cut surface model from Figure 4.3(i). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9^{o} per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.69 hours

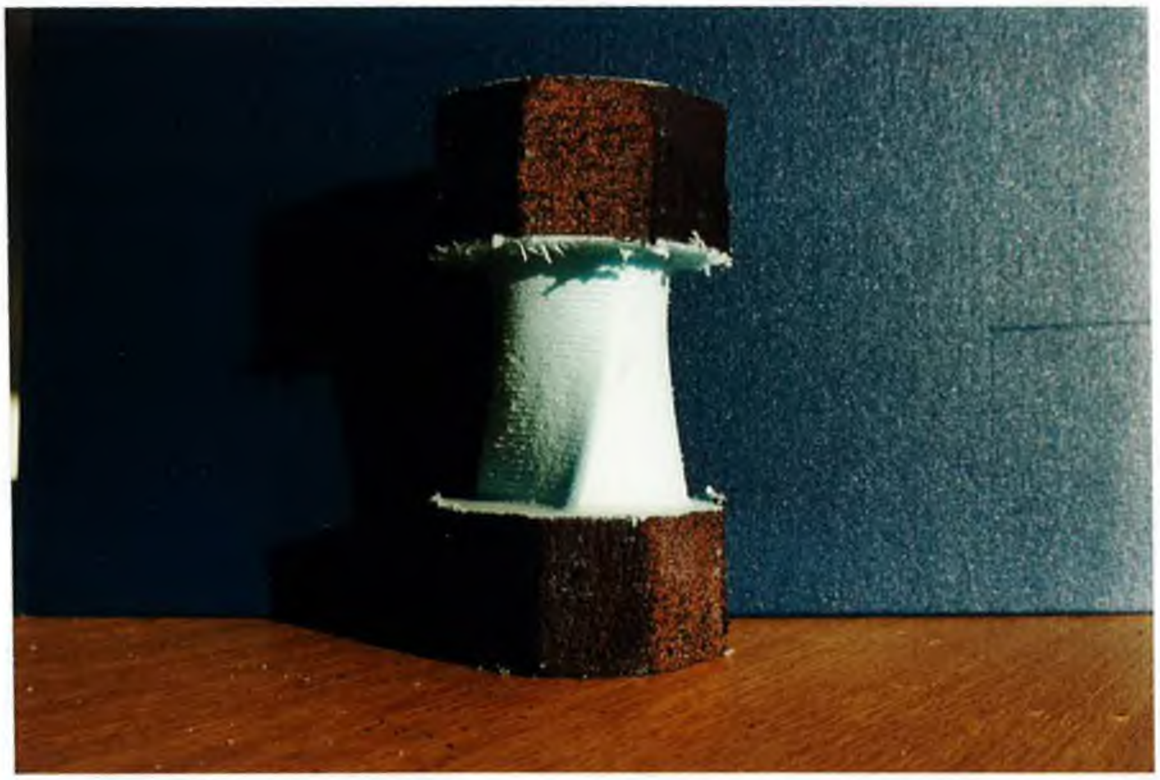


Plate 5.2(c) Result for Circle to Complex Shaped Model

The milled model in Plate 5.2(c) was based on the section cut surface model from Figure 4.3(j). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.63 hours

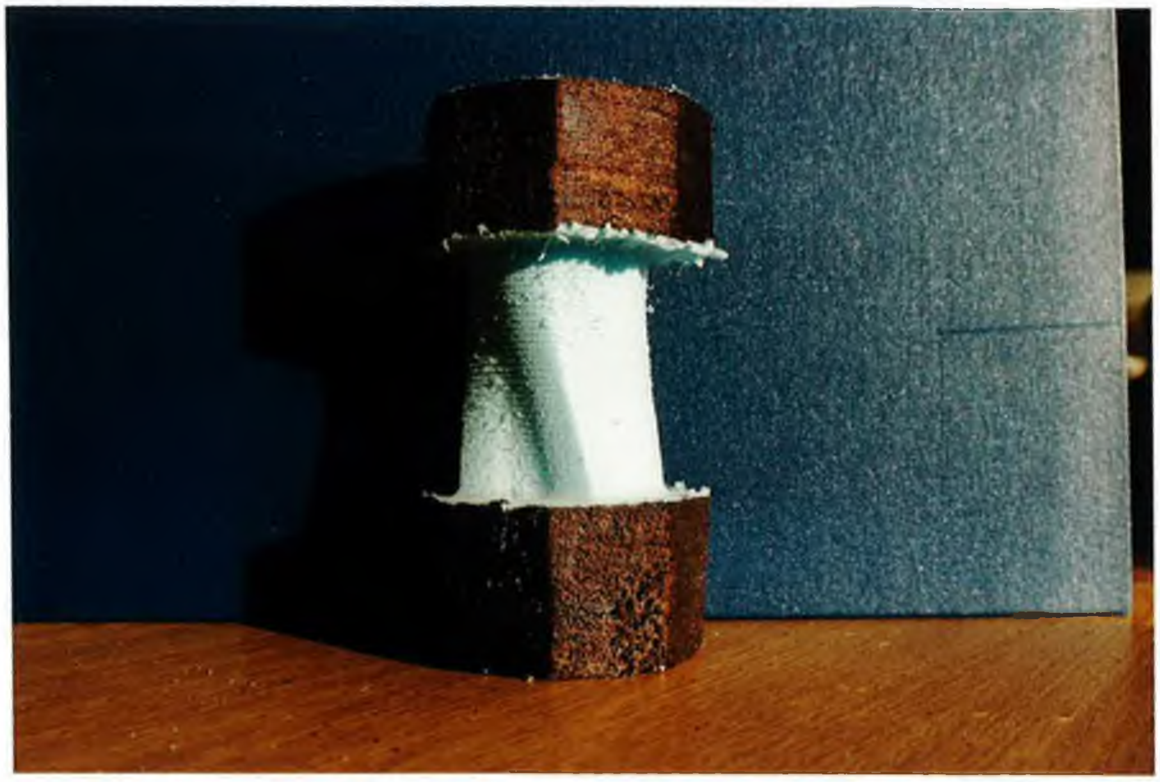


Plate 5.2(d) Result for Circle to Star Shaped Model

The milled model in Plate 5.2(d) was based on the section cut surface model from Figure 4.3(k). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.82 hours

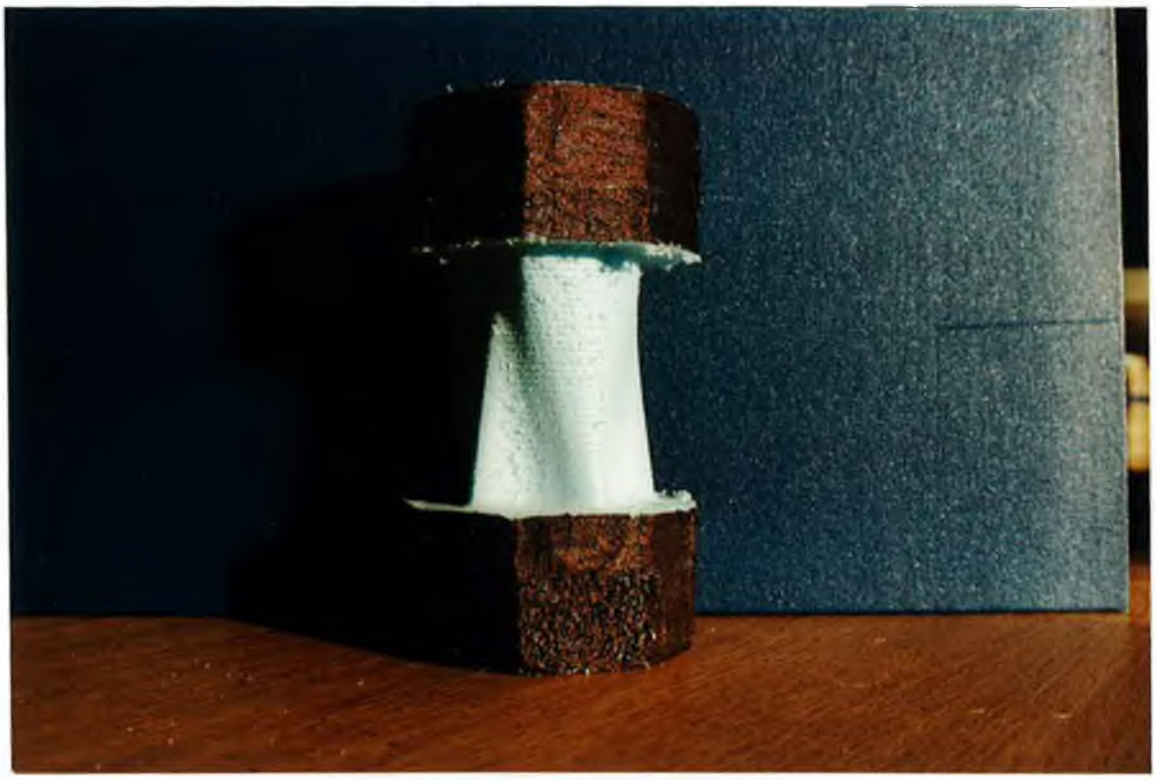


Plate 5.2(e) Result for Heart to Star Shaped Model

The milled model in Plate 5.2(e) was based on the section cut surface model from Figure 4.3(l). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis : 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.89 hours

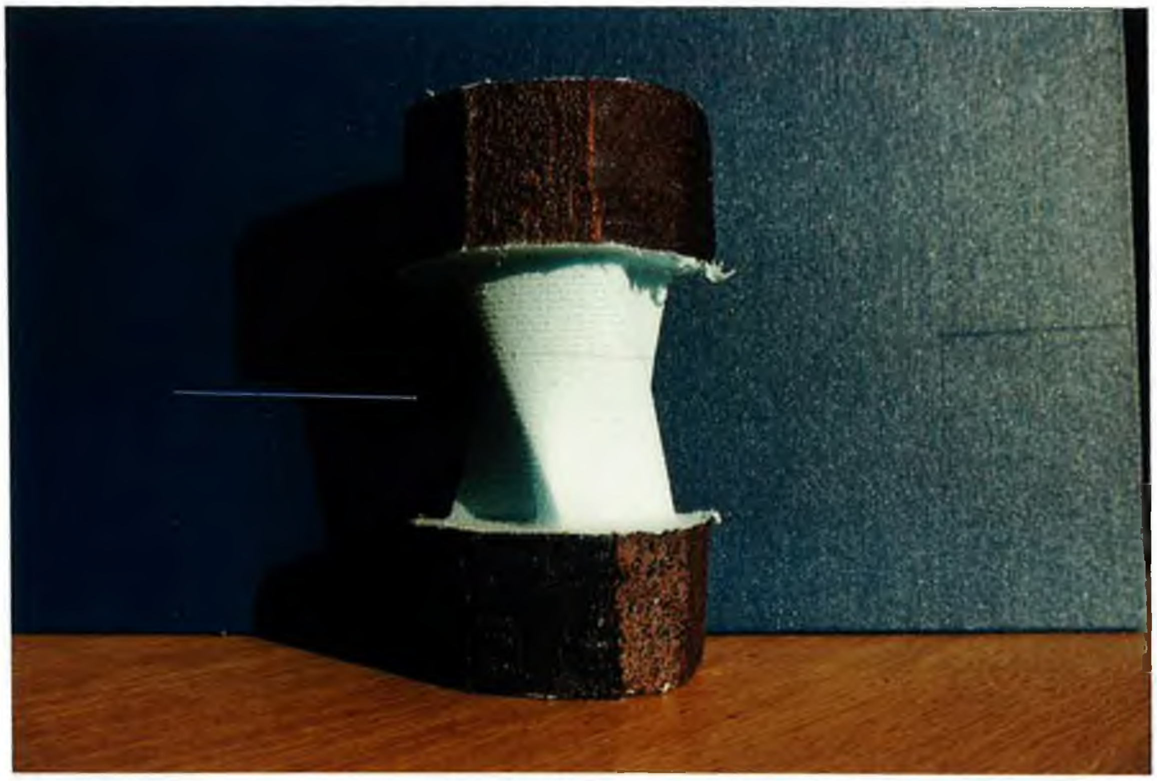


Plate 5.2(f) Result for Complex to Star Shaped Model

The milled model in Plate 5.2(f) was based on the section cut surface model from Figure 4.3(m). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.90 hours

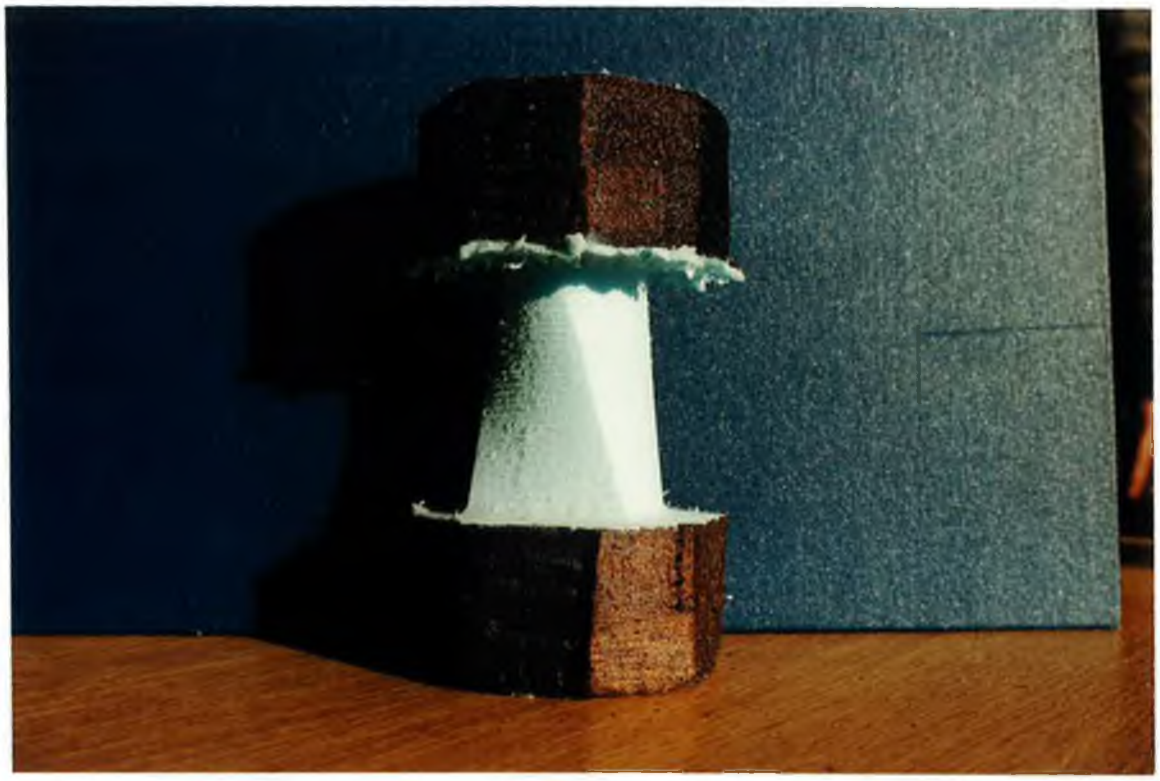


Plate 5.2(g) Result for Circle to Square Shaped Model

The milled model in Plate 5.2(g) was based on the section cut surface model from Figure 4.3(n). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.39 hours

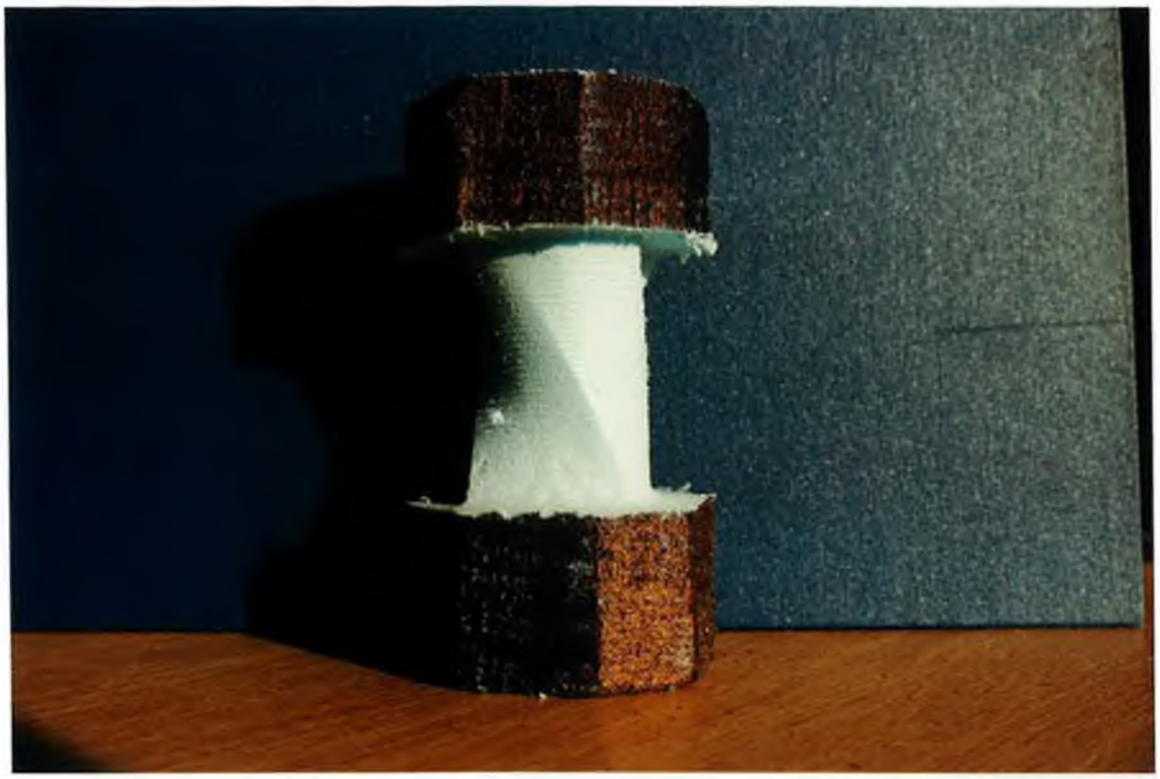


Plate 5.2(h) Result for Complex to Square Shaped Model

The milled model in Plate 5.2(h) was based on the section cut surface model from Figure 4.3(o). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.57 hours

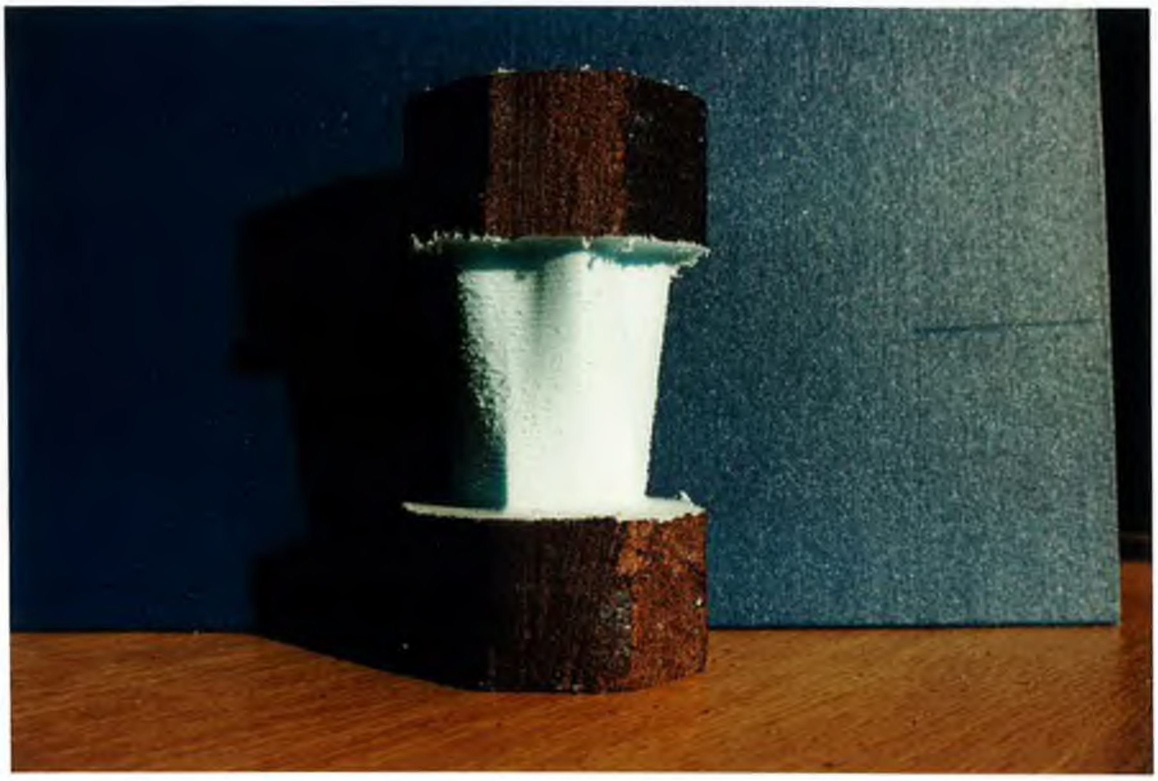


Plate 5.2(i) Result for Star to Pentagon Shaped Model

The milled model in Plate 5.2(i) was based on the section cut surface model from Figure 4.3(p). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.81 hours
Production time	: 4.83 hours

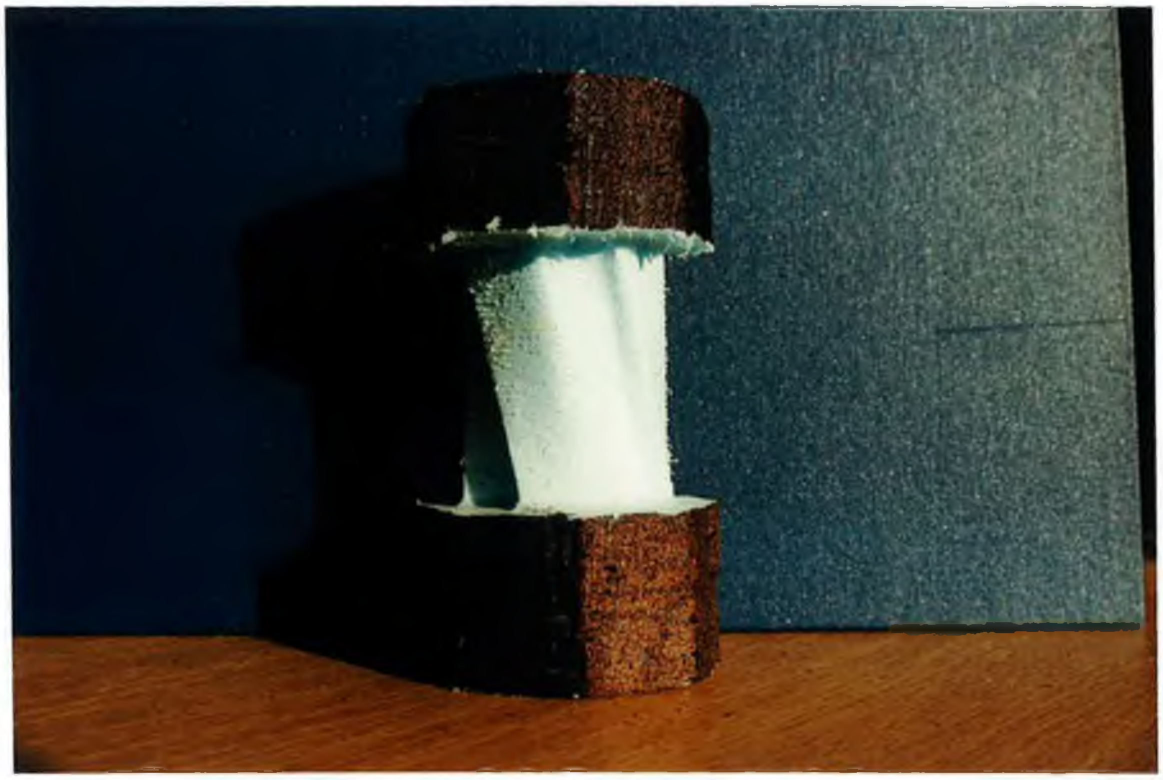


Plate 5.2(j) Result for Cross to 45^{o} Rotated Cross Shaped Model

The milled model in Plate 5.2(j) was based on the section cut surface model from Figure 4.3(q). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9^{o} per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 5.03 hours

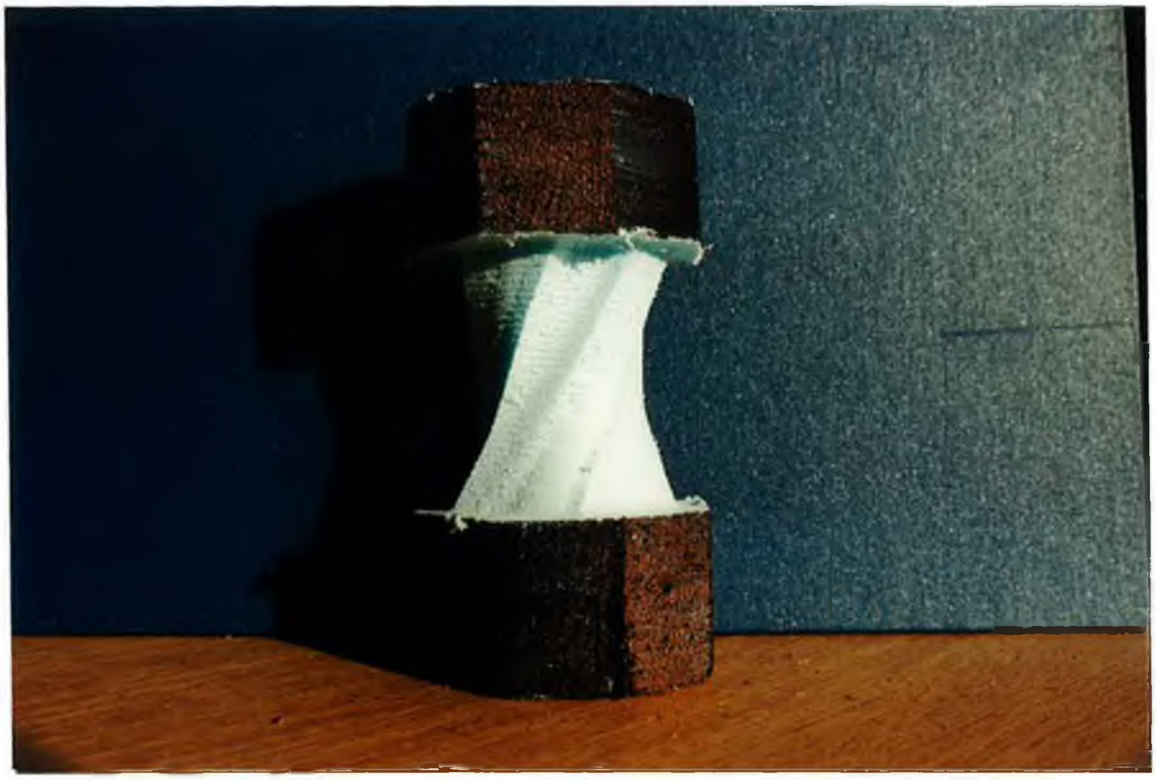


Plate 5.2(k) Result for Cross to Pentagon Shaped Model

The milled model in Plate 5.2(k) was based on the section cut surface model from Figure 4.3(r). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9^{o} per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.66 hours

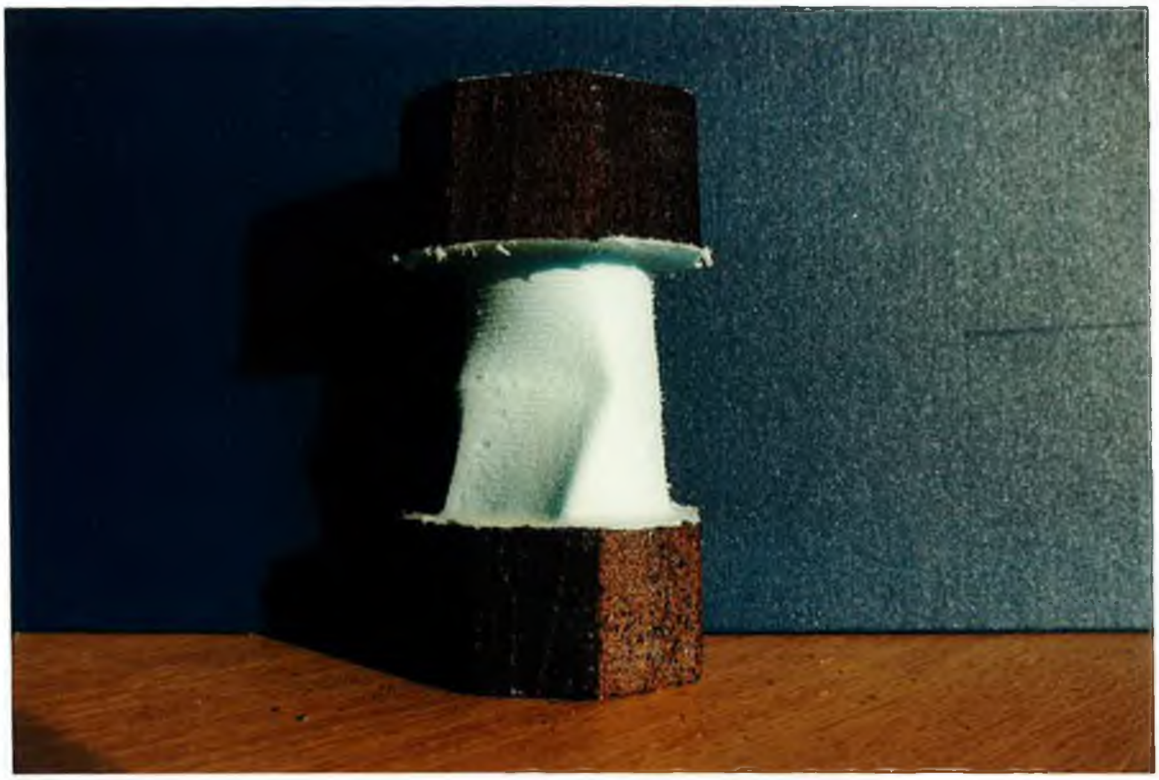


Plate 5.2(l) Result for Circle to Heart to Complex Shaped Model

The milled model in Plate 5.2(l) was based on the section cut surface model from Figure 4.3(s). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.65 hours

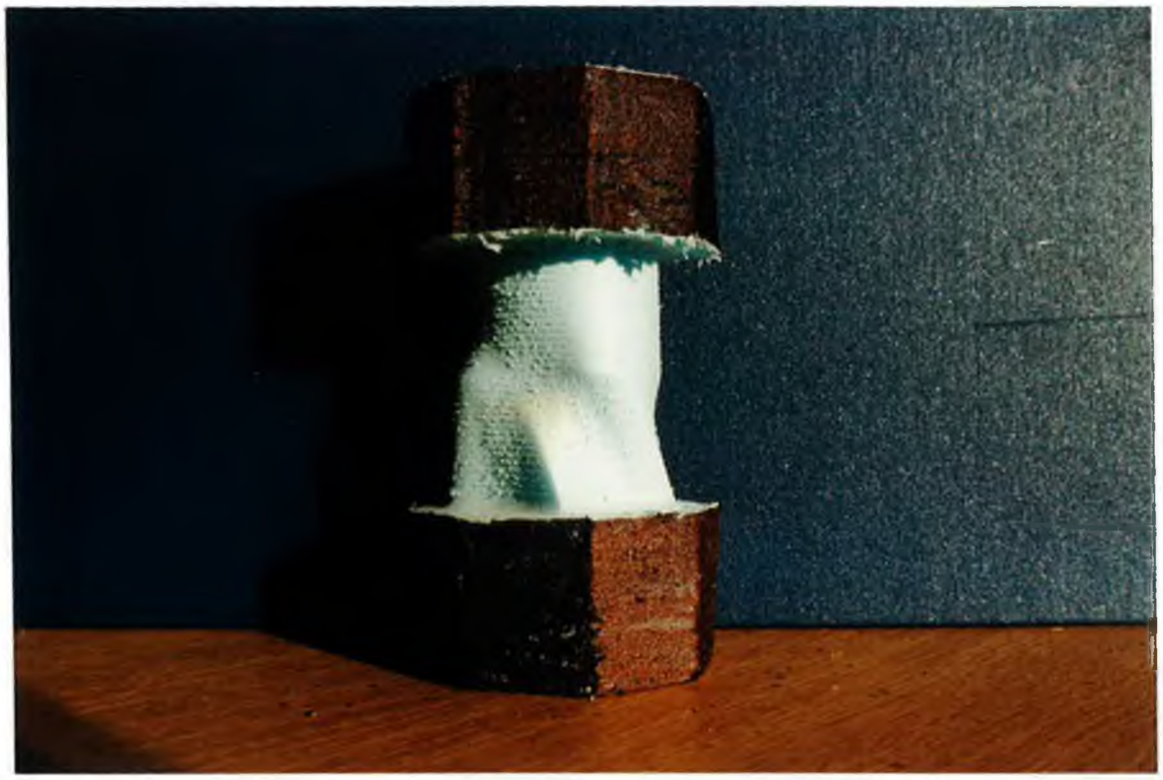


Plate 5.2(m) Result for Circle to Heart to Star Shaped Model

The milled model in Plate 5.2(m) was based on the section cut surface model from Figure 4.3(t). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.75 hours

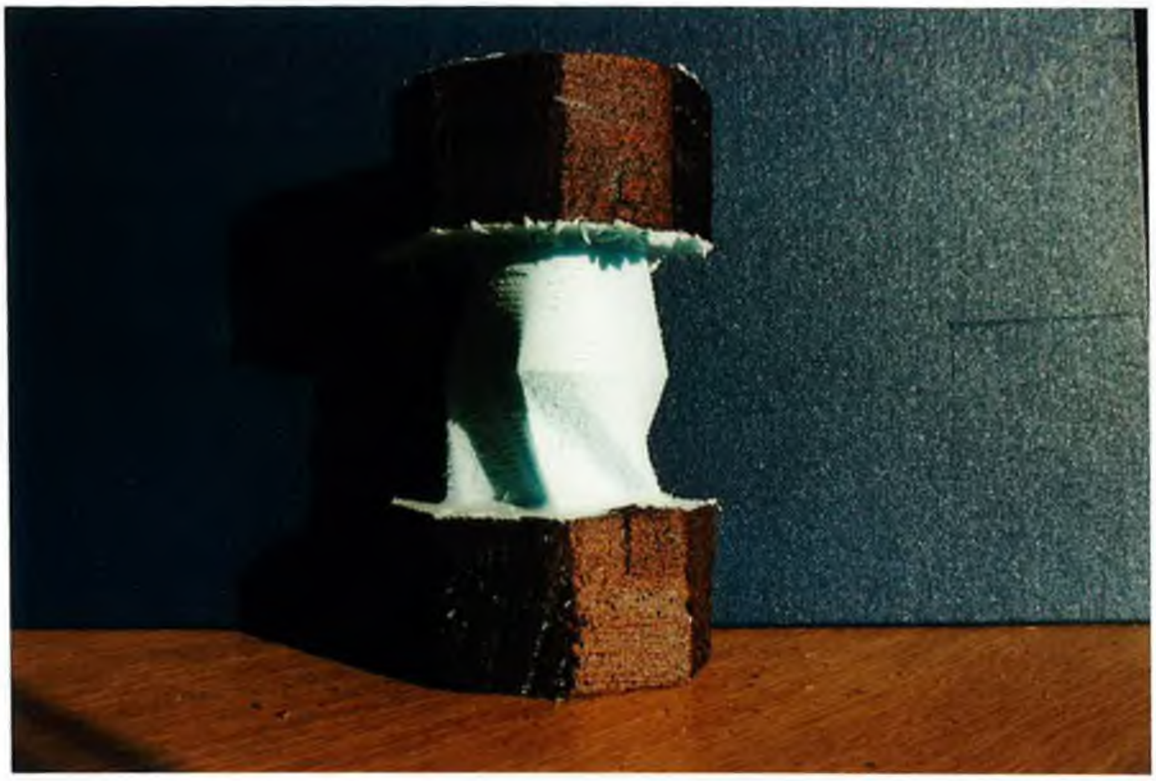


Plate 5.2(n) Result for Circle to Complex to Star Shaped Model

The milled model in Plate 5.2(n) was based on the section cut surface model from Figure 4.3(u). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.83 hours

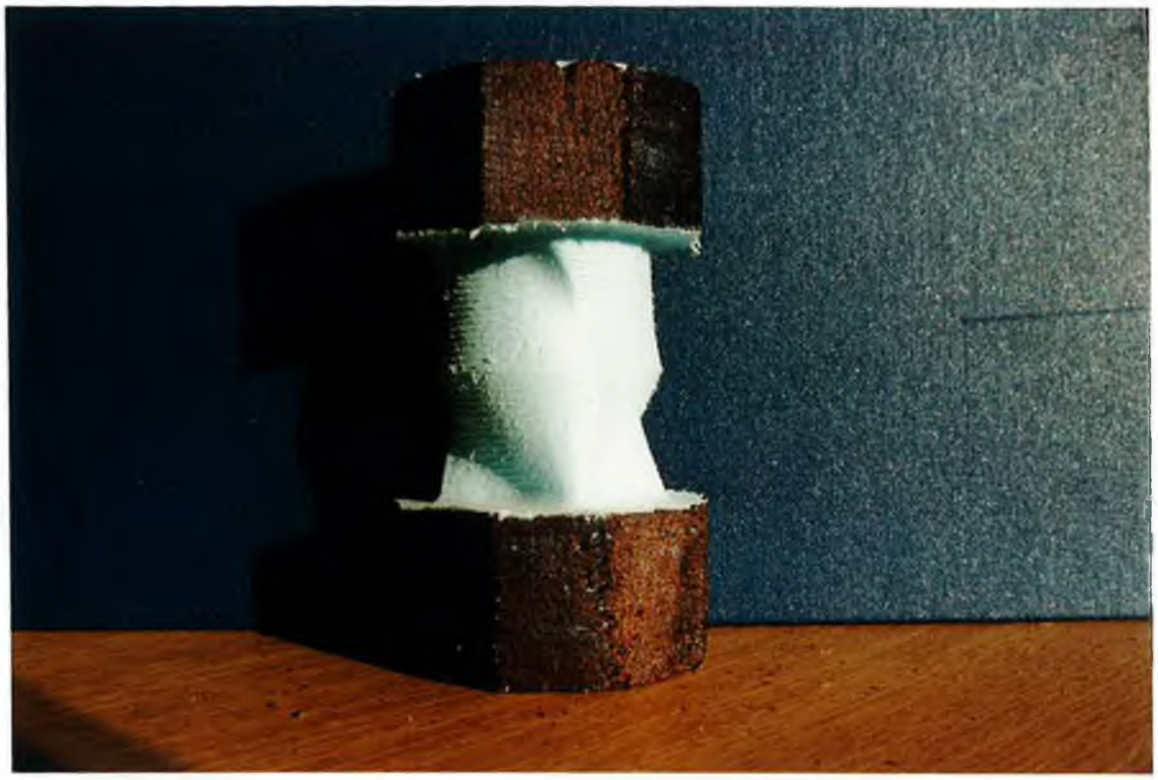


Plate 5.2(o) Result for Heart to Complex to Star Shaped Model

The milled model in Plate 5.2(o) was based on the section cut surface model from Figure 4.3(v). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9^{o} per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.83 hours

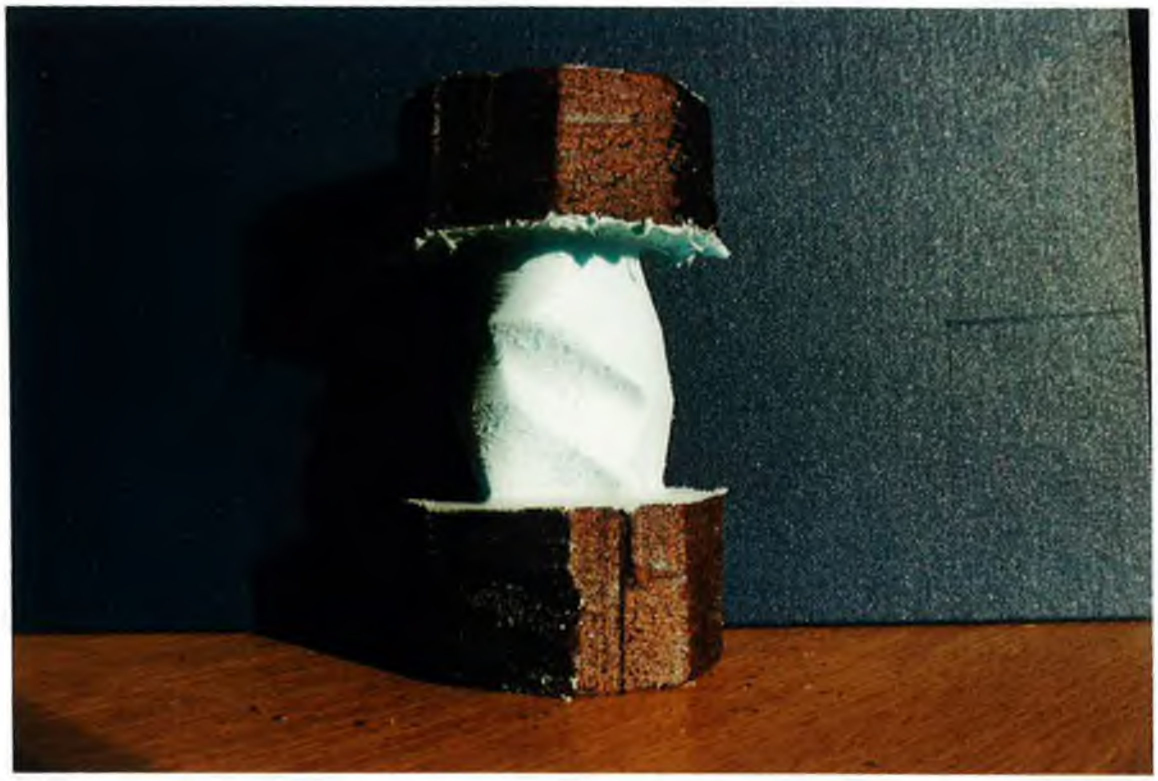


Plate 5.2(p) Result for Circle to Heart to Complex to Star Shaped Model

The milled model in Plate 5.2(p) was based on the section cut surface model from Figure 4.3(w). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis
	: 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.77 hours

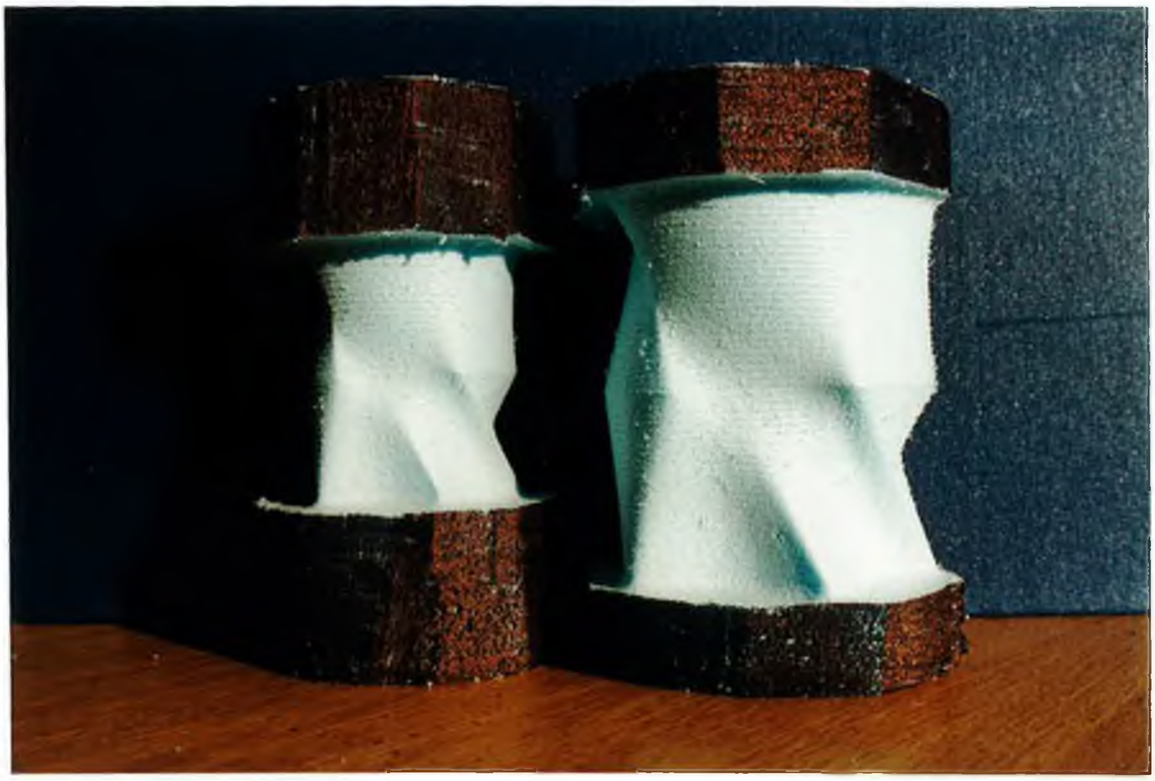


Plate 5.2(q) Result for Scaled Up Model

Plate 5.2(q) shows the original model (left) and the scaled up model (right) based on the model from Figure 4.3(v). The scaling factor is 1.5 (150 percent of the original model size). The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.375 mm per second along x-axis : 0.9° per second (0.0157 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 5.73 hours
Production time	: 7.63 hours

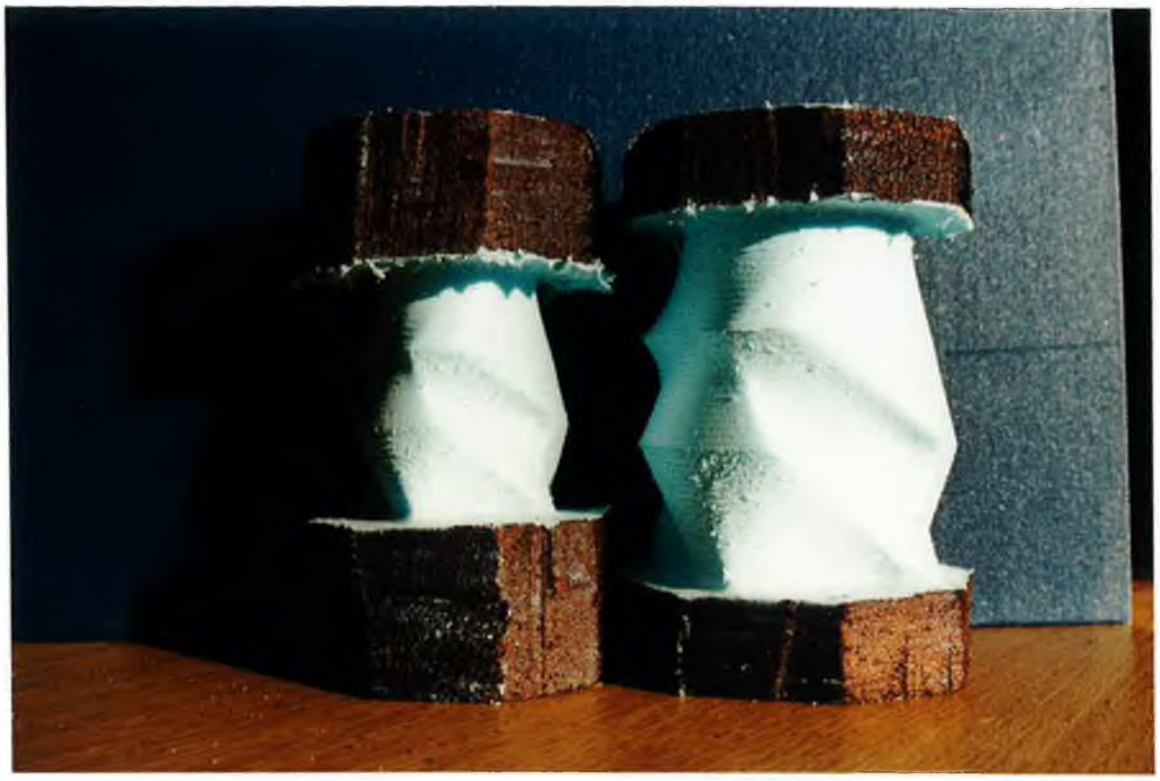


Plate 5.2(r) Result for Scaled Up and Higher Feed Rate Model

Plate 5.2(r) shows the original model (left) and the scaled up model (right) based on the model from Figure 4.3(w). The scaling factor is 1.5 (150 percent of the original model size). The feed rate is two times higher the defaults setting. The machining parameters are shown as below.

Cutter diameter	: 6 mm
Spindle velocity	: 1000 revolution per minute
Feed rate	: 0.75 mm per second along x-axis
	: 1.8° per second (0.0314 radian per second) around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes.)

Step over	: 1.5 mm
Machining time	: 2.87 hours
Production time	: 4.65 hours

Diameter of the first layer for a few milled products had been measured and compared with the dimension of designed models. Table 5.2(a) listed four types of models with their design diameters, average measured diameters, percentage of error and the average percentage of error.

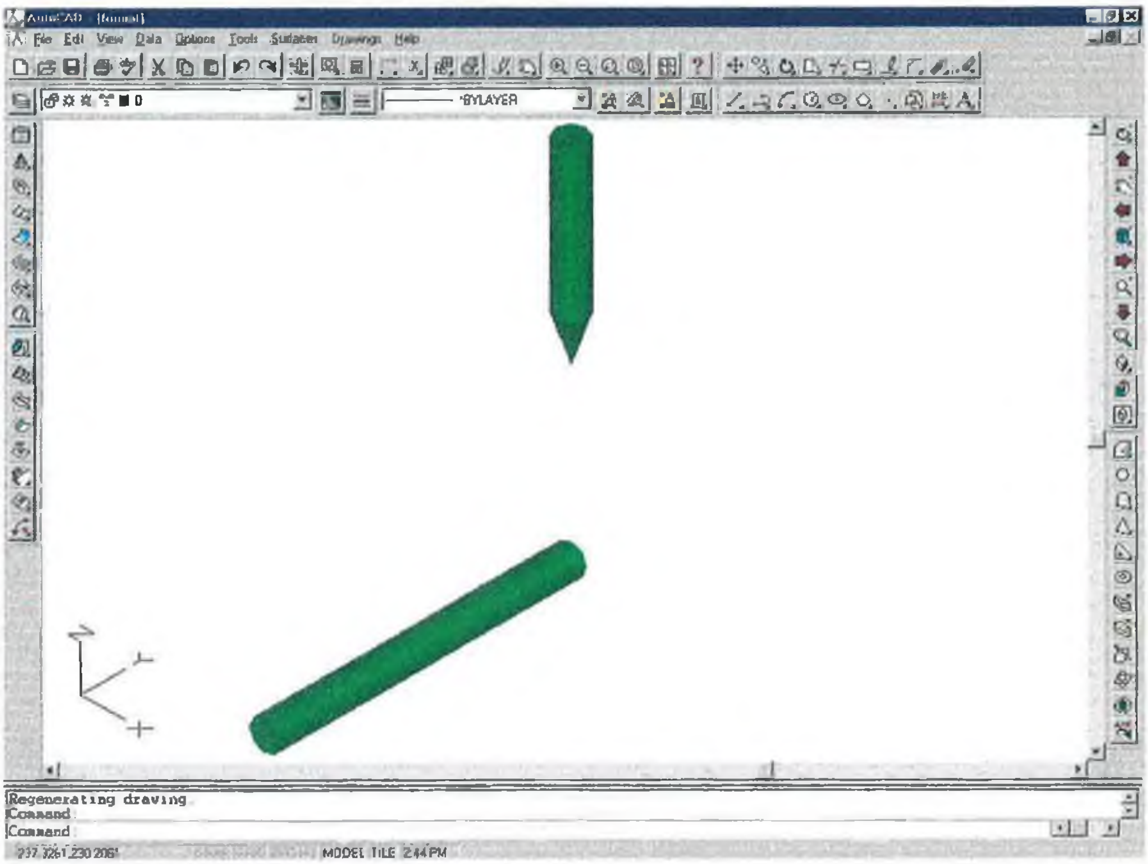
Table 5.2(a) Dimensional Comparison

Model Shape	Design Diameter (mm)	Average Measured Diameter (mm)	Percentage of Error
Circle to Complex	37.8972	38.0900	0.51
Circle to Square	31.2566	31.2250	0.10
Circle to Crescent	39.7138	39.6500	0.16
Circle to Star	39.0072	39.2900	0.72
		Average Percentage of Error	0.37

5.3 RESULTS OF THE ADDITIVE PROTOTYPING PROCESS

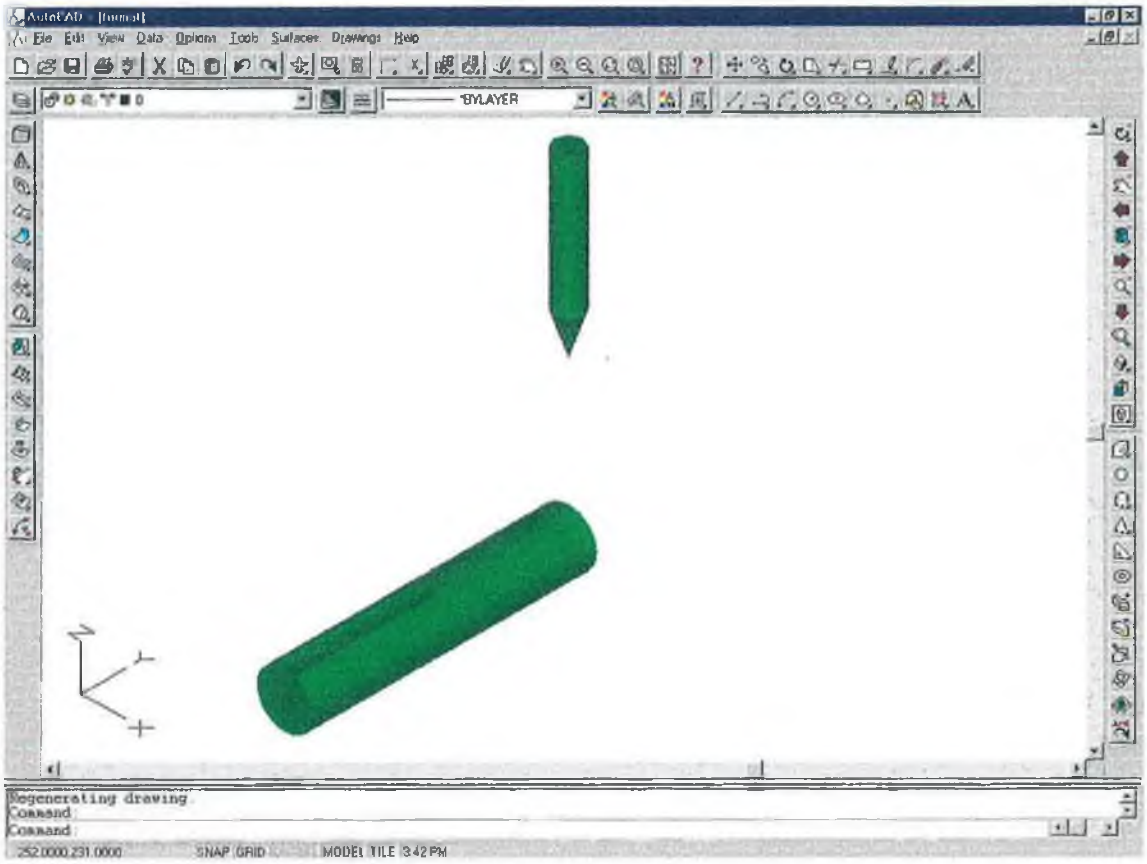
A solid cone model with four internal cavities (parent model) was created by using AutoSurf. Various smaller diameter models were later derived from the parent model (refer section 4.4.1.). The additive prototyping process for making the parent model was simulated graphically by using the AutoLISP programming language.

The simulation can only be done in the AutoCAD or AutoSurf environment. The results of the simulation are shown as below. All the computer-generated pictures are presented in isometric view from South-East Angle. The deposition tool is located in the upper portion of the picture (with sharp end). The tool and the model are normal to each other.



```
Regenerating drawing.
Command:
Command:
```

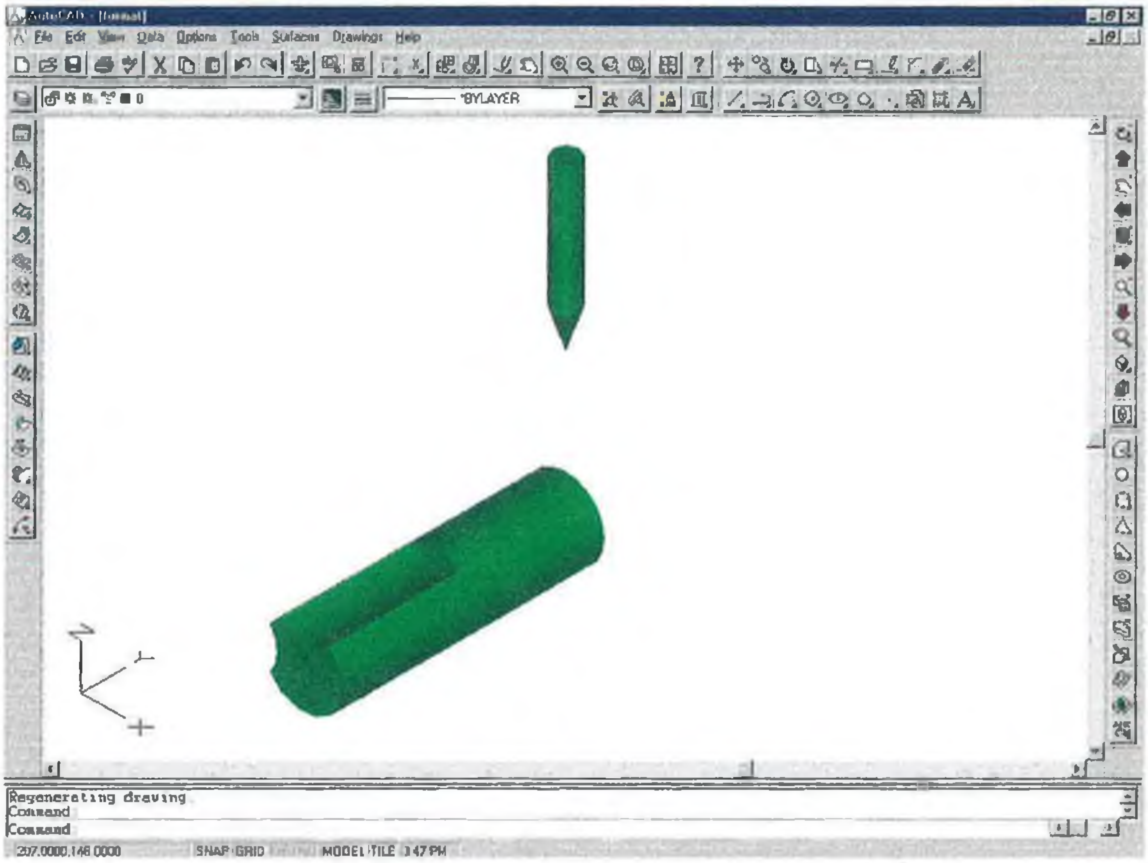
Figure 5.3(a) Deposition Tool with 10 mm Diameter Core Block



```
Regenerating drawing.
Command:
Command:
```

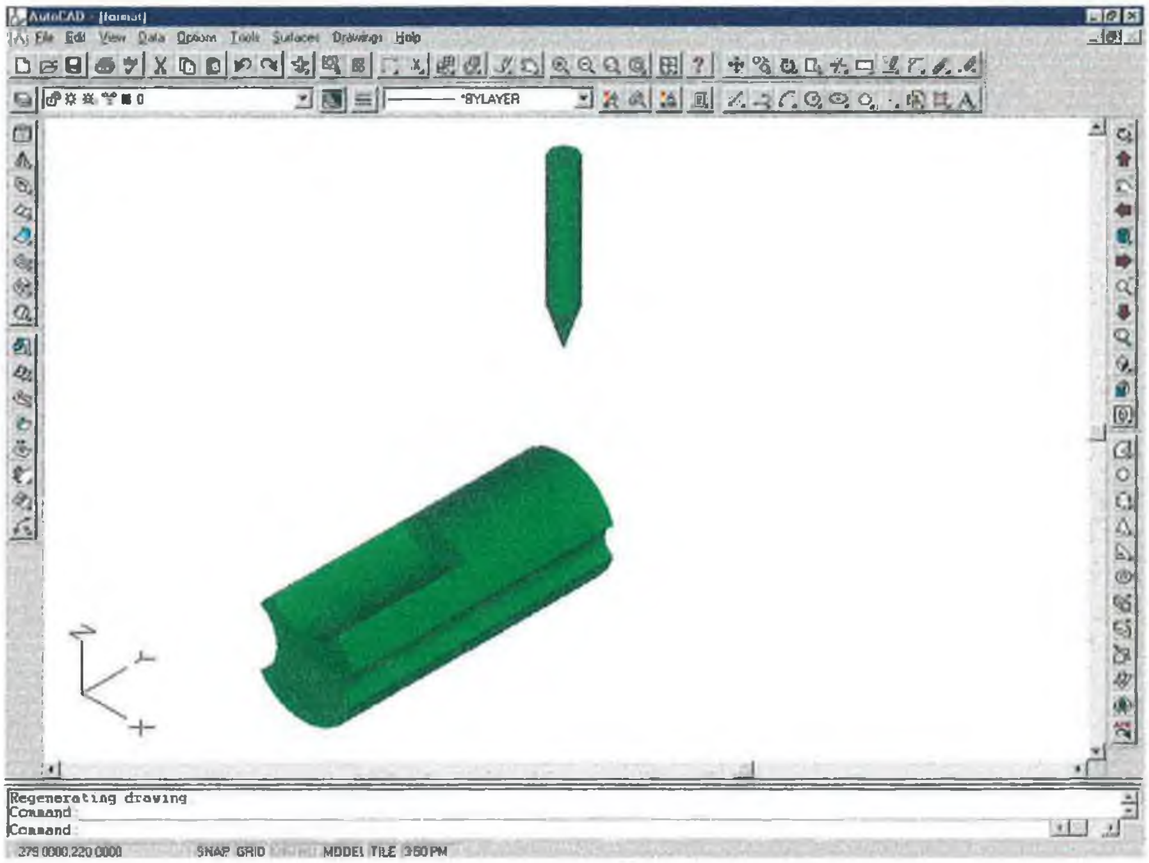
```
252.0000,231.0000    SNAP GRID    MODEL TILE 3:42 PM
```

Figure 5.3(b) Deposition Tool with 20 mm Diameter Product


AutoCAD - [format]
File Edit View Data Options Tools Surfaces Drawings Help
BYLAYER
207.0000,148.0000 SNAP GRID MODEL TILE 3:47 PM

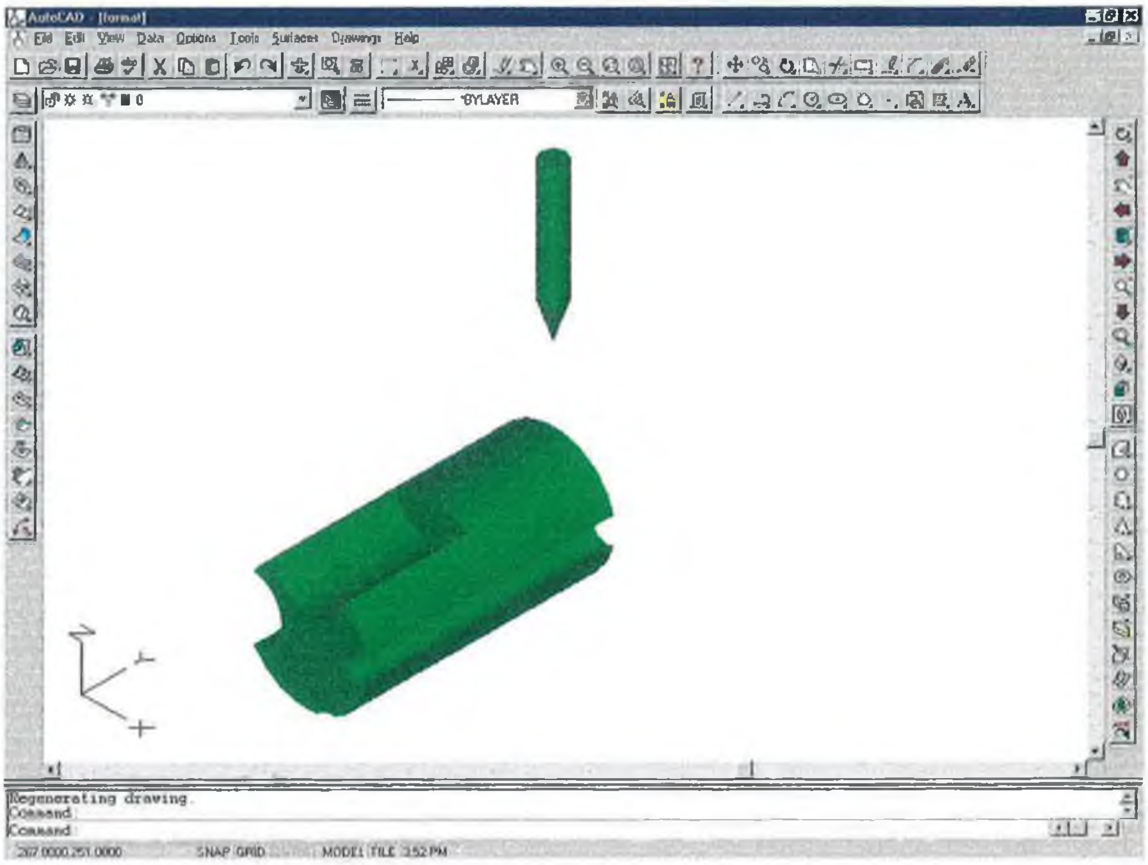
```
Regenerating drawing
Command:
Command:
```

Figure 5.3(c) Deposition Tool with 30 mm Diameter Product


AutoCAD - [format]
File Edit View Data Options Tools Surfaces Drawings Help
BYLAYER
Z
Y
X
SNAP GRID ORTHO MODEL TILE 3:50 PM

```
Regenerating drawing
Command:
Command:
```

Figure 5.3(d) Deposition Tool with 40 mm Diameter Product



```
Regenerating drawing.
Command:
Command:
```

```
207.0000,251.0000     SNAP GRID     MODEL TILE 3:52 PM
```

Figure 5.3(e) Deposition Tool with 50 mm Diameter Product

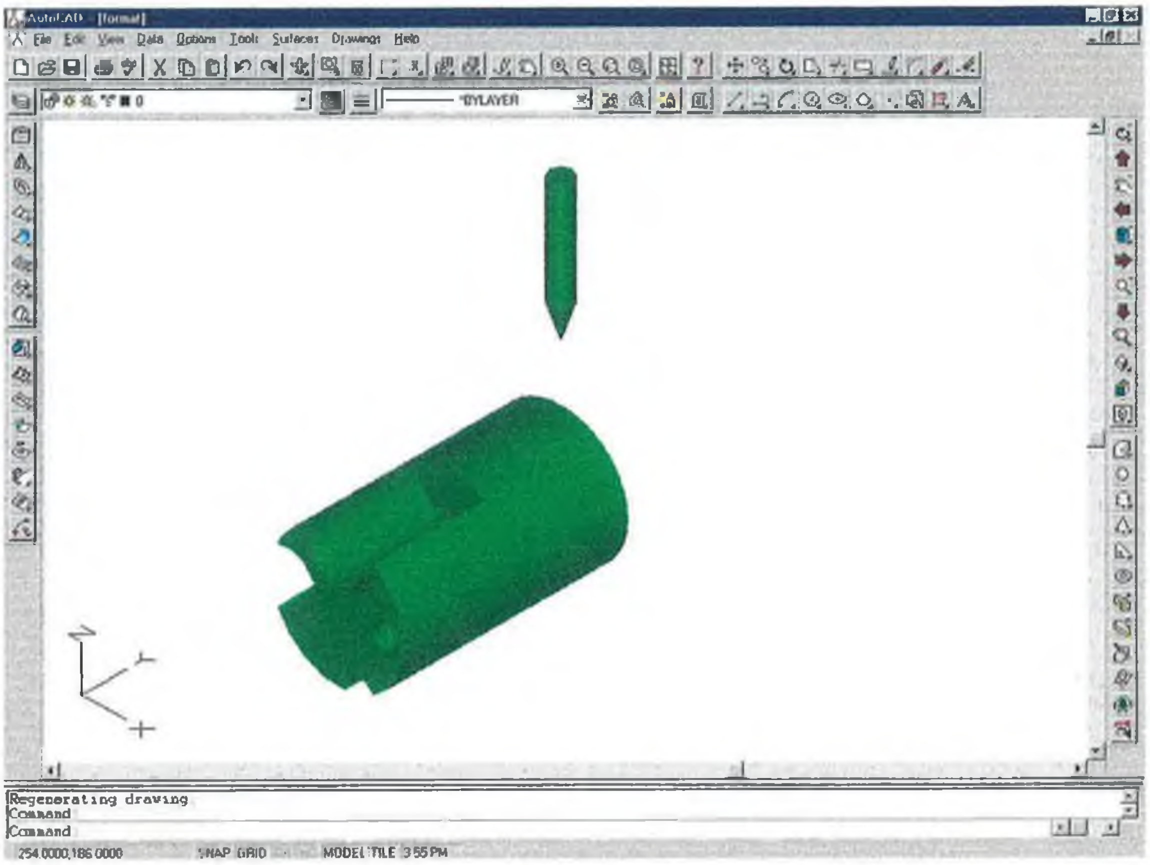
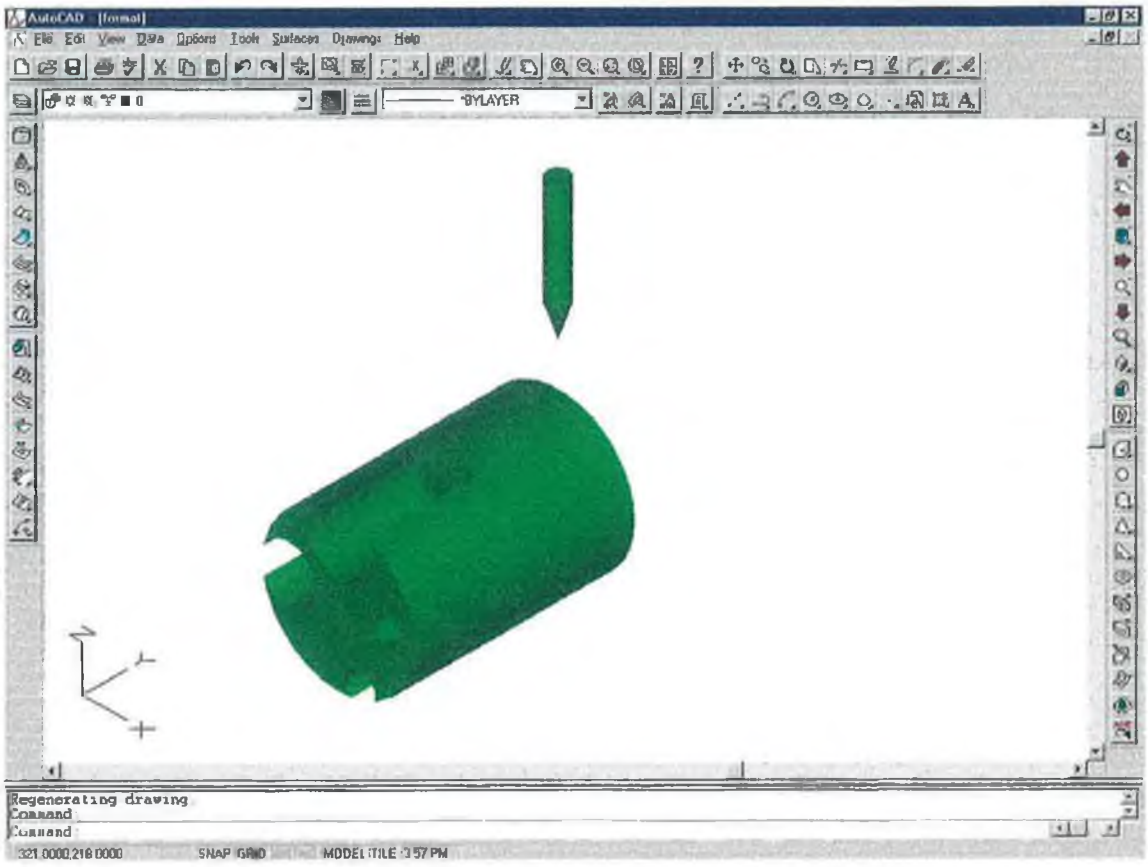


Figure 5.3(f) Deposition Tool with 60 mm Diameter Product



```
Regenerating drawing
Command:
Command:
321.0000,218.0000    SNAP GRID    MODEL TILE 3:57 PM
```

Figure 5.3(g) Deposition Tool with 70 mm Diameter Product

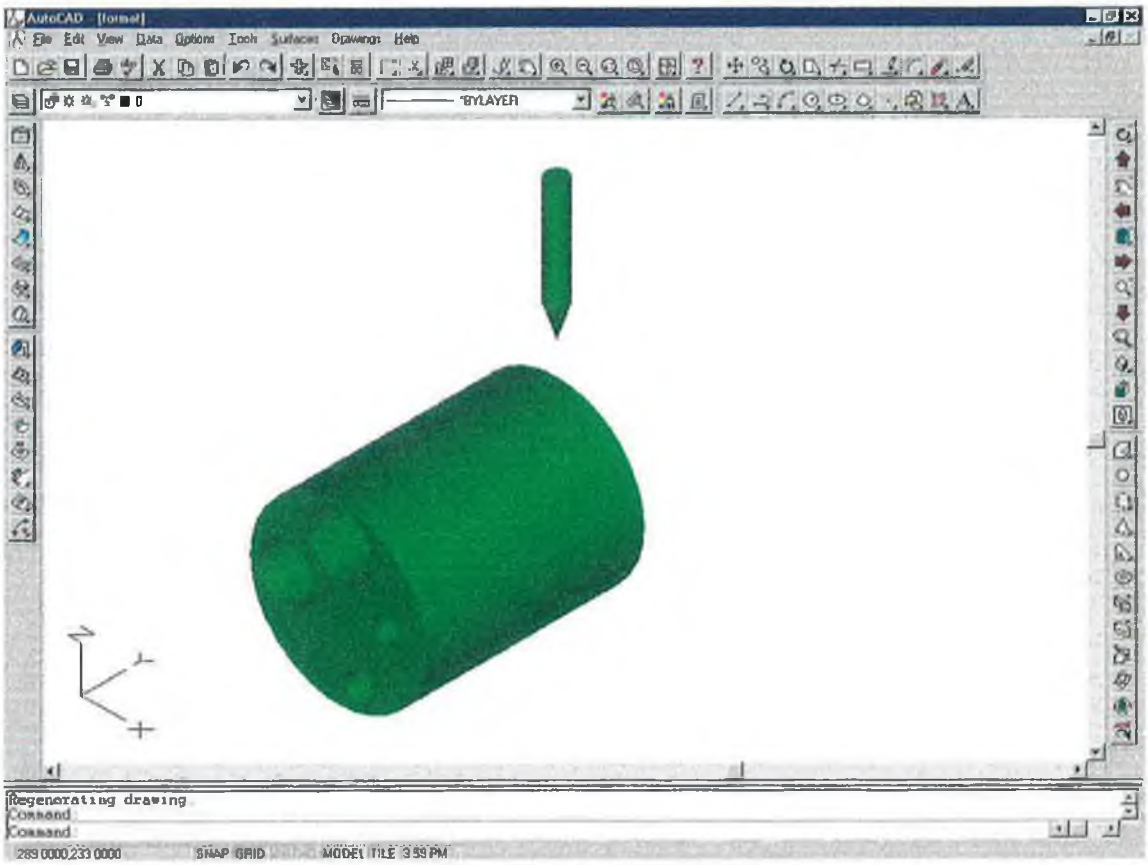
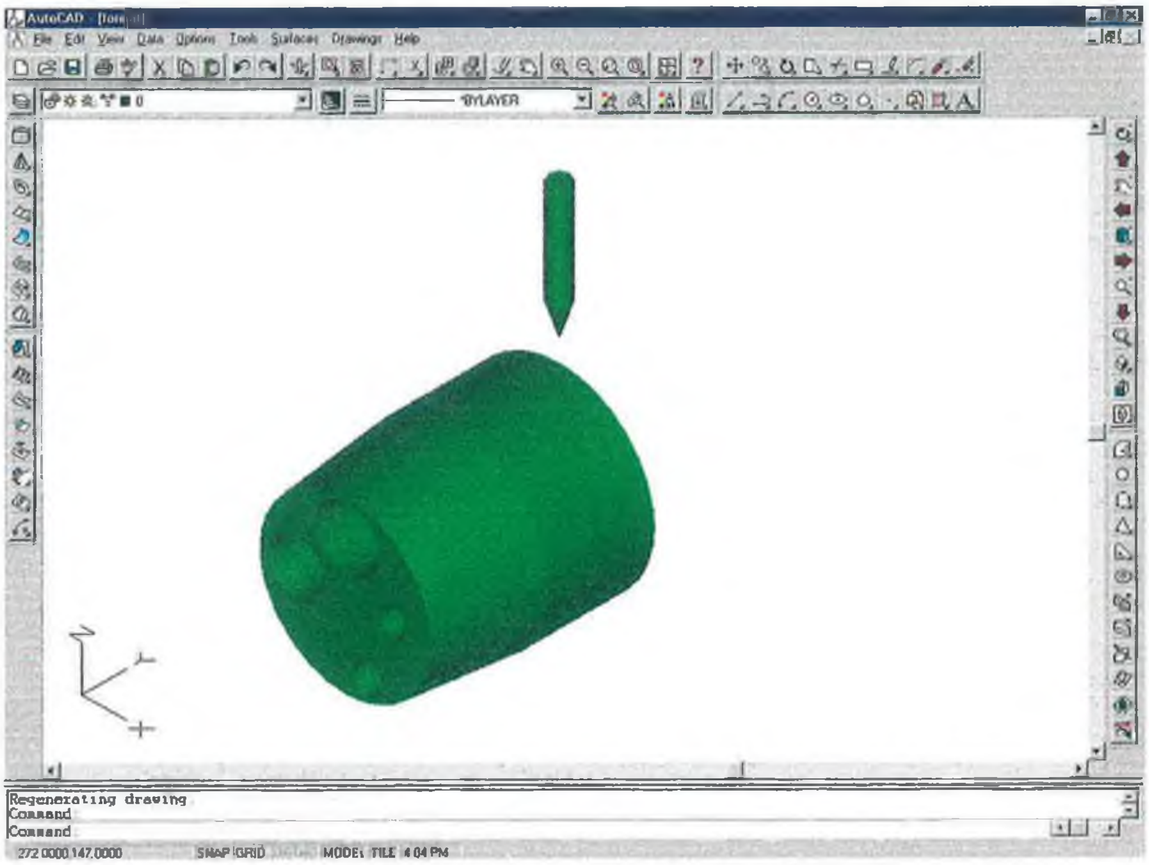
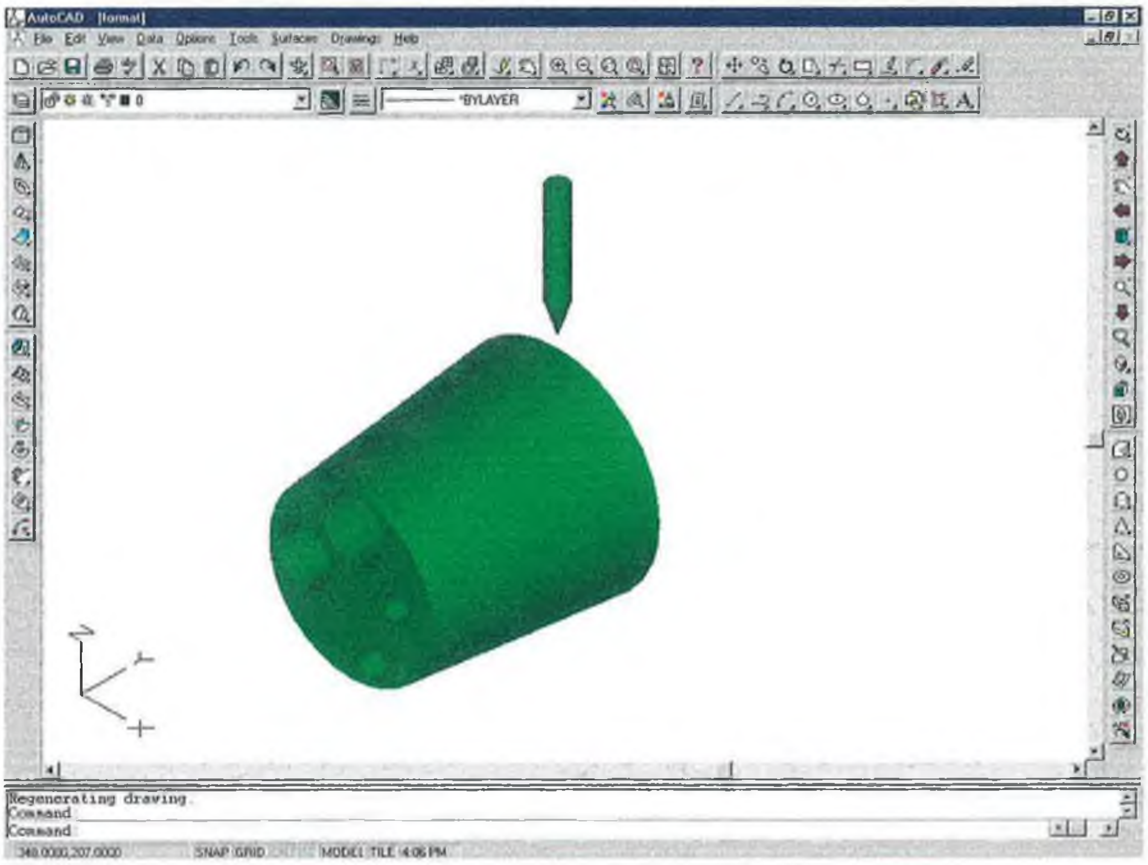


Figure 5.3(h) Deposition Tool with 80 mm Diameter Product


AutoCAD - [tore]
File Edit View Data Options Tools Surfaces Drawings Help
BYLAYER
SNAP GRID MODEL TILE 4:04 PM

```
Regenerating drawing
Command:
Command:
272.0000,147.0000
```

Figure 5.3(i) Deposition Tool with 90 mm Diameter Product



```
Regenerating drawing.
Command:
Command:
348.0000,207.0000    SNAP GRID    MODEL TILE 4:06 PM
```

Figure 5.3(j) Deposition Tool with Complete Product

5.4 DISCUSSION

Sixteen complex shaped models had been produced by using the subtractive prototyping system. Two scaled up models were also produced with one of them being machined using a higher feed rate than the default setting. The milled products are shown in Plate 5.2(a) to Plate 5.2(r) (refer section 5.2). The additive prototyping process result was simulated graphically in a CAD modelling environment. The computer generated graphic frames are shown in Figure 5.3(a) to Figure 5.3(j) (refer section 5.3). The following sections will be discussing about the system, product and other related issues.

5.4.1 Effectiveness of the Equipment

The personal computer that was used in this project provided satisfactory performance. It has a Pentium II microprocessor. Its clock speed is 300 MHz. The random access memory (RAM) is 128 MB. The hard disk capacity is about 5 GB. The personal computer has the capacity of controlling up to two PC-23 indexer since it has two ISA slot at the mother board to accommodate the indexer main circuit boards.

The communication problem between the computer and the indexer was solved (refer section 3.3.3). The system is executing well under the new control program. The new control program can be used to solve the interfacing problem if any higher end personal computer is used in the future. Command execution speed difference due to the difference in the microprocessors of the indexer and the personal computer will not pose a problem.

The personal computer has a very large RAM and hard disk capacity. A number of applications can be run at the same time. Its microprocessor and clock speed is so fast that the executions of all the applications can be carried out smoothly without sacrificing the speed. As a result, it is suitable for running powerful CAD software like

AutoCAD and AutoSurf (part of Mechanical Desktop package). It is also capable of executing programming software like Borland C++ 4.5.

However, when the control program is executing the commands to produce the subtractive and additive prototyping product, other applications in the personal computer cannot be used. It is to ensure that the control program is not interrupted by the time sharing feature of the microprocessor for producing high precision product.

The interfacing system (PC-23 indexer, KS-drives and motors) is more than ten years old. But, while it can still be used to communicate with the latest personal computer, a large amount of work was required by the author to solve the communication problem. Two indexers are available for the system. But, only one was needed since it can control up to three drives. As a result, a spare indexer is available for future development.

Four KS-drives are available. But, only three were used since the subtractive prototyping process required only three motion axes. The additive prototyping process will either need three or two motion axes, depending on the situation specified in section 4.8.2. In general, the interfacing system proved satisfactory in performing the user-specified tasks in such an accurate way.

The manipulator has four motion axes – manipulation along the x-axis, manipulation around the x-axis, manipulation along the y-axis and the manipulation around the y-axis. Due to the nature of the subtractive prototyping control program and the model definition co-ordinate system, only three axes were used and they were manipulation the along x-axis, manipulation the around x-axis and the manipulation along the y-axis. The manipulation unit around the y-axis is not used in the project. It is very stable and not easily moved by external forces. As a result, the manipulation unit around the y-axis was not locked during the subtractive prototyping process.

The gearbox of the manipulation unit around x-axis was contributing a maximum backlash of three minutes (3' = 0.000873 radian). The backlash of 3' is negligible. The manipulator provided a fairly high precision motion, and this was proved by examining the surface roughness and the dimension of the milled polystyrene models in Table 5.2 (a).

The original ball nosed cutter was attached to a Bosch hand drill. The hand drill was found to be not suitable because it cannot continuously run for a few hours. As a result, the IKA drive unit that can stand long machining hours replaced the Bosch hand drill as the subtractive prototyping tool. The new drive unit has the advantage of controlling the rotating speed of the cutter.

Generally, the hardware system configuration, which consists of the personal computer, the interfacing system (indexer, drives and motors), the precision manipulator and the ball nosed cutter equipment were integrated seamlessly. The system resulted in a general-purpose precision robotic manipulator, which could be used as a subtractive and additive prototyping tool for the present, and future projects.

5.4.2 Effectiveness of CAD/CAM

AutoSurf was used in this project for producing ruled surface models and solid models with internal cavities. AutoSurf was also used in changing the solid models into surface models. And, AutoSurf has the capability to produce the section cut surface models from the surface models.

AutoSurf is a very powerful and user friendly CAD modelling software. It has completely outdone the AutoCAD, its predecessor. As far as the project is concern, AutoSurf provided all the CAD requirements for the project. It has the internal pre-processor to change the model graphic file into a good and simple neutral format file – DXF entities file.

The step over distance of the sections cut feature in the AutoSurf will determine the amount of section cut layer of the model. More than 400 co-ordinates in each section cut layer were generated by the AutoSurf. The surface roughness is directly proportional to the amount of co-ordinates in each layer and the quantity of layers. The amount of data was found to be adequate by examining the quality of the milled product. The overall model production time and the dimensional accuracy of the models are affected by the section cut parameter as well.

All the CAM programs used in the project were developed in ANSI C programming language. The platform used is the Borland C++ 4.5 programming software. Five CAM programs were developed to facilitate the subtractive prototyping process. Seven CAM programs were developed to prepare for the additive prototyping process.

The post-processor programs for the DXF file (Extract.c and ExWax.c) can extract all the surface co-ordinates from the model in seconds. The post-processor was found to be very efficient since all the surface co-ordinates of the section cut model can be extracted out from the section cut models. However, the post-processor programs are solely for the AutoSurf section cut surface model's DXF entities file. The product files of the Extract.c and ExWax.c were verified to be correct by plotting the cross sectional profile data into Microsoft Excel.

Other subsequent programs which were built by the author using the ANSI C programming language were used to change the co-ordinate system of the data set, sorting, rearranging and converting the data into machine-readable format for the final control program. The maximum co-ordinates in each section cut layer (cross sectional profile) of the section cut model that can be handled by the Sortadd.c and SaWax.c programs are set at 4000. A higher memory personal computer will be needed for sorting more co-ordinates.

The final control program contains all the require functions to communicate effectively with the PC-23 indexer for command transferring and response decoding. It also has other functions like machining time calculation, command constructions, initialising the indexer, setting the motor resolution of the indexer and so on. The maximum height of the model that the subtractive and additive prototyping process can produce is limited only by the distance of the grippers. However, the height of the model that the additive prototyping process can handle is set at 99 mm by the author in Deposti.c and DpsSlp.c programs.

The CAM programs for the additive prototyping process was tested to be fine by using the data files generated from the section cut models in section 4.4.1.. Once the additive prototyping tool is ready, the solid cone with internal cavities can be produced. Generally, the CAD/CAM integration of the system is complete and adequate.

5.4.3 Effects of Model Shapes, Size and Feed Rate

Various polystyrene models were produced by using the subtractive prototyping system with the aid of the precision robotic manipulator. The ruled surface models were generally composed of two or more polyline profiles. The profiles were circle, heart, complex, square, cross, star and pentagon. The finished models were shown from Plate 5.2(a) to Plate 5.2(p).

The surface finish of the products is fairly good. Some of the milling chips were still attached on the model surface due to the natural properties of extruded polystyrene block. The surface finish can be further improved if the section cut surface models have more than 34 cross sectional layers – which contribute to smaller step over distance and longer production time. The thickness of the section cut layer can be as thin as a piece of A4 paper and its realisation is only limited by the practicality of the machining facilities available.

Most of the milling starting points (0^o) and ending points (360^o) were observed to have a deeper milling depth. The deeper milling depth may be due to the following causes:

(1) The starting points were created by the author by using linear interpolation, which is different from the NURBS (non-uniform rational B-spline) surface of the engineering models.

(2) The cutter was dwelling at the 0^o and 360^o location of each cross sectional profile for a longer time. It is due to the nature of the Sortadd.c program that is adding redundant co-ordinates at the start and end points of each cross sectional layer.

(3) The PID gain setting of the KS-drives may not be the best although the shapes of the milled models were matching the designed models.

The milled polystyrene models were found to have craters. This might be caused by the vibrations of the ball nosed cutter drive unit and the natural properties of the polystyrene. The cutter drive can be more stable if its base is supported from the ground instead of overhang in the air. The cutter drive vibration can also be reduced by fixing an air cushion to it.

The dimensional accuracy of the product was good. The dimensions of a few prototypes had been measured and the results were shown in Table 5.2(a). The average percentage of error is 0.38 percent. The dimensional difference may be due to the positioning system of the manipulator, the milling vibration and the natural properties of the polystyrene which tend to have chips attached on the surface or crater. It is expected that the average percentage of error for all the models of the project will be much lower if there isn't any attached chips or crater on the surface of the models.

Referring to Plate 5.2(q), the scaled up model and the original polystyrene models have the same surface finish. As a result, the scaling factor does not appear to affect the surface roughness of the models produced using the same machining parameters. Referring to Plate 5.2(r), the bigger size model was also had the same surface finish as the original polystyrene model although the feed rate for machining the big model was twice as high as the default setting.

As a result, the machining time can be reduced significantly by applying higher feed rates to the future models. In fact, the models from Plate 5.2(a) to Plate 5.2(p) that had 34 section cut layer were produced in less than five hours each. The bigger size model in Plate 5.2(q) that had 50 section cut layer was produced in less than eight hours.

The large model in Plate 5.2(r) that was also having 50 section cut layer was produced in less than five hours. The actual machining time is different from the overall production time because at least 20 percent of the production time was used in commands transfer and execution. Motion command from the PC and the response from the indexer are transferred one character at a time between the PC and the indexer. The indexer has a very low command execution speed as well. It is believed that the bottleneck of the whole system lies on the two factors mentioned above.

Generally, the system managed to produce complex shaped ruled surface models with a very good dimensional accuracy and fair surface finish. The milled product directly matched the shapes of the designed engineering models in the CAD software. With further improvement, metal products could be produced by using the equipment developed for the project.

5.4.4 Effectiveness of CAD Programming Language

AutoLISP, the programming language of AutoSurf was used in simulating the additive prototyping process. The results were shown in section 5.3. AutoLISP is based on the LISP programming language, which is simple to learn and yet very powerful. AutoLISP's programming structure is very similar to C programming language.

The program only used a few simple commands to simulate the additive process. The commands are for creating and displaying the models, rendering them, rotating them for different views and so on. The simulation can only be shown in an AutoSurf environment since the program needs the built-in LISP interpreter of the AutoSurf.

A string is a group of characters surrounded by quotation marks (refer Table 4.4(a)). The simulation program cannot create or display the model if one of the string in the model definition data is more than 132 characters. In order to ensure that the string length is within the limit, the engineering model should be moved or rotated back to its original position once it is ready for extraction (using 'entget' for extraction).

The simulation program can be executed in any new or existing AutoSurf files. If the new files don't have the required layers, line type, line colour and so on, the program will use the default settings. The default setting will show all the created models in white colour. As a result, it is better to simulate the program in an existing file that have all the necessary layers, line type, line colour and so on.

Generally, the AutoLISP program is sufficient in simulating an additive prototyping process of a solid cone with four internal cavities. Besides using its own specific commands, AutoLISP can access the built-in commands of the AutoSurf directly in its programs. AutoLISP program can be written in any text editor. But, the program must be saved in *.lsp format.

Since AutoSurf read AutoLISP code directly, no compiling is required. By entering code at the command line, the author can see the result immediately. This makes the AutoLISP language an easy one with which to experiment, regardless of the author's programming experience.

CHAPTER 6: CONCLUSION AND RECOMMENDATIONS

6.1 CONCLUSION

A rapid prototyping system using a precision robotic manipulator has been developed. The system comprises of a latest personal computer, interfacing system (PS-23 indexer, KS-drives and motors), a four degrees of freedom precision manipulator and a ball nosed end milling equipment. The hardware is integrated with the commercial available CAD software (AutoSurf) and self-developed CAM programs (for data processing and motion control) for producing subtractive prototyping models and simulating additive prototyping process. The system has the following advantages:

(1) *Lower Cost.* The hardware and software are cheaper than the market available rapid prototyping tools.

(2) *Effective.* The system can produce complex shaped objects with high accuracy.

(3) *Dual Purposes.* The system can produce subtractive prototyping products and ready to develop additive prototyping objects once the deposition tool is available.

(4) *Time Saving.* Complex shaped objects can be produced in hours without sacrificing the surface roughness and accuracy.

(5) *Space Saving.* The robotic based rapid prototyping system can save the floor space compared to the NC or CNC based systems.

(6) *All in One.* All the CAD/CAM activities can be done in one personal computer.

Although the additive prototyping tool is not available yet, its CAM programs were tested by using the data files of the section cuts models from section 4.4.1.. The programs were good and the execution was smooth. Once the equipment is ready, the additive prototyping model can be built.

6.2 THESIS CONTRIBUTION

In the course of this research, it is believed that the following contributions have been made in the general area of the research topic.

(1) Communication Interface – The author has been able to create smooth interfacing between a high specification personal computer and low specification PC-23 indexer. The low specification PC-23 indexer was designed for low specification personal computers such as PC/XT and AT. After considerable investigation, the author discovered that the execution speed of microprocessor in each interfacing component plays a vital role in producing smooth data communication. Special program can be used to slow down the execution speed of the microprocessor in one component if the speed difference is not large such as the case of the PC-23 indexer and the 486DX33 personal computer. When the difference between the execution speed of two interfacing components is large, the communication protocol has to be changed. As a result, the author has developed a communication program based on this new protocol. The new communication program will enable the high execution speed personal computer to wait until it receives the response from the low execution speed PC-23 indexer, instead of waiting for a short period of time in the previous communication protocol. In the future, any higher specification personal computer can be used to interface with the PC-23 indexer.

(2) NURBS Surface Data – The author has developed effective procedures for the production of NURBS surface data from three-dimensional solid and surface models. The three-dimensional model has to be converted into a surface model (if the original model is a solid model) before it is section cut into multiple cross sectional layers. The distance between the section cut layers (step over) will determine the surface finish of the end product. The precision accuracy of the step over distance can be set to as high as sixteen decimal points. Then, the section cut model is changed into DXF entities file for further data processing. Thus, another way of producing NURBS surface data was developed.

(3) Post-processor of DXF – The author also developed a general post-processor program for the section cut surface models' DXF entities files. The surface data of the DXF entities files can be extracted completely by using the post-processor program. This has solved the CAD and CAM interface issue for a neutral formal file, DXF entities file. Thus, the CAD model data can be completely transferred to the CAM processing programs for producing finished product. This has a significant effect on developing country like Malaysia since most of the CAD/CAM users are using AutoDesk products. The Malaysian will realise that it is possible to integrate CAD/CAM into their traditional manufacturing processes. As a result, achieving the status of developed country for Malaysia in the year 2020 is not an impossible dream.

(4) Graphic Simulation with AutoLISP – A graphic simulation program was developed by the author with AutoLISP programming language to simulate the additive prototyping process. The attempt by the author in producing a simple graphical simulation program can be considered as a new contribution towards the usage of a

CAD programming language since there is no publication evident stating the usage of AutoLISP in producing graphical simulation of an additive prototyping process in a personal computer environment.

(5) Dual-Process System – Development of a single system for both subtractive and additive rapid prototyping processes is also seen as additional contribution to the technology. The system can be used to produce three-dimensional complex shaped object in subtractive prototyping process. The CAD/CAM aspect of the additive prototyping process is also ready for production as well.

6.3 RECOMMENDATIONS

In the future, the rapid prototyping system using the precision robotic manipulator can be further improved by implementing the following recommendations:

(1) Replacing the lead screws of the manipulation units along x and y-axis with ball screw that will provide smoother and backlash free motions.

(2) Reducing the vibration of the ball nosed end milling tool by supporting the base of the drive with air cushion.

(3) CAM programs can be linked or integrated to become one.

(4) For perfect accuracy, the methods of getting the home position of the manipulator in subtractive and additive prototyping processes has to be carried out.

(5) If plastic or resin part needs to be produced, then, the ball nosed end milling equipment will need to be modified to withstand high frictional forces between the cutter and the new material surfaces. The grippers will need to be modified too.

(6) A hybrid rapid prototyping system that comprises of additive and subtractive processes can be created. A complex shaped objects with internal cavities and out of line of sights areas can be created layer by layer in additive process. The subtractive process can smoothen up the surfaces after each layer is built.

REFERENCES

1. Engelberger, Joseph F., *Robotics in Practice*, Kogan Page Ltd., 1981(reprint).

2. Kalpakjian, Serope, *Manufacturing Engineering and Technology*, Third edition, Addison-Wesley Publishing Company, Inc., 1995.

3. Coiffet, Philippe and Chirouze, Michael, *Introduction to Robotics*, Kogan Page Ltd., 1983.

4. Scott, Peter B., *The Robotics Revolution*, Bell & Bain Ltd., Glasgow, 1986.

5. Bekey, George A., Trends in Robotics, *Communication of the ACM*, **39**(2), 62, 1996.

6. Dorf, Richard C. and Kusiak, Andrew, *Handbook of Design, Manufacturing and Automation*, John Wiley & Sons Inc., 1994.

7. Smith, Edward H., *Mechanical Engineer's Reference Book*, Twelfth edition, Butterworth-Heinemann Ltd., 1994.

8. Groover, Mikell P., *Automation, Production Systems, and Computer-Integrated Manufacturing*, Prentice-Hall of India Private Ltd., 1992.

9. Groover, Mikell P. and Zimmers, Emory W., Jr., *CAD/CAM: Computer-Aided Design and Manufacturing*, Prentice-Hall, Inc., 1984.

10. McKerrow, Philip, *Introduction to Robotics*, Addison-Wesley Ltd., 1991.

11. Teicholz, Eric, *CAD/CAM Handbook*, McGraw-Hill, Inc., 1985.

12. Snyder, Wesley E., *Industrial Robots: Computer Interfacing and Control*, Prentice-Hall, Inc., 1985.

13. Waurzyniak, Patrick, Robotics Evolution, *Manufacturing Engineering*, **122**(2), 40 – 50, 1999.

14. Nof, S. Y.; Knight, J. L. and Salvendy, G., Effective Utilisation of Industrial Robotics – A Job and Skill Analysis Approach, *AIIE Transaction*, **12**(3), 126 – 255, 1980.

15. Besant, Colin B., *Computer-Aided Design and Manufacture*, Second edition, Ellis Horwood Ltd., 1983.

16. McMahon, Chris and Browne, Jimmie, *CAD/CAM – From Principles to Practice*, Addison-Wesley Publishers Ltd., 1993.

17. Orr, J. N. and Teicholz, E., *Computer Integrated Manufacturing Handbook*, McGraw-Hill, Inc., 1987.

18. Andersen, Marjorie S., Reducing Time to Market with CAD/CAM, *SME Technical Paper (Series)*, 1 – 10, SME, 1993.

19. Zutshi, Aroop, What is Hot and What is Not, *Machine Design*, **65**(10), 76 – 77, 1993.

20. Li, Wen and Bahr, Behnam, Simulation System for a Surface Climbing Robot, *Computers & Industrial Engineering*, **23**(1), 275 – 278, 1992.

21. Chen, Sun, Water Pollution Simulation over a River Basin using a Computer Graphic Model, *Water Science and Technology*, **24**(6), 101 – 108, 1991.

22. Kolodkin, V. M.; Zhukov, O. E. and Zhul'ev, S. I., Computerisation of Sand Mould Casting Technology Design, *Liteinoe Proizvodstvo*, 12, 8 – 9, 1992.

23. Sun, T. L.; Su, C. J.; Mayer, R. J. and Wysk, R. A., Shape Similarity Assessment of Mechanical Parts Based on Solid Models, *American Society of Mechanical Engineers*, **83**(2), 953 – 962, 1995.

24. Chung, Stephen H. and Ludwig, Glenn, Semi-Automated Fragmentation Assessment, *Proceedings of the Annual Symposium on Explosives and Blasting Research*, 131 – 140, January 22 – 23, 1992.

25. Machover, Carl, *The CAD/CAM Handbook*, The McGraw-Hill Companies, Inc., 1996.

26. Kellock, B., Integration – a Matter of Degree, *Machinery and Production Engineering*, **145**(3723), 51 – 56, 1987.

27. Zeid, Ibrahim, *CAD/CAM Theory and Practice*, McGraw-Hill, Inc., 1991.

28. Krouse, John K., *What Every Engineer Should Know About Computer-Aided Design and Computer-Aided Manufacturing*, Marcel Dekker, Inc., 1982.

29. Morgan, M. E., Interactive Computer Graphics CAD/CAM Interfaces to Existing Design and Manufacturing Systems, *CAD/CAM Integration and Innovation*, edited by Taraman, K., 99 – 111, Association of Computer and Automation Systems of SME, 1985.

30. Tseng, A. A.; Kolluri, S. P. and Radhakrishnan, P., Design and Construction of a CNC Machining System for Hardware and Software Development, *Proceedings of Manufacturing International '88*, American Society of Mechanical Engineers, Vol. 1, 43 – 49, 1988.

31. Dombre, E.; Fournier, A.; Quaro, C. and Borrel, P., Trends in CAD/CAM Systems for Robotics, *Proceedings of IEEE International Conference*, Vol. 3, 1913 – 1918, 1986.

32. Trostmann, E., CAD-Based Robot Planning and Control, *Robot Control - Second IFAC Symposium's selected paper*, 56.1 – 56.5, 1988.

33. Welbourn, D. B., CADCAM Benefits for the Foundry, *The Foundryman*, **81**(7), 329 – 331, 1988.

34. Treber, Jerry and Koehn, Mark, Integrated CAD/CAM Sheet Metal Production Process, *Second Biennial International Machine Tool Technical Conference*, Vol. 1, 2.93 – 2.103, 1984.

35. Joormann, Otto and Teunis, Geert, The Application of CAD/CAM Systems at Volkswagen, *Computers and Graphics*, **10**(4), 317 – 325, 1986.

36. Jones, Peter F., *CAD/CAM: Features Applications and Management*, The Macmillan Press Ltd., 1992.

37. Nagasaka, Junji, JAMA Activities for Promoting the Standardisation of CAD Data Exchange, *JSAE Review*, **17**(1), 59 – 63, 1996.

38. Sergeant, Roger N., CIM Update, *Automotive Engineering*, **99**(11), 23 – 24, 1991.

39. Wooley, D. J. and Manix, M. L., Development of an Initial Graphics Exchange Specification Capability, *Journal of Ship Production*, **3**(4), 264 – 273, 1987.

40. Magoon, G. I. And Pfrommer, C. L., Ironing Out IGES, *Computer-Aided Engineering*, **8**(1), 52 – 54, January, 1989.

41. Bloor, M. S. and Owen, J., CAD/CAM Product-Data Exchange: The Next Step, *Computer-Aided Design*, **23**(4), 237 – 243, 1991.

42. Diehl, A., Transferring Files from CAD to CAM, *Computer-Aided Engineering*, 50 – 52, January 1996.

43. Mufti, Aftab A.; Morris, Michael L. and Spencer, William B., Data Exchange Standards for Computer-Aided Engineering and Manufacturing, *International Journal of Computer Applications in Technology*, **3**(2), 70 – 80, 1990.

44. Byun, Dae-Ho; Suh, Eui-Ho; Lee, Jae-Kwan; Cho, Hyun-Seok; Park, Ki-Sik; Lim, Chae-Yeon and Koo, Kyung-Cheol, Prioritising Telecommunication Standardisation Work Areas using the Delphi Analytic Hierarchy Process based on a Spreadsheet Model, *International Journal of Computer Applications in Technology*, **11**(1), 45 – 52, 1998.

45. Yan, Xue and Gu, P., A Review of Rapid Prototyping Technologies and Systems, *Computer-Aided Design*, **28**(4), 307 – 318, 1996.

46. Styger, Lee, Rapid Prototyping and Tooling Technologies, *Materials World*, **1**(12), 656 – 658, 1993.

47. Styger, Lee, Rapid Prototyping & Tooling: The Enabling Technology of the 90's, *IEE Colloquium (Digest)*, 1.1 – 1.5, IEE, Mar 23 1994.

48. Fadel, Georges M. and Kirschman, Chuck, Accuracy Issues in CAD to RP Translations, *Rapid Prototyping Journal*, **2**(2), 4 – 17, 1996.

49. Kruth, J. P.; Leu, M. C. and Nakagawa, T., Progress in Additive Manufacturing and Rapid Prototyping, *Annals of the CIRP*, **47**(2), 1998.

50. Belforte, D. A., Robotic Manipulation for Laser Processing, *Proceedings of the SPIE-High Power Lasers and Their Industrial Applications*, Vol. 650, 262 – 270, 1986.

51. Lerner, Eric J., How Industrial Concepts Become Prototypes Fast, *Laser Focus World*, **35**(4), 117 – 122, 1999.

52. Stucki, Peter; Bresenham, Jack and Earnshaw, Rae, Computer Graphics in Rapid Prototyping Technology, *IEEE Computer Graphics and Applications*, **15**(6), 17 – 19, 1995.

53. Burns, Marshall, Quick Primer on Rapid Fabrication, *Machine Design*, **66**(5), 150 – 152, 1994.

54. Burns, M., *Automated Fabrication*, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1993.

55. Baril, Richard, *Modern Machining Technology*, Delmar Publishers Inc., 1987.

56. Drozda, Thomas J. and Wick, Charles, *Tool and Manufacturing Engineers Handbook, Fourth edition, Vol. 1: Machining*, SME, 1983.

57. *Metals Handbook, Ninth edition, Vol. 16: Machining*, Metals Park, Ohio: ASM International, 1989.

58. *The Handbook*, Dormer Tools, 1995.

59. Jacobs, P. F., *Rapid Prototyping and Manufacturing: Fundamentals of Stereolithography*, SME, Dearborn, Michigan, 1992.

60. Burns, M. *Automated Fabrication – Improving Productivity in Manufacturing*, PTR Prentice-Hall, New Jersey, 1993.

61. Hull, C., Stereolithography: Plastic Prototypes from CAD Data without Tooling, *Modern Casting*, 38, August 1988.

62. Jacobs, P., Rapid Prototyping & Manufacturing, *Society of Manufacturing Engineers*, 397 – 423, 1992.

63. Frontera, E. F., Sculpturing of Art Figures, U. S. Pat. 3301725, September 25, 1963.

64. Meyer, R. M., Relief Models, U. S. Pat. 3539410, November 20, 1967.

65. Styger, Lee, Rapid Prototyping Technologies, *IEEE Colloquium (Digest)*, **77**, 6/1 – 6/5, March 23, 1994.

66. Hansel, Bryan, Fundamentals of Product Development Getting to Market in ½ the Time, *Annual Technical Conference – Antec, Conference Proceedings*, Vol. 3, 3080 – 3083, April 26, 1998.

67. http://www.3dsystems.com

68. Hull, C. W., Apparatus for Production of Three-Dimensional Objects by Stereolithography, U. S. Pat. 4575330, August 8, 1984.

69. Philbin, Matthew L., Rapid Prototyping: A Young Technology Evolves, *Modern Casting*, **86**(3), 54 – 57, 1996.

70. Wohlers, T., Future Potential of Rapid Prototyping and Manufacturing Around the World, *Proceedings of Third European Conference on Rapid Prototyping and Manufacturing*, University of Nottingham, July 6 – 7, 1994.

71. http://www.delcam.com/info/projects/carp.htm

72. Jacobs, P. F., Fundamentals of Stereolithography, *Proceedings of Solid Free Form Fabrication Sysmposium*, 196 – 211, August 3 – 5, 1992.

73. Belforte, D. A., Laser Modelling Reduces Engineering Time, *Laser Focus World*, 103 – 108, June 1989.

74. Dimatteo, P. L., Method of Generating and Constructing Three-Dimensional Bodies, U. S. Pat. 3932923, October 21, 1974.

75. Kinzie, N. F., Method and Apparatus for Constructing a Three-Dimensional Surface of Predetermined Shape and Colour, U. S. Pat. 5015312, December 24, 1988.

76. Conley, J. G. and Marcus, H. L., Rapid Prototyping and Solid Free Form Fabrication, *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, **119**(4B), 811 – 816, 1997.

77. Feygin, M., *Apparatus and Method for Forming an Integral Object from Laminations*, U. S. Pat. 5354414, October 11, 1994.

78. Bjorke, Oyvind, *NOR-SLA Newsletter*, 3, 1, Norway, December 1990.

79. Deckard, C. R., *Part Generation by Layerwise Selective Sintering*, MSc Thesis, Department of Mechanical Engineering, University of Texas at Austin, May 1986.

80. Colley, D. P., Instant Prototypes, *Mechanical Engineering*, 68 – 70, July, 1988.

81. Wohlers, T., Creating Parts by Layers, *Cadence*, 73 – 76, April, 1989.

82. DTM's SLS Selective Laser Sintering Process and the Sinterstation 2000 System, DTM Corportation, Austin, Texas, 1994.

83. http://www.dtm-corp.com

84. Deckard, C. R., U. S. Pat. 4863538, 1989.

85. Deng, X. et al., Parametric Study of Selective Laser Sintering of a Sample Polymer System, *Proceedings of Solid Free Form Fabrication Symposium*, 102 – 109, August 3 – 5, 1992.

86. McAlea, Kevin; Booth, Richard; Forderhase, Paul and Lakshminarayan, Uday, Materials for Selective Laser Sintering Processing, *International SAMPE Technical Conference*, 949 – 961, October 9 – 12, 1995.

87. http://www.stratasys.com

88. Comb, J. W. and Priedeman, W. R., Control Parameters and Materials Selection Criteria for Rapid Prototyping Systems, *Proceedings of Solid Free Form Fabrication Symposium*, 86 – 93, August 9 – 21, 1993.

89. Ahmad, Luqman; Eckstrand, Lesley and Pantarotto, Jason, Rapid Prototyping & Solid Freefrom Manufacture, *Canadian Ceramic Society Journal*, **66**(2), 104 – 107, 1997.

90. http://www.cubital.com

91. Dickens, P. M., Research Developments in Rapid Prototyping, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, **209**(B4), 261 – 266, 1995.

92. Bhatti, M. T. L., *Effectiveness of Computer Controlled Robotic Precision Manipulator*, PhD Thesis, School of Mechanical and Manufacturing Engineering, Dublin City University, 1991.

93. *PC-23 Indexer User Guide*, P/N: 88-007015-03E, Compumotor Division, Parker Hannifin Corporation, U.S.A., 1987.

94. *KS-Drive User Guide*, P/N: 88-007042-01A, Compumotor Division, Parker Hannifin Corporation, U.S.A., 1987.

95. Bradley, D. A.; Dawson, D.; Burd, N. C. and Loader, A. J., *Mechatronics – Electronics in Products and Processes*, Chapman and Hall, 1991.

96. *EUROSTAR Digital User Guide*, EUROSTO795EU, IKA Labortechnik, Germany, 1998.

97. *Roofmate SL Extruded Polystyrene*, Dow Chemical Company Limited, U.K., 1995.

98. *AutoSurf Release 3 - Surface Modelling*, P/N: 03704-000000-5010, Autodesk, Inc., 1996.

99. Holmes, B. J., *Programming with ANSI C*, B. P. C. Hazell Books Ltd., Aylesbury, England, 1995.

100. Head, G. O., *AutoLISP in Plain English – A Practical Guide for Non-Programmers*, Second edition, Ventana Press, Inc., 1989.

101. *AutoCAD Release 13 – Customisation Guide*, P/N: 00105-010000-5040, Autodesk, Inc., 1994.

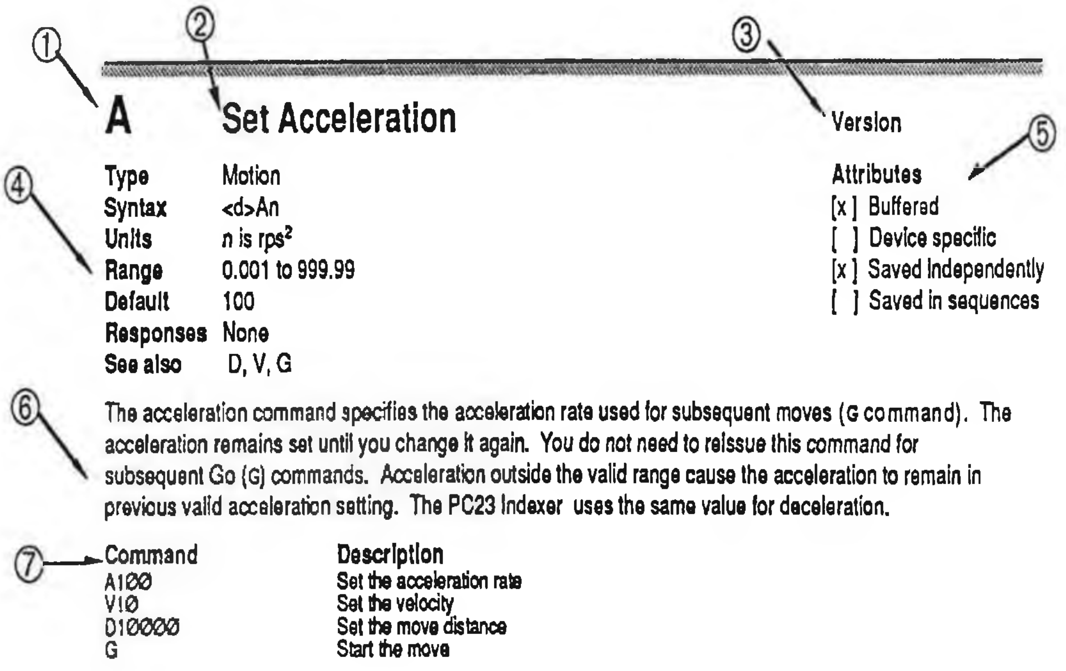
102. Arnold, J. A. and Teicholz, P., Data Exchange: File Transfer, Transaction Processing and Application Interoperability, *Computing in Civil Engineering*, 438 – 444, 1996.

103. Aitken, Peter and Jones, Bradley L., *Teach Yourself C in 21 Days*, Fourth Edition, SAMS Publishing, 1997.

Appendix A PC-23 Indexer Commands

Command Format Description

The following section describes the format of the command descriptions used in this chapter. The numbered arrows refer to the numbered sections below the drawing.



A **Set Acceleration** Version

Type	Motion	**Attributes**
Syntax	<d>An	[x] Buffered
Units	n is rps^2	[ ] Device specific
Range	0.001 to 999.99	[x] Saved Independently
Default	100	[ ] Saved in sequences
Responses	None	
See also	D, V, G	

The acceleration command specifies the acceleration rate used for subsequent moves (G command). The acceleration remains set until you change it again. You do not need to reissue this command for subsequent Go (G) commands. Acceleration outside the valid range cause the acceleration to remain in previous valid acceleration setting. The PC23 Indexer uses the same value for deceleration.

Command	Description
A1ØØ	Set the acceleration rate
V1Ø	Set the velocity
D1ØØØØ	Set the move distance
G	Start the move

① Command Identifier

The letter or letters used to represent the command.

② Command name

This name used to refer to the command. For example, Acceleration for the A command.

③ Version

The revision of software in the PC23 Indexer when the described command was first introduced or last modified. If the revision level of the software you are using is equal to or greater than the revision level listed here, the command is available in your unit. You can determine the level of software in your PC23 Indexer by issuing the Revision Level (RV) command.

④ Characteristics

The following sections describe the main characteristics of the command.

Type

This portion of the box contains the command's type. The four command types are listed below.

Set-Up: These commands define Set-Up conditions for the application. Set-Up commands include the following types of commands:

- ❑ Homing (go home acceleration and velocity, etc.)
- ❑ Input/Output (limits, scan time, in-position time, etc.)
- ❑ Tuning (servo or position tracking)
- ❑ General (set switches, return to factory settings, etc.)

Programming: Programming commands affect programming and program flow. For example, trigger, output, all sequence commands, quote, time delays, pause and continue, enable and front-panel, loop and end loop, line feed, carriage return, and backspace.

Status: Status commands respond (report back) information.

Motion: Motion commands affect motor motion (for example, acceleration, velocity, distance, go home, stop, direction, mode, etc.)

Syntax

This field shows the syntax for the command. PC23 Indexer commands use the following generic syntax: `acspd`

Variable a	This variable is the device address. If the address is optional it is shown in angle brackets: <a>. Only commands which require the PC23 Indexer to send a response require a device address. All commands may use a device address to designate which unit on a daisy chain the command is intended for.
Variable c	This variable is the command identifier, which is one or more letters.
Variable s	This variable represents a sign. A sign is not allowed for all commands. The s is not shown in the syntax if not allowed.
Variable p	This variable represents the parameters the command requires. There may be zero or more parameters. If the number of parameters is zero n is not shown in the syntax
Variable d	This variable is the end of command delimiter. This is always required and is not shown in the following descriptions for clarity. The delimiter may be a space character or a carriage return.

Units

This field describes what unit of measurement the parameter in the command syntax represents.

Range

This is the range of valid values that you can specify for n (or any other parameter specified).

Default

The default setting for the command is shown in this box. A command will perform its function with the default setting if you do not provide a value.

Response

The response to the command is shown in this box. Status commands report a condition in the indexer. Status commands do not affect the status they read.

Commands that set parameters report the parameters when the command is issued without a parameter. For example, A1ØØ sets the acceleration to 100 rps, but 1A returns the current setting. *Note: To receive a response, a device address is required.*

See Also

Commands that are related or similar to the command described are listed here.

⑤ Attributes

Each command has attributes as shown below.

Attributes
[x] Buffered
[ ] Device specific
[x] Independently saved
[ ] Saved in sequences

Buffered

If the Buffered box is checked the command is buffered. If it is not checked the command is acted on immediately. Buffered commands are executed in the order they are received. An internal buffer, or storage area, holds the commands in a queue until the previous command has been executed.

Immediate commands are executed as they are received. Immediate commands are executed even if the command buffer has commands in it. For example, the Stop (S) command is immediate. When a Stop command is received the motor is stopped as soon as the command is received. The PC23 Indexer does not process the commands in its command buffer before stopping the motor.

Device specific

If the Device specific box is checked the command requires a device identifier. If it is not checked the command may be used with or without a device identifier. Commands which are device specific are normally Status commands. Device specific commands have a syntax description with a d by itself before the command. If it is not device specific the command syntax description has a <d> in angle brackets before the command.

Saved always

If the Independently saved box is checked the parameter controlled by the command is always saved. This differs from commands which may only be saved in sequences and those which are never saved. If neither the Saved always nor the Saved in sequences box is checked the command is never saved.

Saved in sequences

If the Saved in sequences box is checked the command will be saved only if it is in a sequence and you issue the Save command (SV). If neither the

Saved always nor the Saved in sequences box is checked the command is never saved.

⑥ Description

A description of the command appears in this area along with any special considerations you should know about.

⑦ Example

An example of how to use the command appears in this area. The left column contains the commands you would issue to the PC23 Indexer. The right column contains descriptions of what the commands do in the program.

Alphabetical Command List

A Set Acceleration

Version A

Type	Motion
Syntax	aAn
Units	n = rps^2
Range	0.01 to 999.99 (Motor dependant)
Default	100
Response	None
See also	D, V, G

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Acceleration command specifies the acceleration rate to be used upon executing the next Go (G) command. The acceleration remains set until you change it. You do not need to reissue this command for subsequent G commands. Accelerations outside the valid range cause the acceleration to remain in previous valid A setting.

Command	Description
>MN	Set to Normal mode
>A5	Set acceleration to 5 rps^2
>VØ	Set velocity to 10 rps
>D1ØØØØ	Set distance to 10,000 steps
>G	Execute the move (Go)

AB Report Analog Voltage (Binary)

Version B3X

Type	Status
Syntax	aABn
Units	n = channel
Range	0 - 3
Default	None
Response	nn (two ASCII characters)
See also	AV

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Read the analog input voltage on channel number n referenced to analog input ground and respond in binary value. The analog voltage request responds with two bytes. The first byte is the A/D channel number (analog channel 0 through 3). The second byte is the 8 bit number for that channel on the joystick's A/D converter.

The 8 bit number corresponds to values between 0 and 255. Since the analog input voltage ranges between 0-2.5VDC, the binary reading will approximately correspond to the voltage on the input.

This command is very useful for quickly reading the analog input voltage.

Command	Description/Response
AB1	Reports voltage on Channel 1

AV Report Analog Voltage (ASCII)

Version A

Type	Status
Syntax	aAVn
Units	None
Range	None
Default	None
Response	CHn:n.nnV
See also	AB

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Read the analog input voltage on a channel number (n) referenced to analog input ground. The response is in ASCII. This command is necessary for joystick setup. After hooking up the joystick, read each channel and verify that Channel 0 - Channel 1 and Channel 2 - Channel 3 are both positive. Refer to the joystick installation procedure for test and calibration.

Command	Response
AVØ	CH0:1.45V

B Report Buffer Status

Version A

Type	Status
Syntax	aB
Units	None
Range	None
Default	None
Response	*R or *B
See also	None

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The buffer status command will report the status of the command buffer. The command buffer is 1,000 bytes long. The response to this command is:

*R = More than 31 bytes are free
*B = Less than 32 bytes are free

This command is commonly used when a long series of commands will be loaded remotely. If the buffer size is exceeded, the extra commands will not be received by the Controller.

Command	Response
1B	1:*R (more than 31 bytes of the buffer are free)

C Continue

Version A

Type	Programming
Syntax	aC
Units	None
Range	None
Default	None
Response	None
See also	PS, U

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Continue (C) command ends a pause state. It enables your indexer to continue executing buffered commands. After you initiate a pause with the Pause (PS) command or the Pause and Wait for Continue (U) command, you can clear it with a C command. This command is useful when you want to transmit a string of commands before you actually need to execute them.

Command	Description
PS	Pause execution on axis #1 until the indexer receives a C command
MC	Set to Continuous mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
G	Execute the move (Go)
T1Ø	Wait 10 seconds after the move
VØ	Set velocity to zero
G	Decelerate the motor to zero velocity
C	Start executing commands in buffer

CG Set Correction Gain

Version A

Type	`Set-Up`
Syntax	`aCGn`
Units	`n = gain`
Range	`0 - 8`
Default	`8`
Response	`None`
See also	`FSB, FSC, CM, MV`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to set the amount of error (steps) that should be corrected on the initial position maintenance (`FSC1` command) correction move (which takes place whenever the motor is stationary). This function is only valid in the Encoder Step mode (`FSB1` command).

The percentage of error that the Position Maintenance function will attempt to correct on each correction move is `n/8 • 100%`. If you set n to 1, the system will correct the error slowly (1/8 of the error is corrected on each try). This type of correction is performed smoothly. If you set n to 8, the system will correct the error faster. However, there may be more overshoot and ringing at the end of this type of correction move.

Command	Description
`FSB1`	Set to Encoder Step mode
`FSC1`	Enable position maintenance while the motor is stationary
`CG3`	The system corrects 3/8 of the final-position error on the initial correction move

CR Carriage Return

Version A

Type	`Set-Up`
Syntax	`aCR`
Units	`None`
Range	`None`
Default	`None`
Response	`None`
See also	`None`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Carriage Return (`CR`) command determines when the indexer reaches a particular point in the execution buffer. When the indexer reaches this command in the buffer, it responds by issuing a carriage return (ASCII 13) over its interface back to the host computer. If you place the `CR` command after a Go (`G`) command, it will indicate when a move is complete. If you put the `CR` command after a Trigger (`TR`) command, it will indicate when the trigger condition is met.

Command	Description
`MN`	Set to Normal Mode
`MPA`	Set to Absolute Position mode
`A5`	Set acceleration to 5 rps^2
`V5`	Set Velocity to 5 rps
`D5ØØØ`	Set distance to 5,000 steps
`G`	Execute the move (Go)
`CR`	Send a carriage return at the end of the move

D Set Distance

Version A

Type	`Motion`
Syntax	`aDn`
Units	`n = steps`
Range	`0 - ±99,999,999`
Default	`25,000`
Response	`None`
See also	`A, V, G, MN`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Distance (`D`) command defines either the number of steps the motor will move or the absolute position it will seek after a Go (`G`) command is entered. In incremental mode (`MPI` or `FSAØ`), the value set with the `D` command will be the distance (in steps) the motor will travel on all subsequent `G` (Go) commands.

In absolute mode (`MPA` or `FSA1`), the distance moved by the motor will be the difference between the current motor position and the position (referenced to the zero position) set with the `D` command The Distance (`D`) command has no effect on continuous moves (`MC`).

Command	Description
`2MN`	Set to Normal mode
`2A5`	Set acceleration to 5 rps^2
`2V10`	Set velocity to 10 rps
`2D5ØØØØ`	Set distance to 50,000 steps
`2G`	Execute the move (Go)

DB Set Dead Band

Version A

Type	Set-Up
Syntax	`<a>DBn`
Units	n = Encoder Counts
Range	0 - 999999
Default	0
Response	None
See also	DW, FS

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Once the initial move is completed, position maintenance will determine if a position error condition exists. The error is expressed in terms of Encoder Counts. If the error exceeds the DB setting (in either direction) position maintenance will generate a correction move, in the appropriate direction, based on the DGn and CMn settings.

Position maintenance will remain active and correction moves will continue to be generated until a FSCØ command is issued by the host.

The maximum allowable DB setting is 999999. Any attempt to exceed the maximum will leave the previous setting unchanged.

DPA Display Position Actual

Version A

Type	Status
Syntax	`<a>DPA<n>`
Units	n = steps
Range	None
Default	None
Response	None
See also	PR, P, PX, W

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Display Position Actual (DPA) command displays the actual position in feedback steps. The function is canceled by sending any single ASCII character to the indexer.

Command	Description
DPA	Continually report the actual position of axis #1
2DPA1	Reports the actual position of axis 2, one time

DW Set Dead Band Window

Version A

Type	Set-Up
Syntax	`<a>DWn`
Units	n = Motor steps (FSBØ) or Encoder Counts (FSB1)
Range	0 - 999999
Default	250
Response	None
See also	FSB

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command specifies a backlash deadband. The deadband compensates for mechanical systems with backlash and allows use of the Stall Detection features. Stall detection occurs when the position error exceeds the backlash setting.

Command	Description
FSB1	Enter Encoder Step mode
D25ØØØ	Set distance to 25,000 encoder counts
DW1ØØ	Allow 100 counts of error
G	Execute the move (Go)

ER Set Encoder Resolution

Version A

Type	`Set-Up`
Syntax	`aERn`
Units	`n = encoder steps`
Range	`1 - 50,000`
Default	`4,000`
Response	`None`
See also	`FS commands`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command defines the number of encoder counts per one revolution of the motor. The number of lines on an encoder should be multiplied by 4 (quadrature) to arrive at the correct encoder resolution value per revolution of the motor. In other words, one line of an encoder produces 4 encoder steps.

If you are not sure what the resolution of the encoder is, you may do the following to find the encoder resolution.

① Zero the position counter using the `PZ` command.

② Move the motor 1 revolution in motor mode (`FSBØ`).

③ Read the encoder position (`PX`) command. This reading indicates what the resolution will be.

Repeat this several times and use the average value as the `ER` command.

A 4:1 ratio of motor steps to encoder steps is necessary for proper closed-loop operation. If a lower ratio is used, it may be difficult to tune the position maintenance (Servoing) feature of your indexer.

Command	Description
`2ER8ØØØ`	Set encoder resolution to 8,000 steps on axis #2.

FR Report Encoder Functions

Version A

Type	`Status`
Syntax	`aFR`
Units	`None`
Range	`0 = function off, 1 = function on`
Default	`0`
Response	`a:nnnnnnnn`
See also	`FS commands`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to request the status of `FS` command functions. The response contains one ASCII digit per function (zero [Ø] or one [1]). The digits (`n`) correspond to the functions, (A - H, left to right). One (1) means an `FS` function is on. Zero (Ø) means an `FS` function is off.

A *Incremental = OFF (Ø), Absolute = ON (1).* Defines the move distances (`D`) as either incremental from current position, or as absolute referenced to the absolute 0 position).

B *Motor step mode = OFF (Ø), Encoder step mode = ON (1).* Defines the distance in motor steps or encoder steps.

C *Position Maintenance = OFF (Ø), = ON (1).* Enables position maintenance. This causes the indexer to servo the motor to the desired position if it is not in the correct position at the end of a move, or, if the motor is forced out of position while at rest.

D *Terminate Move on Stall Detect = OFF (Ø), = ON (1).* Instructs the indexer to abort a move if it detects a stall.

E *Turn on Output on Stall Detect = OFF (Ø), = ON (1).* Instructs the indexer to turn an output on if it detects a stall.

F *Multiple axis stop = OFF (Ø), = ON (1).* Instructs the indexer to abort any move if a signal is received on the Trigger #6 input . If output on stall is enabled (`FSE1`), the indexer will also turn on an output when a trigger is seen.

G ***Reserved***

H ***Reserved***

Command	Response
`1FR`	`1:11000000`. Axis 1 is in absolute encoder step mode. All other `FS` functions are OFF.

FSA Set Absolute/Incremental Positioning Mode

Version A

Type	Set-Up
Syntax	aFSAn
Units	n = function
Range	0 = off, 1 = on
Default	0
Response	None
See also	FR, MPI, MPA, PZ, PR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the indexer to perform its moves in either absolute or incremental positioning mode.

FSAØ = Incremental mode (equivalent to MPI command)
FSA1 = Absolute mode (equivalent to MPA command)

In Incremental mode (FSAØ or MPI), all moves are made with respect to the position at the beginning of the move. This mode is useful for repeating moves of the same distance.

In Absolute mode (FSA1 or MPA), all moves are made with respect to the absolute zero position. The absolute zero position is set to zero when you power up the indexer or execute the Position Zero (PZ) command.

Command	Description
MN	Set to Normal mode
FSA1	Set Indexer to absolute mode
PZ	Reset the absolute counter to zero
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Move motor to absolute position 25,000
G	Execute the move (Go)
D5ØØØØ	Move motor to absolute position 50,000
G	Execute the move (Go)

The motor moves 25,000 steps, and an additional 25,000 steps to reach the absolute position of 50,000.

FSB Set Motor/Encoder Step Mode

Version A

Type	Set-Up
Syntax	aFSBn
Units	n = function
Range	0 = off, 1 on
Default	0
Response	None
See also	D, ER, FR, FSC, MR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets up the indexer to perform moves in either motor steps or encoder steps.

FSBØ = Motor step mode
FSB1 = Encoder step mode

In Motor Step mode, the distance command (D) defines moves in motor steps.

In Encoder Step mode, the distance command defines moves in post-quadrature encoder steps. You must set up the indexer for the correct encoder resolution The Encoder Resolution (ER) command is used to define the post-quadrature encoder resolution.

If you enable encoder step mode (FSB1), without having the encoder connected to the PC23, upon receiving a Go (G) command, the motor will drift at low velocity. This drift is a result of the PC23 not receiving encoder pulses properly.

Enabling Encoder Step mode does not guarantee that your moves will position to the exact encoder step commanded. Position maintenance (FSC) must be enabled to activate closed loop servoing.

Command	Description
ER4ØØØ	Set encoder resolution to 4,000 post-quadrature encoder pulses
FSB1	Set to Encoder Step mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D4ØØØ	Set distance to 4,000 encoder steps
G	Execute the move (Go)

FSC Enable Position Maintenance

Version A

Type	Set-Up
Syntax	aFSCn
Units	n = function
Range	0 = off, 1 = on
Default	0
Response	None
See also	ER, FR, FSB, FSD, CM

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command enables and disables the position maintenance function.

FSC1 = Enable Position Maintenance
FSCØ = Disable Position Maintenance

Enabling position maintenance will cause the indexer to servo the motor until the correct encoder position is achieved. This occurs at the end of a move (if the final position is incorrect) or any time the indexer senses a change in position while the motor is at zero velocity. You must have an encoder connected, and set the indexer in Encoder Step mode (FSB1) in order to enable position maintenance.

If you enabled position maintenance (FSC1) and the motor drifts, the encoder may not be connected properly. The motor will drift at 0.1 rps per revolution.

CAUTION

If you are making a move with position maintenance enabled and the encoder is disconnected, the PC23 will continue to output pulses trying to find the desired position. Consequently, the PC23 could conceivably output pulses forever, or until a limit is encountered. For safety reasons, you should enable Stall Detection (OSE1) and Stop On Stall (FSD1) to ensure that the motor will stop if such a situation occurs.

Command	Description
ER4ØØØ	Set encoder resolution to 4,000
FSB1	Set to encoder step mode
OSE1	Enable stall detection
FSD1	Enable stop on stall
FSC1	Enable position maintenance

FSD Enable Stop on Stall

Version A

Type	Set-Up
Syntax	aFSDn
Units	n = function
Range	0 = off, 1 = on
Default	0
Response	None
See also	ER, FR, SS, OSE

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command enables and disables the Stop on Stall function.

FSD1 = Enable Stop on Stall
FSDØ = Disable Stop on Stall

Entering FSD1 will cause the indexer to stop the move in progress when a stall is detected. The move is stopped immediately; no deceleration. This command is only valid if stall detection has been enabled (OSE1). It will have no effect otherwise.

Stall detection will work in either motor step mode (FSBØ) or encoder step mode (FSB1).

Entering FSDØ will cause the indexer to attempt to finish the move when a stall is detected, even if the load is jammed.

Command	Description
ER2ØØØ	Set encoder resolution to 2,000 steps/rev
OSE1	Enable stall detect function
FSD1	Enable stop on stall

FSE Enable Output #6 on Stall

Version A

Type	Set-Up
Syntax	aFSEn
Units	n = output, on/off
Range	0 or 1
Default	0
Response	None
See also	SS, ER, FR, FSF

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

FSEØ = Do not enable Output #6 on stall
FSE1 = Enable Output #6 on stall

Entering FSE1 will cause the indexer to turn on Output #6 when a stall is detected. This is useful for signaling other components in you system that a stall has occurred. This command will only be valid if Stall Detect (OSE1) has been enabled.

Output #6 is unaffected by a stall when FSEØ and OSE1 are entered.

Command	Description
ER4ØØØ	Set encoder resolution to 4,000 steps/rev
OSE1	Enable stall detect
FSE1	Turn on output number 4 when a stall is detected

FSF Enable Stop on Trigger #6

Version A

Type	Set-Up
Syntax	aFSFn
Units	n = function
Range	0 = off, 1 = on
Default	0
Response	None
See also	FR, TR, TS

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command enables and disables the Stop on Trigger function.

FSFØ = Do not terminate move on Trigger #6
FSF1 = Terminate move when Trigger #6 is high

Entering FSF1 will cause any move in progress to be stopped whenever Trigger #6 is brought high. For multiple axis application, setting up another unit to turn on Output #6 when it detects a stall, enables the user to implement a multi-axis stop on stall by connecting the output of one axis to the trigger of the other. The move will be decelerated at the maximum acceleration rate.

Entering FSFØ causes the indexer to treat Trigger #6 as a standard trigger input.

Command	Description
FSF1	Trigger #6 is now dedicated as a remote stop input

G Go

Version A

Type	Motion
Syntax	aG
Units	None
Range	None
Default	None
Response	None
See also	A, D, V, FSA, FSB, MA, MC, MN

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Go (G) command instructs the motor to make a move using motion parameters that you have previously entered. You do not have to re-enter Acceleration (A), Velocity (V), Distance (D), or the current mode (MN, MA, or MC) commands with each G command. In the Incremental Preset mode (MPI), a G command will initiate the steps you specified with the D command.

A G command in the Absolute Preset mode (MPA) will not cause repeated motion unless you enter a change in distance (D command).

In the Continuous mode (MC), you only need to enter the Acceleration (A) and Velocity (V) commands prior to the G command. The system ignores the Distance (D) command in this mode.

No motor motion will occur until you enter the G command in the Normal (MN), the Continuous (MC), or Alternating (MA) mode.

If motion does not occur with the G command, an activated end-of-travel limit switch may be on, the indexer is waiting for a trigger input (TR), or the indexer is in a pause state (PS, U) and waiting for a continue (C). Check the limit switches.

Command	Description
1MN	Set to Normal mode
FSAØ	Set to Incremental mode
FSBØ	Set Motor Step mode
1A5	Set acceleration to 5 rps^2
1A5	Set acceleration to 5 rps^2
1V5	Set velocity to 5 rps
1D25ØØØ	Set distance to 25,000 steps
1G	Execute the move (Go)
1A1	Set acceleration to 1 rps^2
1G	Execute the move (Go)

Assuming the indexer is in Incremental Preset mode, the motor turns 25,000 steps and repeats the 25,000-step move using the new acceleration value of 1 rps(Total distance moved = 50,000 steps).

Gnnn Go (Synchronized)

Version A

Type	Motion
Syntax	aGnnn
Units	n = axis
Range	None
Default	None
Response	None
See also	D, V, G, MN, I

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to put a special synchronized go command in each specified buffer. Each buffer will wait until all specified axis buffers have reached the synchronized go command. Each axis should start within 150 µsec of one another. Typically it is used to synchronize moves between more than one axis but it may also be used to synchronize two axes buffers (by issuing a zero distance move). The command may also be used to do simple multi-axis linear interpolation.

You should not send a new G or G123 command until motion is completed.

Command	Description
G12	Synchronize Axis #1 and #2 to start moving together

GA Go Home Acceleration

Version A

Type	Motion
Syntax	aGAn
Units	n = rps^2
Range	0.01 to 999.99
Default	10
Response	None
See also	GH, OS

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Go Home Acceleration (GA) command sets the linear acceleration value that the motor will use during any subsequent Go Home (GH) moves.

Command	Description
2GA5	Sets go home acceleration on axis #2 to 5 rps^2
2GH-5	The motor accelerates at 5 rps^2 to 5 rps in the CCW direction and searches for home

GH Go Home

Version A

Type	Motion
Syntax	aGHsn
Units	n = rps^2, s = ±
Range	0.001-20 (25,000 step/rev motor)
Default	
Response	None
See also	A, GA, OS, PZ

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Go Home (GH) command instructs the controller to search for the home position, either in the motor step mode or the encoder step mode. When in the motor step mode, the controller looks only at the Home Limit input. It will define Home as the CCW edge of the Home Limit signal (the edge closest to the CCW limit input).

The process is the same in encoder step mode, except the controller also looks for the Z channel input as well as the Home Limit input to be active at the same time. This means that the Z channel pulse must be *enveloped* by the active region of the Home Limit input. When both are active, the indexer defines that position as the home position.

The indexer will reverse direction if an end-of-travel limit is activated while searching for Home; however, if a second end-of-travel limit is encountered in the new direction, the Go Home procedure will stop and the operation will be aborted.

After the GH command is issued, the motor will run in the direction and velocity specified. The motor will keep running after the home switch is activated until it is deactivated. It will then decelerate and reverse direction.

The position counter is set to zero at the conclusion of the go home move.

Command	Description
GH-2Ø	The motor moves in the CCW direction at 20 rps and looks for the Home Limit input to go active

^H Backspace

Version A

Type	Programming
Syntax	^H
Units	None
Range	None
Default	None
Response	None
See also	None

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to delete the last character that you entered (unless it was a delimiter). The ^H command will not prevent execution of an immediate command. A new character may be entered at that position to replace the existing character. (^H indicates that the Ctrl key is held down when the H key is pressed.) This command prompts the indexer to backup one character in the command buffer, regardless of what appears on the terminal. On some terminals, the Ctrl and the left arrow <-- keys produce the same character. *Pressing the delete key does not delete the previous character.*

H Set Direction

Version A

Type	Motion
Syntax	<a>H<s>
Units	s = direction
Range	±
Default	+
Response	None
See also	None

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Set Direction (H) command changes or defines the direction of the next move that the system will execute. This command does not effect moves already in progress.

❑	H+	Sets move to CW direction
❑	H-	Sets move to CCW direction
❑	H	Changes direction from the previous setting
❑	CW	Direction of motor rotation when viewed from the front face

In preset moves, a Distance (D) command entered after the H command overrides the direction set by the H command. In Continuous mode (MC) only the H command can set the direction of motion.

Command	Description
MN	Set to Normal mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set distance to 25,000 steps
G	Execute the move (Go) in + direction
H	Reverse direction
G	Execute the move (Go) in - direction
MC	Set to Continuous mode
H+	Set direction to + direction
G	Move continuously in + direction

I Load Move Data

Version A

Type	Programming
Syntax	<a>I
Units	None
Range	None
Default	None
Response	None
See also	SSC, Gnnn

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Load Data (I) command allows the indexer to precalculate move data sent, so that execution of the move will begin within 10 ms of receiving a Go (G) command. The move profile may be repeatedly executed with only a 10 ms calculation delay until any one of the three move parameters: Acceleration (A), Velocity (V), or Distance (D) is changed.

Without use of the I command, a delay of up to 30 ms per axis will occur before execution of the move.

You must keep in mind that if you issue an Acceleration (A), Velocity (V), and Distance (D) command the PC23 will take up to 30ms to calculate the open-loop move profile. Therefore, the load data will be useful if you can send the I command, and go elsewhere to read data, then come back to execute the Go (G) command.

Command	Description
2MN	Set to Normal mode
2A5	Set acceleration to 5 rps^2
2V1Ø	Set velocity to 10 rps
2I	Load move data (precalculates move profile)
2TR1XXXXX	Wait for trigger #1 to go high
2G	Execute the move (Go)

IO Immediate Output

Version F

Type	Status
Syntax	aIOnnnnnnn
Units	None
Range	o = off, 1 = on, x = don't care
Default	None
Response	a:nnnnnnnn
See also	O, OS, SS

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Immediate Output (IO) command turns the programmable output bits (POBs) on and off. This can be used for signaling remote controllers, turning on LEDs, or sounding whistles. POB #1 is controlled by the first position after the IO, POB #2 is controlled by the second position, etc.

IS Input Status

Version A

Type	Status
Syntax	aIS
Units	None
Range	None
Default	None
Response	None
See also	LD, TR, TS

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command reports the status of the inputs for the specified axis. The response is 8 characters terminated with a carriage return. The first two characters are the axis number followed by a colon. The next characters are the status of 2 trigger bits, the CW limit, CCW limit, Home input and Z channels respectively.

Axis #1:	TRIG1, TRIG2, CW Limit 1, CCW Limit 1, HOME1, Z channel 1
Axis #2:	TRIG3, TRIG4, CW Limit 2, CCW Limit 2, HOME2, Z channel 2
Axis #3:	TRIG5, TRIG6, CW Limit 3, CCW Limit 3, HOME3, Z channel 3

Command	Response
`1IS`	`1:000011` Axis 1 has the home and z channel active
`2IS`	`2:100100` Axis 2 reports TRIG3 active and that its CCW limit is active
`3IS`	`3:010010` TRIG6 is active and the HOME switch is active

J Enable/Disable Joystick

Version A

Type	`Set-Up`
Syntax	`<a>Jn`
Units	`n = function`
Range	`0 = disable, 1 enable`
Default	`JØ = Immediate`
Response	`None`
See also	`JB, JD, JV, JZ, OSF`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to enter and exit the Joystick mode.

`JØ` = Disable Joystick Mode
`J1` = Enable Joystick Mode

When you enter the Joystick mode (`J1`), the command clears the buffer before entering the mode. The joystick is slew-rate limited by the last acceleration command. If the address preceding the command is the first to enter the Joystick mode, the differential signal between analog channel 0 and analog channel 1 will serve as the axis reference. In joystick mode, the last acceleration and velocity rates that you specify will be used to determine both axis' maximum velocity with full deflection and how rapidly the motor will track the joystick movements. High velocities will result in coarse velocity resolution. Compumotor recommends that you set the acceleration rate high to follow the joystick's movement as stiffly as possible.

The first axis to request a joystick receives channels Ø and 1. The second axis to request a joystick receives channels 2 and 3. To ensure proper assignment of an axis to the different channels, wait at least 25 ms between `J1` commands.

Command	Description
`1J1`	Axis 1 uses differential voltage Channels 0 & 1
`2J1`	Axis 2 uses differential voltage Channels 2 & 3

JB Set Joystick Backlash

Version B

Type	`Set-Up`
Syntax	`<a>JBn`
Units	`n = steps`
Range	`0 - 999`
Default	`0`
Response	`None`
See also	`JB, JD, JV, JZ`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to set the joystick backlash compensation distance in steps. If you change the direction of the joystick, a corrective move of `nnn` steps will be made in the new direction at the velocity you specified with the (Joystick Backlash Compensation Velocity (`JV`) command. This command overcomes the lag time that exists between changing the joystick's direction (when backlash is present) and actually moving in the new direction.

Command	Description
`1JV2`	Set the maximum backlash velocity to 2 rps
`1JB25Ø`	Set the backlash corrective move to 250 steps

JD Set Joystick Dead Band

Version B

Type	Set-Up
Syntax	<a>JDn
Units	n = volts
Range	0.000 to 0.500
Default	0.500
Response	None
See also	JB, JD, JV, JZ

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to set the the dead band value (in volts). You should set the joystick dead band value to a voltage that allows for the mechanical hysteresis in the resistive joystick assembly.

There will be no movement between zero point set by the JZ command and this dead band voltage.

Command	Description
1JZ	Set joystick's 0 point
1JD.Ø5	Set the joystick dead band 50mV above and below the joystick zero point.

JV Set Joystick Backlash Compensation Velocity

Version B

Type	Set-Up
Syntax	<a>JVn
Units	n = rps
Range	00.000 - 99.000
Default	0
Response	None
See also	JB, JD, JZ

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to set the joystick's backlash compensation peak velocity. This is the velocity at which the motor will travel when it corrects a position error (backlash). You can determine the distance of any joystick backlash move with the JB command.

Command	Description
1JV2	Set maximum backlash velocity to 2 rps
1JB25Ø	Set the joystick backlash move to 250 steps

JZ Set Joystick to Zero

Version B

Type	Set-Up
Syntax	<a>JV
Units	None
Range	None
Default	None
Response	None
See also	JB, JD, JV

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command allows you to initialize the current joystick position as the zero or no movement position. The position must be set with a differential channel voltage (restricted to 1.25 ±0.25V). If you send the command while the voltage is greater than the specified range, the PC23 will generate an error message. Use the following equations to calculate CCW and CW velocity resolution.

$$\text{CCW Velocity Resolution} = \frac{\text{Last Velocity Specified}}{\text{Zero Point} - 0.01}$$

$$\text{CW Velocity Resolution} = \frac{\text{Last Velocity Specified}}{2.5 - \text{Zero Point}}$$

Command	Description
1JZ	Set axis #1 zero velocity point
2JZ	Set axis #2 zero velocity point

K Kill Motion

Version A

Type	`Motion`
Syntax	`<a>K`
Units	None
Range	None
Default	None
Response	None
See also	S

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Kill (`K`) command is an emergency stop command and should only be used as such. This command causes indexing to cease immediately. There is *no* deceleration of the motor. If Kill causes the motor to slip (i.e., large loads at high speed), the load could be driven past limit switches and cause damage to the mechanism and possibly to the operation.

In addition to stopping the motor, the `K` command will terminate a loop, end a time delay and clear the command buffer.

Command	Description
`1A5`	Set acceleration on axis #1 to 5 rps^2
`1V2`	Set velocity to 2 rps
`1MC`	Set to Continuous mode
`1G`	Execute the move (Go)
•	
•	
•	
`1K`	Stop the motor instantly

L Loop

Version A

Type	`Programming`
Syntax	`<a>Ln`
Units	`n = number of loops`
Range	`0 = infinite, otherwise: 1 - 4,299,467,294`
Default	0
Response	None
See also	C, N, U, Y

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

When you combine the Loop (`L`) command with the End-of-Loop (`N`) command, all of the commands between `L` and `N` will be repeated the number of times indicated by n. If you enter the `L` command without a value specified for n, or with a Ø, subsequent commands will be repeated continuously.

The End-of-Loop command prompts the indexer to proceed with further commands after the designated number of loops have been executed. The Stop Loop (`Y`) command indicates where execution will stop. The Immediate Pause (`U`) command allows you to temporarily halt loop execution. You can use the Continue (`C`) command to resume loop execution. Nested loops are allowed up to 8 levels deep.

Command	Description
`L5`	Loop 5 times on axis #1.
`A5`	Set acceleration to 5 rps^2
`V1Ø`	Set velocity to 10 rps
`D1ØØØØ`	Set distance to 10,000 steps
`G`	Execute the move (Go)
`N`	Specifie the above 10,000-step move to be repeated five times

LA Limit Acceleration

Version F

Type	`Set-Up`
Syntax	`<a>LAn`
Units	`n = rps`^{2}
Range	`.01 - 999.99`
Default	`999.99`
Response	None
See also	A, LD

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Limit Acceleration (`LA`) command allows you to define the deceleration rate that should be used when an end-of-travel limit is encountered. This command is useful if you do not want an abrupt stop upon encountering a limit. However, you should be careful to specify a deceleration rate that will stop the load before it can do any damage. Normally, limit switches are placed so that the motor has room to safely decelerate the load.

Command	Description
`LA5Ø`	The motor on axis #1 decelerates at 50 rps^2 when it encounters an end-of-travel limit.

LD Limit Disable

Version A

Type	Set-Up
Syntax	<a>LDn
Units	n = enable/disable limits
Range	0 - 3
Default	3
Response	None
See also	RA

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved Independently
[ ] Saved in sequences

The Limit Disable (LD) command allows you to enable/disable the end-of-travel limit switch protection. The LDø condition does not allow the motor to turn without properly installing the limit inputs. If you want motion without wiring the limits, you must issue the LD3 command.

Enable CCW and CW limits	n = 0
Disable CW limits	n = 1
Disable CCW limits	n = 2
Disable CCW and CW limits	n = 3 (Default)

If you wire the limit switches, these switches will be ignored unless you enable the limit switch inputs using the LDø command.

Command	Description
1LDø	Enables CW and CCW limits on axis #1.
1LD3	Allows you to make any move, regardless of the limit input state.

MA Set Mode Alternate

Version A

Type	Motion
Syntax	<a>MAn
Units	None
Range	None
Default	None
Response	None
See also	A, D, V, G, MC, MN, SSD

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved In sequences

The Mode Alternate (MA) command, like the Mode Normal (MN) and the Mode Continuous (MC) commands, effects the way moves are preformed. In Mode Alternate (MA), a move is made to a position that you define with the Distance (D) command. When the motor reaches the specified distance it reverses direction and returns to the starting position. This cycle will continue until you issue a Stop (S) or Kill (K) command.

The way the motor stops when a Stop (S) command is issued can be set by using the SSD command. The default is so immediately stop.

Command	Description
MA	Set to Alternate mode
A5	Set acceleration to 5 rps^2
V1	Set velocity to 1 rps
D1øøø	Set distance to 1,000 steps
G	Execute the move (Go)

The motor makes a positive move 1,000 units in the CW direction. It then reverses direction and moves 1,000 steps in the CCW direction. This motion continues until you issue a Stop (S) or Kill (K) command.

MC Set Mode Continuous

Version A

Type	Motion
Syntax	<a>MCn
Units	None
Range	None
Default	None
Response	None
See also	A, G, H, MA, MN, T, TR, V

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Mode Continuous (MC) command causes subsequent moves to ignore any distance parameter and move continuously. You can clear the MC command with the Move Normal (MN) or the Mode Alternate (MA) command.

The indexer uses the Acceleration (A) and Velocity (V) commands to reach continuous velocity. The direction of the move should be specified with the H+ or H- command. Using the Time Delay (T), Trigger (TR), and Velocity (V) commands, you can achieve basic velocity profiling.

Command	Description
MC	Set to Continuous mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
T2	Move at 5 rps for 2 seconds
VØ	Change velocity to 0.00 rps
H+	Set move to CW
G	Execute the move (Go)

The motor turns CW at 5 rps until it is halted by the Stop (S) command, Kill (K) command, a limit switch, or by a new velocity specification (VØ followed by a G command).

MN Set Mode Normal

Version A

Type	Motion
Syntax	<a>MN
Units	None
Range	None
Default	None
Response	None
See also	A, D, V, G, MC, MPI, MPA

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Mode Normal (MN) command sets the system to mode normal. In Mode Normal, the motor moves the distance specified with the distance (D) command. To define the complete move profile, you must define Acceleration (A), Velocity (V), and Distance (D). The MN command is used to change the mode of operation from Mode Continuous (MC) or Mode Alternating (MA) to the preset mode.

Command	Description
2MN	Set to Normal mode on axis #2
2A5	Set acceleration to 5 rps^2
2V5	Set velocity to 5 rps
2D1ØØØ	Set distance to 1,000 steps
2G	Execute the move (Go)

MPA Set Position Absolute Mode

Version A

Type	Set-Up
Syntax	<Ma>MPA
Units	None
Range	None
Default	None
Response	None
See also	D, FSA, MC, MN, MPI

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the positioning mode to absolute. In this mode all move distances are referenced to absolute zero. Note that in Mode Absolute (MPA or FSA1), giving two consecutive go (G) commands will cause the motor to move once, since the motor will have achieved its desired absolute position at the end of the first move.

Mode Position Absolute (MPA) is most useful in applications that require moves to specific locations.

You can set the absolute counter to zero by cycling power or issuing a Position Zero (PZ) command.

Command	Description
MN	Set to Normal mode
MPA	Set to Absolute Position mode
A5	Set acceleration to 5 rps^2
PZ	Set absolute counter to zero
V1Ø	Set velocity to 10 rps
D25ØØØ	Set distance to 25,000 steps
G	Move motor to absolute position 25,000
D125ØØ	Set absolute position to +12,500 steps
G	Move motor to absolute position +12,500 steps

MPI Set Position Incremental Mode

Version A

Type	`Set-Up`
Syntax	`<a>MPI`
Units	None
Range	None
Default	None
Response	None
See also	`D, FSA, MN, MPA`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the positioning mode to incremental. In incremental mode, all move distances specified with the Distance (`D`) command are referenced to the current position. Position Incremental mode (`MPI` or `FSAØ`) is useful in applications that require repetitive movements, such as feed-to-length applications.

Command	Description
`1MN`	Set to Normal mode
`1MPI`	Set to Positioning Incremental mode
`1A5`	Set acceleration to 5 rps^2
`1V1Ø`	Set velocity to 10 rps
`1D1ØØØØ`	Set distance of move to 10,000 steps
`1G`	Move 10,000 steps CW
`1G`	Move another 10,000 steps CW

MR Select Motor Resolution

Version A

Type	`Set-Up`
Syntax	`<a>MRn`
Units	`n = resolution codes`
Range	`0 - 46`
Default	`10`
Response	None
See also	`A, V, ER`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Motor Resolution (`MR`) command sets the number of steps per revolution according to the numeric suffix n (see table below). This command allows the indexer to control drives of different resolutions while maintaining the commanded acceleration and velocity.

Motor Resolution	Velocity Max (rps)	Pulse W. (µsec)	PC23 command
200	160	15	MR0
400	80	15	MR1
800	78	7.5	MR2
1000	100	4	MR3
1600	60	4	MR4
3200	78	2	MR5
5000	100	1	MR6
6400	78	1	MR7
10000	50	1	MR8
21600	23	1	MR9
25000	20	1	MR10
25400	19.5	1	MR11
36000	13.8	1	MR12
50000	10	1	MR13
50800	9.8	1	MR14
4096	122	1	MR15
12800	39	1	MR16
25600	19.5	1	MR17
12500	40	1	MR18
16384	30	1	MR19
20000	25	1	MR20
25000	80	.25	MR21
50000	40	.25	MR22
100000	20	.25	MR23
101600	19	.25	MR24
63488	15	.50	MR25
76800	13	.50	MR26
81920	24	.25	MR27
102400	19	.25	MR28
126976	15	.25	MR29
128000	15	.25	MR30
153600	13	.25	MR31
163840	12	.25	MR32
204800	9.7	.25	MR33
253952	7.8	.25	MR34
256000	7.8	.25	MR35
307200	1.6	1	MR36
327680	3.0	.50	MR37
409600	2.4	.50	MR38
507904	1.9	.50	MR39
521000	1.9	.50	MR40
614400	1.6	.50	MR41
655360	3.0	.25	MR42
819200	2.4	.25	MR43
1024000	1.9	.25	MR44
2000	50	4	MR45
4000	125	1	MR46

The motor resolution you select does not determine the resolution of the motor. The motor/drive determines the resolution of the system. You use the `MR` command to match the motor/drive to your system, so that you can program your acceleration (`A`), and Velocity (`V`) in revolutions.

Command	Description
`MN`	Set to Normal mode
`MR1`	Set motor resolution to 400 steps/rev
`A5`	Set acceleration to 5 rps^2
`V1Ø`	Set velocity to 10 rps
`D8ØØ`	Set distance of move to 800 steps
`G`	Execute the move (Go)

A 400 step per revolution motor/drive will turn 800 steps (two revs) CW at an acceleration of 10 rps^2 and a velocity of 10 rps after the `G` command.

If this command set is sent to a motor/drive with a resolution of 4,000, the motor will still turn 800 steps (1/5 of a rev). However, the actual acceleration would only be 0.5 rps^2 and the actual velocity would only be 1 rps. Indexer resolution and motor/drive resolution must match to get the commanded velocity and acceleration. **This command does NOT affect distance.**

MSL Identify Clock Source for Timed Data Streaming Mode

Version A

Type	`Set-Up`
Syntax	`<a>MSLn₁n₂n₃`
Units	`Value for each`
Range	`0, 1, 2, 3, or x`
Default	`None`
Response	`None`
See also	`MSS, TD, SD, QØ, Q2, Q3`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Identify Clock Source for Timed Data Streaming Mode (`MSL`) command sets the Source/Receiver relationship for clock source for each axis in the timed data streaming mode. The following table defines the relationship for each axis. Any axis not in timed data streaming mode can still receive and execute other motion commands.

n	Value of n	Mode
n_1	Ø	Axis #1 is a slave and uses the external rmclk
n_1	1	Axis #1 is a master and uses its internal rmclk
n_1	2	Axis #1 is a slave and uses axis #2's rmclk
n_1	3	Axis #1 is a slave and uses axis #3's rmclk
n_1	X	The X axis is not in the timed data streaming mode and no master/slave specification is necessary
n_2	Ø	Axis #2 is a slave and uses the external rmclk
n_2	1	Axis #2 is a slave and uses axis #1's rmclk
n_2	2	Axis #2 is a master and uses its internal rmclk
n_2	3	Axis #2 is a slave and uses axis #3's rmclk
n_2	X	Axis #2 is not in the timed data streaming mode and no master/slave specification is necessary
n_3	Ø	Axis #3 is a slave and uses the external rmclk
n_3	1	Axis #3 is a slave and uses axis #1's rmclk
n_3	2	Axis #3 is a slave and uses its internal rmclk
n_3	3	Axis #3 is a master and uses axis #3's rmclk
n_3	X	Axis #3 is not in the timed data streaming mode and no master/slave specification is necessary

Command	Description
`MSL12X`	Axes #1 and #2 are in Timed Data Streaming mode. Both are masters, Axis #3 is independent.
`MSLØXX`	Axis #1 is In Timed Data Streaming modeAxis #1 is a slave to the external RMCLK.

MSS Start Master Clock for Timed Data Streaming Mode

Version A

Type	`Programming`
Syntax	`<a>MSS`
Units	`None`
Range	`None`
Default	`None`
Response	`None`
See also	`MSL, TD, SD, QØ, Q2, Q3`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Start Master Clock (`MSS`) command will cause the clocks on any master axes to start (Initiate Motion). If Profiling (`SD`) commands are stored in the buffer, entering the `MSS` command causes execution of these moves. Subsequent Streaming Data (`SD`) commands will be executed sequentially in a buffered manner. Care must be taken to not overflow these buffers. If the buffer is empty, no motion will result during the particular update interval.

Command	Description
`1Q2`	Set axis #1 to Time Distance mode
`1TDØ6`	Set update interval to 6 ms
`MSL1XX`	Set axis #1 as a Master, using its internal rate multiplier clock. Axis #2 and #3 are not in the time distance mode.
`SD8Ø28`	Set distance to be moved during the first update interval.
`SD8Ø45`	Set distance to be moved during the next update interval
`SD8Ø45`	Set distance to be moved during the next update interval.
`SD8Ø28`	Set distance to be moved during the next update interval.
`SD8Ø1D`	Set distance to be moved during the next update interval.
`MSS`	Start Master Clock

MV Set Maximum Correction Velocity

Version A

Type	`Set-Up`
Syntax	`<a>MVn`
Units	`n = rps`
Range	`0.01 - 20.00`
Default	`0.2`
Response	`None`
See also	`CG, CM, ER, FS`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

If the motor is running with the encoder feedback (`FSB1`), and position maintenance is turned on `FSC1`, this command will specify the maximum velocity for correction moves.

The correction move is performed after the preset move is finished, and only when the motor is positioned outside the dead band.

Command	Description
`MN`	Set to Normal mode
`MV2`	Set the maximum correction velocity to 2 rps
`FSB1`	Set motion in encoder mode
`FSC1`	Set position maintenance on

N End of Loop

Version A

Type	`Programming`
Syntax	`<a>N`
Units	`None`
Range	`None`
Default	`None`
Response	`None`
See also	`C, L, PS, Y`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command marks the end of loop. Use this command in conjunction with the Loop (`L`) command. All buffered commands that you enter between the `L` and `N` commands are executed as many times as the number that you enter following the `L` command. You may nest loops 8 levels deep.

Command	Description
`PS`	Pause the execution of buffered commands
`MN`	Set to Normal mode
`A5`	Set acceleration to 5 rps^2
`V5`	Set velocity to 5 rps
`D1ØØØØ`	Set move distance to 10,000 steps
`L5`	Loop the following commands five times
`G`	Execute the move (Go)
`N`	End the loop
`C`	Clear pause and executes all the buffered commands

O Output

Version A

Type	`Programming`
Syntax	`<a>Onnnnnn`
Units	`n = output`
Range	`0 = off, 1 = on, x = don't change`
Default	`None`
Response	`None`
See also	`IO, FSE`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Output (`O`) command turns the programmable output bits (POBs) on and off. This is used for signaling remote controllers, turning on LEDs, or sounding whistles. The output can indicate that the motor is in position, about to begin its move, or is at constant velocity, etc. POB #1 is controlled by the first position after the `O`, POB #2 is controlled by the second position, etc.

Command	Description
`A5`	Set acceleration to 5 rps^2
`V5`	Set velocity to 5 rps
`D2ØØØØ`	Set move distance to 20,000 steps
`OØ1XXXX`	Set programmable output 1 off and output 2 on (outputs 3 through 6 are the same)
`G`	Execute the move (Go)
`OØØXXXX`	After the move ends, turn off output 2

O,D Output on the Fly

Type	Programmable	**Version**
Syntax	<a>Op,Dn (see below)	**Attributes**
Units	see below	[x] Buffered
Range	see below	[ ] Device specific
Default	None	[ ] Saved independently
Response	None	[ ] Saved in sequences
See also	O, D	

Syntax:	<axis>Opppppp,Dnnnnnn Where p = bit pattern and n = distance
Units:	p = O (off), 1 = (on), or X (don't change) for each of the programmable output bits. n = steps
Range:	<a> = axis 1, 2, or 3.
	p = any combination of Ø's, 1's, or x's for a total of 6 bits.

The O,D command allows the user to program a specified bit pattern to appear on the outputs, at a specified distance.

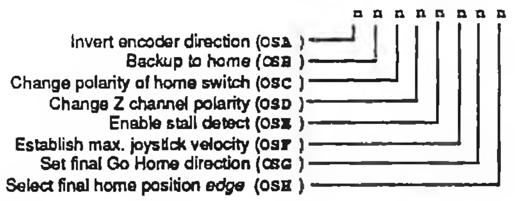
Command	Description
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set distance to 25,000 steps
O101010	Output the pattern 101010
OXXXXX1,D15ØØØ	Change the output LSB to a 1 at 15,000 steps
G	Execute the move (Go)
O1XXXXX	Change the MSB to a 1 when the move is complete

The pattern 101010 will appear and remain on the outputs, prior to the move. When 15,000 steps have been generated the output pattern will change to 101011. When the move is complete, the pattern will change to 001011.

OR Report Function Set-Ups

Type	Status	**Version** A
Syntax	aOR	**Attributes**
Units	None	[x] Buffered
Range	None	[ ] Device specific
Default	None	[ ] Saved independently
Response	a:nnnnnnnn (a = axis number, n = 0, 1)	[ ] Saved in sequences
See also	OS	

This command results in a report of which software switches have been set by the OS command. The reply is eight digits. This command reports OSA through OSF setup status in binary format.


n n n n n n n n
Invert encoder direction (OSA)
Backup to home (OSB)
Change polarity of home switch (OSC)
Change Z channel polarity (OSD)
Enable stall detect (OSE)
Establish max. joystick velocity (OSF)
Set final Go Home direction (OSG)
Select final home position *edge* (OSH)

The digits (n) correspond to OSA through OSH commands (left to right). One (1) means an OS function is on.

OSA:	Encoder Direction = Normal (Ø), = Inverted (1).
OSB:	Backup to Home = Disabled (Ø), = Enabled (1).
OSC:	Active State of Home Input = High (Ø), = Low (1).
OSD:	Active State of Z Channel = High (Ø), = Low (1).
OSE:	Enable Stall Detect = Disabled (Ø), = Enabled (1).
OSF:	Set Max. Joystick Velocity = Use max. velocity (Ø), = Use last specified velocity (1).
OSG:	Final Go Home Direction = CW (Ø), = CCW (1).
OSH:	Select Final Home Position Edge = CW (Ø), = CCW (1).

Command	Description
1OSA1	Inverts the encoder direction for axis #1.
1OR	Response is 1:1ØØØØØØØ

OSA Set Encoder Direction

Version A

Type	Set-Up
Syntax	<a>OSAn
Units	None
Range	0 or 1
Default	0
Response	None
See also	FSB, OR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Entering OSA1 inverts the up/down count direction of the encoder. It causes the motor to expect encoder pulses back in the opposite direction of the motor pulses being sent out. This is useful if the encoder is on the load and is turning in the opposite direction of the motor due to the mechanical configuration of the system. A typical symptom of the encoder counting in the opposite direction is, if the motor drifts at a low velocity in encoder mode (FSB1).

Entering OSAø will change the up/down count of the encoder back to normal if it has been inverted with the OSA1 command.

Command	Description
1OSA1	Invert the encoder direction for axis #1.
FSB1	Enable encoder mode

OSB Backup to Home Switch

Version A

Type	Set-Up
Syntax	<a>OSBn
Units	None
Range	0, 1
Default	1
Response	None
See also	FS, GH, OR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

OSBø = Do not back up to home switch
OSB1 = Back up to home switch

If this function is disabled (OSBø), the PC23 will consider the motor at Home if the home input is active at the end of deceleration after encountering the active edge of Home region. If this function is enabled (OSB1), the PC23 will decelerate the motor to a stop after encountering the active edge of the Home region, and then move the motor in the opposite direction of the initial Go Home move at .1 rev/sec until the active edge of the Home region is encountered. The PC23 will then consider the motor at Home. This will occur regardless of whether or not the home input is active at the end of the deceleration of the initial Go Home move.

OSC Define Active State of Home Switch

Version A

Type	Set-Up
Syntax	<a>OSCn
Units	n = active state, high/low
Range	0 or 1
Default	0
Response	None
See also	GH, OSD, OR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

OSCø = Active state of home input is high
OSC1 = Active state of home input is low

This command inverts the active state of the home input. It enables you to use either a normally closed or a normally open switch for homing. OSCø requires that a normally closed switch be connected to the home limit input.

OSC1 requires that a normally open switch be connected to the home limit input.

Command	Description
OSC1	Set the active state of the home input to low

OSD Define Active State of Z Channel

Version A

Type	`Set-Up`
Syntax	`<a>OSDn`
Units	`n = active state high/low`
Range	`0 or 1`
Default	`0`
Response	`None`
See also	`GH, OSC, OR`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`OSDØ` = Active state of Z Channel is high
`OSD1` = Active state of Z Channel is low

This command inverts the active state of the encoder Z channel input. `OSDØ` requires that the Z Channel input be an active high input. `OSD1` requires that the encoder Z Channel input be an active low.

The Z channel input can be used for encoder based homing (`GH`) only. In encoder mode, you must have the home switch and the Z channel active at the same time to successfully home the motor.

Command	Description
`OSD1`	Set the active state of the encoder Z Channel to low

OSE Enable Stall Detect

Version A

Type	`Set-Up`
Syntax	`<A>OSEn`
Units	`n = enable/disable`
Range	`0 or 1`
Default	`0`
Response	`None`
See also	`FSB1, FSC1, FSD, FDE, OR OSH`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`OSEØ` = Stall detect disabled
`OSE1` = Stall detect enabled

When enabled (`OES1`), a stall is detected if the indexer receives no encoder pulses after moving the motor 1/50th of a rev (corresponds to 1 mechanical pole of a 50-pole motor). Thus, when the indexer and motor/drive system resolution is 25,000 steps/rev, 500 steps (25,000 ÷ 50) are output before a stall is detected.

To stop motor on a stall (`FSD1`), or to turn on output on stall (`FSE1`), enable stall detection (`OSE1`).

Command	Description
`OSEØ`	Disable Stall Detection

OSF Establish Maximum Joystick Velocity

Version A

Type	`Set-Up`
Syntax	`<a>OSFn`
Units	`None`
Range	`0 or 1`
Default	`1`
Response	`None`
See also	`OR, V`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`OSFØ` = Use the maximum system velocity as the joystick maximum velocity
`OSF1` = Use the previously defined velocity (`V`) as the joystick maximum velocity

OSG Set Final Go Home Direction

Version A

Type	`Set-Up`
Syntax	`<a>OSGn`
Units	`None`
Range	`0, 1`
Default	`1`
Response	`None`
See also	`GH, OR, OSB, OSC`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`OSGØ` = Sets the final portion of the go home move sequence to CW
`OSG1` = Sets the final portion of the go home move sequence to CCW

OSH Reference Edge of Home Switch

Version A

Type	Set-Up
Syntax	`<a>OSHn`
Units	None
Range	0, 1
Default	1
Response	None
See also	OR, OSG

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`OSHØ` = Selects the CW side of the Home signal as the *edge* on which the final approach will stop
`OSH1` = Selects the CCW side of the home signal as the *edge* on which the final approach will stop

The CW edge of the Home switch is defined as the first switch transition seen by the indexer when traveling off of the CW limit in the CCW direction. If n = 1, the CCW edge of the Home switch will be referenced as the Home position. The CCW edge of the Home switch is defined as the first switch transition seen by the indexer when traveling off of the CCW limit in the CW direction.

P Report Incremental Position

Version A

Type	Status
Syntax	`aP`
Units	None
Range	None
Default	None
Response	`a:±nnnnnnnn (n = 0 - 9)`
See also	PB, PR, PX, PXB, W1, W3

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Reports incremental distance traveled during the last move in decimal format. The command reports positions in an 8-digit number preceded by sign (+/-), and followed by a carriage return. The range for the position report is 0 - ±99,999,999. If the indexer is in the encoder mode, the report is in encoder steps. If the indexer is in motor mode, the report is in motor steps.

Command	Description
`MN`	Set to Normal mode
`FSBØ`	Set to Motor Step mode
`A1Ø`	Set acceleration to 10 rps^2
`V5`	Set velocity to 5 rps
`D25ØØØ`	Set move distance to 25,000 steps
`G`	Execute the move (Go)
`1P`	Request Position
	Response = `1:+ØØØ25ØØØ`

PB Report Incremental Position (Binary)

Version A

Type	Status
Syntax	`aPB`
Units	None
Range	None
Default	None
Response	`nnnnn`
See also	W1, W3, P, PR, PX, PXB

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command reports back the encoder's (in `FSB1` mode) or the motor's (in `FSBØ` mode) incremental position in binary mode. The binary response corresponds to a decimal range of 0 - ±2,147,483,647. The response format is five bytes ($n_1n_2n_3n_4n_5$). The first byte (n_1) is the axis number (Axis 1 - 3). The next four bytes ($n_2n_3n_4n_5$) are the most significant bit to the least significant bit in 2's compliment notation. These four bytes give the 32-bit encoder position.

Converting 2's Compliment to Decimal

If the most significant byte is in the range `ØØ` - `7F`, the result is positive. If the most significant byte is in the range `8Ø` - `FF`, the result is negative.

For positive results, use the following conversion procedure:

① Convert each byte (ASCII character) to decimal.

② Multiply each decimal number by the following:

n_2 x 16,777,216
n_3 x 65,536

n_4 x 256
n_5 x 1

③ Add together the products of step 2.

Response from the 1PB command is as follows:

$$\underbrace{\text{Ø1}}_{n_1}\underbrace{\text{ØØ}}_{n_2}\underbrace{\text{ØØ}}_{n_3}\underbrace{\text{61}}_{n_4}\underbrace{\text{A8}}_{n_5}$$

① n_2 = ØØ hex = ØØ decimal
n_3 = ØØ hex = ØØ decimal
n_4 = 61 hex = 97 decimal
n_5 = A8 hex = 168 decimal

② n_2 (ØØ) x 16,777,216 = Ø
n_3 (ØØ) x 65,536 = Ø
n_4 (97) x 256 = 24,832
n_5 (168) x 1 = 168

③ Ø + Ø + 24,832 + 168 = 25,000. Thus, the last move was 25,000 steps in the CW direction.

Binary Approach

① Convert the hexadecimal response to binary form.
② Complement the binary number.
③ Add 1 to the binary number.
④ Convert the binary result to a decimal value.

The response from the 1PB command is Ø1EFFE271Ø. Ø1 is the axis number and EFFE271Ø is the negative incremental position report.

1	2	3	4
E = 111Ø	111Ø = ØØØ1	ØØØ1 = ØØØ1	ØØØ1 = 1
F = 1111	1111 = ØØØØ	ØØØØ = ØØØØ	ØØØØ = Ø
F = 1111	1111 = ØØØØ	ØØØØ = ØØØØ	ØØØØ = Ø
E = 111Ø	111Ø = ØØØ1	ØØØ1 = ØØØ1	ØØØ1 = 1
2 = ØØ1Ø	ØØ1Ø = 11Ø1	11Ø1 = 11Ø1	11Ø1 = D
7 = Ø111	Ø111 = 1ØØØ	1ØØØ = 1ØØØ	1ØØØ = 8
1 = ØØØ1	ØØØ1 = 111Ø	111Ø = 1111	111Ø = F
Ø = ØØØØ	ØØØØ = 1111	1111 = ØØØØ	1111 = Ø

1ØØ1D8FØ hex = 268,556,528 decimal. The last move was 268,556,528 steps in the CCW direction.

Computer Approach

① Convert the hexadecimal number to a decimal number.
② Subtract 168 (= 232 = 4,294,967,296) from the decimal number derived from step 1.

The response from the 1PB command is Ø1EFFE271Ø. Ø1 is the axis number and EFFE271Ø is the negative incremental position report.

① EFFE271Ø hex = 4,026,410,768 decimal
② 4,026,410,768 - 4,294,967,296 = 268,556,528
The last move was 268,556,528 steps in the CCW direction.

PR Report Absolute Position

Version A

Type	Status
Syntax	aPR
Units	None
Range	None
Default	None
Response	a:±nnnnnnnn (n = 0 - 9)
See also	D, MN, MPA, MPI, P, PB, PZ, W

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved Independently
[ ] Saved In sequences

This is an absolute position counter. It reports motor position with respect to power-up position. The absolute position counter can track up to ± 99,999,999 steps. If the counter is overrun in the relative position mode, the absolute position will be invalid.

If in the encoder mode, position will be reported in encoder steps. If you are in motor mode, position will be reported in motor steps. In preset mode, response to this command will be reported after the move is done. In continuous mode, the response to this command will be reported after the motor reaches constant velocity.

The Position Report (PR) command responds with the cumulative position of the motor with respect to the zero position. The zero position can be defined by the position of the motor after a Position Zero (PZ) command is issued or by the successful completion of a Go Home (GH) command.

This command can only respond when the motor is not being commanded to move. Should you need relative positional information when the motor is moving, see the Report Position Relative to Start of Current Move (W) command.

Command	Description
PZ	Set current position to absolute zero
MN	Set to Normal mode
FSBØ	Set to Motor Step mode
A1Ø	Set acceleration to 10 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set move distance to 25,000 steps
G	Execute the move (Go)
G	Execute the move (Go)
1PR	Request Position Response = 1:+ØØØ5ØØØØ

The motor on axis #1 will move 50,000 motor steps, then the indexer will report the distance moved since the last PZ command.

PS Pause

Version A

Type	Programming
Syntax	<a>PS
Units	None
Range	None
Default	None
Response	None
See also	C, U

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command pauses execution of a command string or sequence following the Pause (PS) command until the indexer receives a Continue (C) command. This command is useful if you need to enter a complete string of commands before you can execute your other commands.

This command is useful for interactive tests and in synchronizing multiple indexes that have long command strings.

Command	Description
PS	Pause execution of following commands until axis # 1 receives the Continue (C) command
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set move distance to 25,000 steps
G	Execute the move (Go)
T2	Delay the move for 2 sec
G	Execute the move (Go)
C	Continue Execution

When axis #1 receives the C command, the motor moves 25,000 steps twice with a two second delay between moves.

PX Report Encoder Absolute Position (ASCII)

Version A

Type	Status
Syntax	aPX
Units	None
Range	None
Default	None
Response	a:±nnnnnnnn (n = 0 - 9)
See also	W1, W3, P, PB, PR, PXB

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command reports encoder position with respect to power-up position. The absolute position counter can track up to ±99,999,999 encoder steps. If the counter is overrun in the relative position mode, the absolute position will be invalid.

Whether in encoder step or motor step mode position will be reported in encoder steps. In preset mode, response to this command will be reported after the move is done. In continuous mode, the response to this command will be reported after the motor reaches constant velocity.

The Position Report (PX) command responds with the cumulative position of the encoder with respect to the zero position. The zero position can be defined by the position of the encoder after a Position Zero (PZ) command is issued or by the successful completion of a Go Home (GH) command.

This command can only respond when the motor is not being commanded to move. Should you need relative positional information when the motor is moving, see the Report Position Relative to Start of Current Move (W) command or the absolute encoder position binary (PXB)command below.

Command	Description
MN	Set to Normal mode
A1Ø	Set acceleration to 10 rps^2
V5	Set velocity to 5 rps
D4ØØØ	Set move distance to 4,000 encoder steps
G	Execute the move (Go)
1PX	Request position of axis 1
	Response = 1:+ØØØØ4ØØØ

Axis 1# moves move 4,000 encoder steps, then the indexer reports the distance that axis #1 just moved.

PXB Report Encoder Absolute Position (Binary)

Version B3

Type	Status
Syntax	aPXB
Units	None
Range	None
Default	None
Response	nnnnn
See also	W1, W3, PB, PR, PX

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command reports back the encoder absolute position in binary mode. The binary response corresponds to a decimal range of 0 - ±2,147,483,647. The response format is five bytes ($n_1n_2n_3n_4n_5$). The first byte (n_1) is the axis number (Axis 1 - 3). The next four bytes ($n_2n_3n_4n_5$) are the most significant bit to the least significant bit in 2's compliment notation. These four bytes give the 32-bit encoder position. Refer to the PB command for procedures and examples of how to convert the binary position report.

PZ Set Zero Position

Version A

Type	Status
Syntax	<a>PZ
Units	None
Range	None
Default	None
Response	None
See also	D, MN, MPI, MPA, PR, GH

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the absolute position counter to zero. When your indexer powers up, the absolute counter sets to zero. After moving the motor, the PZ command is used to reset the absolute counter to zero.

In Absolute Mode (MPA), all the moves will be made with respect to the absolute counter. When you execute this command, the Position Report (PR), Pause (P), and Report Absolute Encoder Position (PX) commands will report the zero position.

Command	Description
PZ	Set the absolute counter to zero
MPA	Set to Preset mode with respect to Absolute zero position
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØ	Set move distance to 2,500 steps
G	Execute the move (Go)
1PR	Report Absolute Position Response = 1:+ØØØ25ØØØ
PZ	Set the absolute counter to zero
1PR	Report absolute position Response = 1:+ØØØØØØØØ

Q Complete Current Command and Clear Buffer

Version A

Type	Programming
Syntax	`<a>Q`
Units	None
Range	None
Default	None
Response	None
See also	K, S

Attributes
- [ ] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved in sequences

The Q command completes the current command being executed and clears the remainder of the command buffer on the specified axis.

This command is useful if you do not wish to interrupt the current move, or any other command being executed.

The Kill (K) command will stop any command being executed and clears the remainder of the commands in the buffer on the specific axis.

Command	Description
Q	Completes current command and clears command buffer

QØ Exit Streaming Mode

Version A

Type	Set-Up
Syntax	`<a>QØ`
Units	None
Range	None
Default	None
Response	None
See also	Q1, Q2, Q3, RM

Attributes
- [ ] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved in sequences

The QØ command is an exit command for all streaming modes. The motor will stop when QØ is issued.

Refer to Q1 and Q2 examples.

Q1 Enter Immediate Velocity Streaming Mode

Version A

Type	Set-Up
Syntax	`<a>Q1`
Units	None
Range	None
Default	None
Response	None
See also	QØ, RM

Attributes
- [ ] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved In sequences

The Q1 command enters the immediate velocity streaming mode. *The command buffer is cleared and any motion is killed on the specified axis.* Subsequent RM commands will cause an immediate change in motor velocity. Use QØ to exit this mode.

Command	Description
Q1	Enter Velocity Profiling mode
RMØØ11	Go to RM velocity of (11 hex) RM rps
RMØØ55	Go to RM velocity of (55 hex) RM rps
RMØ1ØØ	Go to RM velocity of (100 hex) RM rps
RMØØ55	Go to RM velocity of (55 hex) RM rps
RMØØ11	Go to RM velocity of (11 hex) RM rps
QØ	Exit Velocity Profiling mode

Motor movement will stop when the QØ command is entered. See RM command for more details.

Q2 Enter Time-Distance Streaming Mode

Version A

Type	Set-Up
Syntax	`<a>Q1`
Units	None
Range	None
Default	None
Response	None
See also	Q∅, MSL, MSS, TD, SD

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Enter Time-Distance Streaming mode (Q2) command causes the indexer to enter the Time-Distance Streaming mode for the specified axis. The Time-Distance Streaming mode will interpret values entered with the SD command as the number of motor steps to be output in the update interval specified by the TD command. *The command buffers are cleared and motion is killed on the specified axis when the Q2 command is issued.*

This mode is useful in applications that require multi-axis contouring, synchronization, or custom move profiles.

Refer to *Chapter 4, Application Design*, for a detailed discussion of the Time-Distance Streaming mode.

Q3 Enter Time Velocity Streaming Mode

Version A

Type	Set-Up
Syntax	`<a>Q1`
Units	None
Range	None
Default	None
Response	None
See also	Q∅, MSL, MSS, TD, SD

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Enter Time-Velocity Streaming mode (Q3) command causes the indexer to enter the Time-Velocity Streaming mode for the specified axis. The Time-Velocity Streaming mode will interpret values entered with the SD command as the velocity in motor steps to be output in the update interval specified by the TD command. *Note that the command buffers are cleared and motion is killed on the specified axis when the Q3 command is entered.*

Refer to *Chapter 4, Application Design*, for a detailed discussion of the Velocity Streaming mode.

QI Interrupt Status Report

Version A

Type	Set-Up
Syntax	`<a>QI`
Units	None
Range	None
Default	None
Response	nnnnnnnn (n = 0 or 1)
See also	QS

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

QI command indicates any active interrupt conditions. The response is the address followed by 8 bits (A through H) as described in the table below. The bits do not need to be enabled with the QS commands. For example, bit A will be high if trigger 1 is active, whether or not QSA1 was entered.

Bit	Function	Options	Value
A	Trigger 1 active	NO (∅) or YES (1)	n
B	Move Complete	NO (∅) or YES (1)	n
C	*Not Used*	∅	∅
D	Limit Encountered	NO (∅) or YES (1)	n
E	Ready to Respond*	∅	0
F	*Not Used*	∅	∅
G	Command Buffer Full	NO (∅) or YES (1)	n
H	Motor Stall	NO (∅) or YES (1)	n

*The ODBRDY bit in the status register is set. Refer to *Chapter 4, Application Design*, for more on communication.

Move complete will always display a result, even if the QSB command is not enabled.

Command	Response
1QI	Request source interrupt status for axis #1. Response = 1:∅1∅∅∅∅∅∅

QIB Interrupt Status Report (Binary)

Version A

Type	Set-Up	**Attributes**
Syntax	<a>QIB	[ ] Buffered
Units	None	[ ] Device specific
Range	None	[ ] Saved independently
Default	None	[ ] Saved in sequences
Response	a:n	
See also	QI	

The Interrupt Status Report (QIB) command responds with two bytes. The first byte is axis number (1-3). The second byte is the binary equivalent of the QI response (range is 0 - 255). The format for the second byte functions are shown below.

Since the response in in the binary format, the user must convert the binary value to ASCII value of you wish to print the value to the monitor.

Function	Options		Value
A Trigger 1 active	NO/YES	(n = 0,1)	n
B Move Complete	NO/YES	(n = 0,1)	n
C Not Used		(0)	0
D Limit Encountered	NO/YES	(n = 0,1)	n
E Ready to Respond*		(0)	0
F Not Used		(0)	0
G Command Buffer Full	NO/YES	(n = 0,1)	n
H Motor Stall	NO/YES	(n = 0,1)	n

*The ODBRDY bit in the status register is set. Refer to *Chapter 4, Application Design* for more on communication.

Move complete will always display a result, even if the QSB command is not enabled.

QR Report QS Command Function Enable Status

Version A

Type	Status	**Attributes**
Syntax	aQR	[x] Buffered
Units	None	[ ] Device specific
Range	None	[ ] Saved independently
Default	None	[ ] Saved in sequences
Response	a:nnnnnnnn (n = 0 or 1)	
See also	QS, QI	

Request for the functions enabled or disabled by the QSA through QSH commands. The reply is eight digits, indicating which signal functions are active, as shown below:

Function	Options		Value
A Trigger 1 high	OFF/ON	(0)	n
B Move Complete	OFF/ON	(n = 0,1)	n
C Not Used		(0)	0
D Limit Encountered	OFF/ON	(n = 0,1)	n
E Ready to Respond		(0)	0
F Not Used		(0)	0
G Transmit on Command Buffer Full	OFF/ON	(n = 0,1)	n
H Motor Stall	OFF/ON	(n = 0,1)	n

QS Interrupt on Signal Commands

Version A

Type	Set-Up	**Attributes**
Syntax	<a>QSxn	[x] Buffered
Units	x = function	[ ] Device specific
Range	x = A, B, C, D, E, F, G, H n = on/off	[ ] Saved independently
Default	None	[ ] Saved in sequences
Response	None	
See also	QR	

The QS commands allow the PC23 to send an interrupt to the host computer when attention is required. Once enabled, QS commands reside in the PC23's background during normal operation. When the condition that prompts the interrupt is met, the interrupt is generated. The interrupt is determined by the host computer via the QI or QIB command. The interrupt will be generated each time the condition occurs, until the function is disabled (Refer to the parameter n in the description box above. When n = ø (off). When n = 1 (on). If more than one interrupt is pending, the rest are ignored (if one is buffered, the rest are lost).

Refer to *Chapter 4, Application Design*, for information on setting hardware interrupts on the PC bus.

Bit	Function
A	Trigger 1 active
B	Interrupt on Move Complete
C	Not Used
D	Interrupt on Limit Encountered
E	Interrupt on Ready to Respond
F	Not Used
G	Interrupt on Command Buffer Full
H	Interrupt on Motor Stall

QSA Interrupt on Trigger #1 High

Version A

Type	Set-Up
Syntax	<a>QSAn
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	TR, TS, QR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

QSAø = Do not send interrupt on Trigger #1 being high
QSA1 = Interrupt signal on Trigger #1 being high

Entering QSA1 causes the indexer to transmit an interrupt signal to the host computer whenever trigger #1 goes high. How the interrupt is handled is dependent upon the host computer and your interrupt routine within the host.

QSB Interrupt on Move Complete

Version A

Type	Set-Up
Syntax	<a>QSBn
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	QR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

QSBø = Do not send interrupt on move complete
QSB1 = Interrupt signal on move complete

Entering QSB1 causes the indexer to transmit an interrupt signal to the host computer whenever a move has been completed. How the interrupt is handled is dependent upon the host computer and your interrupt routine within the host.

QSD Interrupt on Limit Encountered

Version A

Type	Set-Up
Syntax	<a>QSDn
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	QR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

QSDø = Do not interrupt signal upon reaching a limit
QSD1 = Interrupt signal upon reaching a limit

Entering QSD1 causes the indexer to transmit an interrupt signal to the host computer whenever a CW or CCW limit has been reached. How the interrupt is handled is dependent upon the host computer and your interrupt routine within the host.

QSE Interrupt on Ready to Respond

Version A

Type	Set-Up
Syntax	`<a>QSEn`
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	QR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`QSEØ` = Do not interrupt when indexer is ready to respond
`QSE1` = Interrupt when indexer is ready to respond

Entering `QSE1` causes the indexer to transmit an interrupt signal to the host computer whenever the indexer is ready to respond from a status command. How the interrupt is handled is dependent upon the host computer and your interrupt routine within the host.

QSG Interrupt on Command Buffer Full

Version A

Type	Set-Up
Syntax	`<a>QSGn`
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	QR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`QSGØ` = Do not interrupt on buffer full (less than 32 bytes free)
`QSG1` = Interrupt on buffer full

Entering `QSG1` causes the indexer to transmit an interrupt signal to the host computer whenever the command buffer is full. How the interrupt is handled is dependent upon the host computer and your interrupt routine within the host.

QSH Interrupt on Motor Stall

Version A

Type	Set-Up
Syntax	`<a>QSAn`
Units	n = function on/off
Range	0 or 1
Default	0
Response	None
See also	DQ, OSE1, OR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

`QSHØ` = Do not interrupt on stall
`QSH1` = Interrupt on stall detect

Entering `QSH1` causes the indexer to transmit an interrupt signal to the host computer whenever a stall is detected. How the interrupt is handled is dependent upon the host computer and your own program within the host.

This command is functional only if you enable stall detection with the `OES1` command. When enabled, a stall is detected if the indexer receives no encoder pulses after moving the motor 1/50th of a rev (corresponds to 1 mechanical pole of a 50-pole motor). Thus, when the indexer and motor/drive system resolution is 25,000 steps/rev, 500 steps (25,000 ÷ 50) are output before a stall is detected.

Entering `QSHØ` causes no interrupt to be transmitted by the indexer when a stall is detected.

Command	Description
QSH1	An interrupt will be transmitted to the host when a stall occurs.

R Report Indexer Status

Version A

Type	Status
Syntax	aR
Units	None
Range	None
Default	None
Response	*x (x = R, B, S, or C)
See also	RA, RB, RC

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

You can use the Request Indexer Status (R) command to review the status of the indexer. Possible responses are:

Response	Definition
*R	Ready
*S	Ready, Attention Needed
*B	Busy
*C	Busy, Attention Needed

When the indexer is prepared to execute an immediate command, the following conditions delay the execution:

- ❑ Performing a preset move
- ❑ Accelerating/decelerating during a continuous move
- ❑ A time delay is in progress (T command)
- ❑ Paused (PS)
- ❑ Waiting on a Trigger (TR)
- ❑ Going Home
- ❑ Executing a loop

The following conditions cause error responses.

- ❑ A feedback error condition exits.
- ❑ Go home failed
- ❑ Limit has been encountered

You can obtain further details on the error condition with the RA, RB, or RC commands. *Compumotor does not recommended that you use this command in tight polling loops. It may overload the microprocessor. Time delays can alleviate this problem.*

Do not use this command to determine if a move is complete. Use it after a move is complete to determine if other errors or faults exist. Use a buffered status request command such as a CR (Carriage Return) or a programmable output to verify move completion.

Command	Response
1R	1:*R Axis #1 is ready to accept a command—no error conditions require attention

RA Report Limit Switch Status

Version A

Type	Status
Syntax	aR
Units	None
Range	None
Default	None
Response	*x (x = @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, or O)
See also	R, RB, RC, LD

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Limit Switch Status Report (RA) command responds with the status of the end-of-travel limits during the last move as well as the present condition. This is done by responding with one of 16 characters representing the conditions listed below.

Response Character	Last Move Terminated By		Current Move Limited By	
	CW Limit	CCW Limit	CW Limit	CCW Limit
*@	NO	NO	NO	NO
*A	YES	NO	NO	NO
*B	NO	YES	NO	NO
*C	YES	YES	NO	NO
*D	NO	NO	YES	NO
*E	YES	NO	YES	NO
*F	NO	YES	YES	NO
*G	YES	YES	YES	NO
*H	NO	NO	NO	YES
*I	YES	NO	NO	YES
*J	NO	YES	NO	YES
*K	YES	YES	NO	YES
*L	NO	NO	YES	YES
*M	YES	NO	YES	YES
*N	NO	YES	YES	YES
*O	YES	YES	YES	YES

The `RA` command is useful when the motor will not move in either or both directions. The report back will indicate whether or not the last move was terminated by one or both end of travel limits.

If you wish to disable the limit inputs, you may issue the `LD3` (Disable both limits) command.

Command	Response
2RA	2:*@

This response indicates that the last move on axis 2 was not terminated by a limit and that no limits are currently active.

RB Report Loop, Pause, Shutdown, and Trigger Status

Version A

Type	Status
Syntax	aRB
Units	None
Range	None
Default	None
Response	*x (x = @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, or O)
See also	L, PS, R, RA, RC, ST, TR

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The `RB` command responds with the status of the command buffer if it is presently executing a Loop (`L`) or a Shutdown (`ST`) command. It will also indicate if a Pause (`PS`) command is being executed or if a Trigger (`TR`) condition is presently being waited on.

The controller responds with one of 16 different characters, each of which represents one of the conditions listed below.

Response Character	Loop Active	Pause Active	Shutdown Active	Trigger Active
*@	NO	NO	NO	NO
*A	YES	NO	NO	NO
*B	NO	YES	NO	NO
*C	YES	YES	NO	NO
*D	NO	NO	YES	NO
*E	YES	NO	YES	NO
*F	NO	YES	YES	NO
*G	YES	YES	YES	NO
*H	NO	NO	NO	YES
*I	YES	NO	NO	YES
*J	NO	YES	NO	YES
*K	YES	YES	NO	YES
*L	NO	NO	YES	YES
*M	YES	NO	YES	YES
*N	NO	YES	YES	YES
*O	YES	YES	YES	YES

This command is useful to determine the present status of the execution buffer, especially when execution is held up or the response is unclear.

When you send a buffered command and the indexer does not execute the command, you may execute this command to receive a status of the indexer.

Command	Response
1RB	1:*A Axis #1 is currently executing a loop

RC Report Closed Loop and Go Home Status

Version A

Type	Status
Syntax	aRC
Units	None
Range	None
Default	None
Response	*x (x = @, A, B, or C)
See also	R, RB, OSE, GH, L, FSC

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Report Closed Loop And Go Home Status (RC) command responds with a character that represents one of the conditions described below.

- ❑ ***Homing Function Failure*** — In this condition, the controller reaches both end-of-travel limits or one of several possible Stop commands or conditions. The Go Home motion was concluded, but not at home.
- ❑ ***Stall*** — In this condition, the controller detects either a deviation between motor and encoder position that is larger than one pole of the motor while running, or a deviation larger than one pole of the motor plus the backlash parameter following a direction change.

Response Character	Stall Detected	Go Home Unsuccessful	Static Loss Detected
*@	NO	NO	NO
*A	YES	NO	NO
*B	NO	YES	NO
*C	YES	YES	NO
*H	NO	NO	YES
*I	YES	NO	YES
*J	NO	YES	YES
*K	YES	YES	YES

Command	Response
1RC	1:*C

This means that while attempting its last move, axis 1 detected a stall.

RM Rate Multiplier in Immediate Velocity Streaming Mode

Version A

Type	Set-Up
Syntax	<a>RMxxxx
Units	xxxx = rate multiplier value (hexadecimal)
Range	0000 - FFFF
Default	0000
Response	None
See also	Q1, QØ

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The RM command, followed by 4 hexadecimal digits, represents an immediate velocity setting. The variable nnnn is a hex value (in ASCII format) ranging from 0000 - FFFF which corresponds to 15.259 Hz - 500 kHz in increments of 15.259 Hz. The velocity change is instantaneous. There is no acceleration/ deceleration ramp between velocities. A limit switch-closure stops movement while in the axis is in velocity profiling mode, but does not cause the PC23 to exit immediate velocity streaming mode. Hex values 0000 - 7FF3 result in CCW rotation. Hex values 8000 to FFF3 result in CW rotation. Hex values 7FF4 - 7FFF and FFF4 - FFFF are special values that you can use to set POBs (refer to Chapter 4, Application Design for detailed information on contouring).

Command	Description
Q1	Enter Velocity Streaming mode on axis #1
RMØ666	Jump to 1 rps in the CCW direction
RMØCCC	Jump to 2 rps
RM1332	Jump to 3 rps
RM1998	Jump to 4 rps
RM1332	Jump to 3 rps
RMØCCC	Jump to 2 rps
RMØ666	Jump to 1 rps
RMØØØØ	Jump to 0 rps
QØ	Exit Velocity Streaming mode

The motor jumps to 1 rps when you issue the first RM value. The velocity increases by 1 rps when you issue the next three RM commands. The subsequent RM commands decrease the motor velocityin 1 rps increments until the motor stops. The amount of time that elapses between the issuance of each RM command determines the exact motion profile.

RV Report Software Part Number

Version A

Type	Status
Syntax	`<a>RV`
Units	None
Range	None
Default	None
Response	`92-nnnnnn-nn(xn) (n = 0 - 9, x = A - Z)`
See also	None

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The `RV` command responds with the software part number and its revision level. The response is in the form shown below:

```
        92-nnnnnn-nn<xn>
        part number <revision level>
```

The part number identifies which product the software is written for, as well as any special features that the software may include. The revision level identifies when the software was written. You may want to record this information in your own records for future use. This information is useful when you consult Parker Compumotor's Applications Engineering Department.

Command	Response
`1RV`	`92-006887-01K`

The product is identified by 92-006887. The revision level is identified by 01K.

S Stop

Version A

Type	Motion
Syntax	`<a>S`
Units	None
Range	None
Default	None
Response	None
See also	`K, SSD, SSH, A`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command decelerates the motor to a stop using the last defined Acceleration (`A`) command. This command normally clears any remaining commands in the command buffer, unless prevented from doing so by the Save Command Buffer On Stop (`SSB1`) command. When the `SSB1` command is executed, the `S` command stops only the current move and goes on to the next command in the buffer.

Command	Description
`MC`	Set to Continuous mode
`A1`	Set acceleration to 1 rps^2
`V1Ø`	Set velocity to 10 rps
`G`	Execute the move (Go)
`S`	Stop motor—motor comes to 0 rps at a deceleration rate of 1 rps^2

The `S` command cannot be put into the buffer since it is an immediate command. As soon as the indexer receives the `S` command, it stops motion.

SA Stop All

Version A

Type	Motion
Syntax	`<a>SA`
Units	None
Range	None
Default	None
Response	None
See also	`S, K`

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

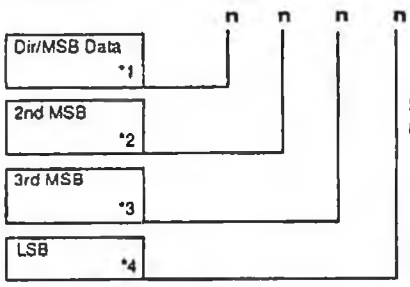
The Stop All command is equivalent to a 1S, 2S, 3S.

SD Define Timed Data Mode Streaming Data Version A

Type	Programming	**Attributes**
Syntax	`<a>SDn`	[x] Buffered
Units	n = hex digit up to 3 sets of 4	[ ] Device specific
Range	0 - F	[ ] Saved independently
Default	None	[ ] Saved in sequences
Response	None	
See also	Q2, Q3, TD, MSL, MSS	

This command allows you to define streaming data. In the time-distance (Q2) mode, you can specify the number of steps that the motor will move per interval with this command. In the time-velocity (Q3) mode, you can specify the velocity that the motor will move per interval. You can set the time interval with the TD command.

You may define streaming data with one or two or three four-digit hexadecimal numbers. If only one axis is in a timed data streaming mode, only one four-digit hex number is required (SDnnnn). If two axes are in a timed data streaming mode, two four-digit hexadecimal numbers are required (SDnnnnnnnn). If you have a 3 axis system, you need to send 3 four digit hexadecimal numbers (SDnnnnnnnnnnnn) You can define data for all three axes with a single SD command. Each SD command is only executed during a single TD interval.

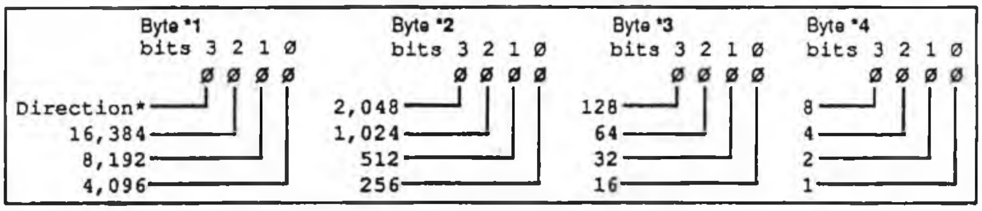


MSB = most significant bit
LSB = least significant bit

If you want to define data for one axis in a timed data streaming mode, use the configuration SDnnnn. If you want to define data for two axes in a timed data streaming mode, use the configuration SDnnnnnnnn. The first four hex numbers define data for the lower numbered axis. The next four hex numbers define data for the higher numbered axis.

The function of each byte (*1, *2, *3, and *4) is described below.

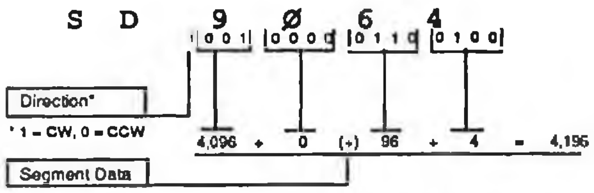
***1** sets the direction in which the axis will move. A Ø in the most significant bit (MSB) specifies CCW motion. A 1 in this position indicates CW motion. The remaining three bits of this byte are the 3 MSBs to define the magnitude of distance (Q2 mode) or velocity (Q3 mode). ***2** is the most significant *full* byte of segment data. ***3** is the next most significant *full* byte of segment data. ***4** is the least significant *full* byte of segment data. The weight of each bit in bytes ***1**, ***2**, ***3**, and ***4** is as follows:



In Q2 mode, this byte indicates the number of steps the motor will move per time interval. In Q3 mode, this byte indicates the velocity that the motor will move per time interval. You can set the time interval with the TD command. In the Q3 mode, the velocity desired per update interval is related to the four hex digits as follows (disregarding the most significant bit which is the direction bit):

```
Vel (in rps for a 25000 step/rev motor) = (nnnn * 15.259)/25000
```

Command	**Response**
`1SD9064`	Assuming that axis 1 is in a timed data streaming mode, this command specifies a distance of 4,196 CW steps for axis 1. Refer to the graphic depiction below.



Certain SD data points allow you to turn on outputs, wait on triggers, and loop. Refer to Chapter 4, *Application Design*, for more information.

SR Report Configuration Status

Version A

Type	`Status`
Syntax	`aSR`
Units	`None`
Range	`None`
Default	`None`
Response	`a:nnnnnnnn (a = axis number, n = 0 or 1)`
See also	`SS`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The SR command is a special status request command. This command provides information on the status of up to 5 software switches that you use to turn configuration options on and off. You can set these options with the SS commands.

The table below lists the response functions that you can receive status information on.

Code	Function	On/Off Status
A	*Not Used*	*N/A*
B	*Not Used*	*N/A*
C	Load and Go Mode	Ø = Disabled, 1 = Enabled
D	Alternate Mode Stop	Ø = Cycle End, 1 = Immediate
E	*Not Used*	*N/A*
F	Velocity Range	Ø = Normal, 1 = Low
G	Command Buffer on Limit	Ø = Purge, 1 = Save
H	Command Buffer on Stop	Ø = Purge, 1 = Save

Command	Response
`1SR`	`1:ØØØØØ1ØØ` This indicates that axis #1 is in the low-velocity range.

SSD Mode Alternate Stop Mode

Version A

Type	`Set-Up`
Syntax	`<a>SSDn`
Units	`n = function (on/off)`
Range	`0 or 1`
Default	`0`
Response	`None`
See also	`MA, SR`

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command determines the method of stopping when in the MA move mode.

SSDØ = Stop immediately
SSD1 = Stop at the end of the current loop

If you enable SSD1, upon receiving the Stop (S) command, the motor will move back to the starting position then stop motion. If you use the SSDØ command, the motor will decelerate to a stop immediately.

Command	Description
`SSD1`	Stop at end of loops in mode alternate

SSF Normal/Low Velocity Range

Version A

Type	Set-Up
Syntax	`<a>SSFn`
Units	n = function (on/off)
Range	0 or 1
Default	0
Response	None
See also	SR, V

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the velocity range for the Velocity (V) command. The normal range (SSFØ) for a 25,000-step/rev motor is .01 - 20 rps. The low range (SSF1) for a 25,000-step/rev motor is .001 - 2.00 rps.

Command	Description
SSF1	Set low velocity range for axis #1

SSG Clear/Save the Command Buffer on Limit

Version A

Type	Set-Up
Syntax	`<a>SSGn`
Units	n = function (on/off)
Range	0 or 1
Default	0
Response	None
See also	LD, SR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

In most cases, it is desirable that upon activating an end of travel limit input, all motion should cease until the problem causing the over-travel is rectified. This will be assured if all commands pending execution in the command buffer are cleared when hitting a limit. This is the case if SSGØ is specified. If SSG1 is specified and a limit is activated, the current move is aborted, but the remaining commands in the buffer continue to be executed.

Command	Description
SSG1	Save buffer on limit
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set distance to 25,000 steps
G	Execute the move (Go)
O11	Turn on outputs 1 and 2

If a limit switch is encountered while executing the move, outputs 1 and 2 will still go on.

SSH Clear/Save the Command Buffer on Stop

Version A

Type	Set-Up
Syntax	`<a>SSHn`
Units	n = function (on/off)
Range	0 or 1
Default	0
Response	None
See also	S, SR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

SSHØ = Clears command buffer on stop
SSH1 = Saves command buffer on stop

In Normal Operation (SSHØ) the Stop (S) command will cause any commands in the command buffer to be cleared. If you select the Save Command Buffer On Stop (SSH1) command,a Stop (S) command will only stop execution of a move in progress. It will not stop execution of any commands that remain in the buffer.

Command	Description
SSHØ	Clear command buffer on stop
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set distance to 25,000 steps
L5Ø	Loop 50 Times
G	Execute the move (Go)
T.5	Pause the motor 500 ms
N	End Loop
S	Stop motion

When you issue the S command, the indexer will clear the buffer and stop the move.

ST Shutdown

Version A

Type	Set-Up
Syntax	`<a>STn`
Units	n = function (on/off)
Range	0 or 1
Default	0
Response	None
See also	None

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Shutdown (`ST1`) command rapidly decreases the motor current to zero. The system ignores move commands that you issue after the `ST1` command. Torque on the motor is not maintained after you issue the `ST1` command.

The `STØ` command rapidly increases the motor current to normal level. Once you restore the current, you can execute moves.

This command is useful for reducing motor heating and allows you to manually position the load. The motor position counter is set to zero when you re-energize the motor using the `STØ` command.

Most Compumotor drives have a Shutdown Input along with Step and Direction inputs. The `ST1` command activates the shutdown input of the drive, disabling the current going through the motor.

Command	Description
`ST1`	Shuts off current to the motor

T Time Delay

Version A

Type	Programming
Syntax	`<a>Tn`
Units	n = seconds
Range	0.01 to 999.99
Default	None
Response	None
See also	None

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Time Delay (`T`) command causes the indexer to wait the number of seconds that you specify before it executes the next command in the buffer. This command is useful whenever you need to delay the motor's actions.

Command	Description
`MN`	Set to Normal mode
`A5`	Set acceleration to 5 rps^2
`V5`	Set velocity to 5 rps
`D25ØØØ`	Set distance to 25,000 steps
`T1Ø`	Pause motor movement 10 seconds
`G`	Execute the move (Go)
`T5`	Pause the motor for 5 seconds after the move ends
`G`	Execute the move (Go)

TD Set Time Interval for Timed Data Streaming Mode

Version A

Type	Motion
Syntax	`<a>TDnn`
Units	nn = ms
Range	02 - 50
Default	10
Response	None
See also	MSL, MSS, SD, QØ, Q2, Q3

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

This command sets the time interval for execution of segments defined with the `SD` command. Each `SD` command will be executed during one time interval only. In `Q2` mode, the segment profile will be derived from the time (in milliseconds—ms) specified by nn, where nn = 02 to 50 (2 to 50 ms in increments of 2 ms) , and the segment distance specified with an `SD` command (i.e., the motor will move the distance—in steps—specified with the `SD` command in the time set with the `TD` command). In `Q3` mode, the segment profile will be derived from the time specified by nn, and the velocity (in steps per second) specified by the `SD` command. i.e. The motor will achieve the velocity specified with the `SD` command in the time set with the `TD` command.

Command	Description
Q2	Enter Time Distance mode
TDØ4	Sets update time to 4 milliseconds
MSL11X	Axis #1 and axis #2 are in Time Distance mode
SDØØØØ8Ø28	Move axis #2 40 steps within the update time
SDØ1ØØØ28	Move axis #1 256 steps, axis #2 40 steps within 4 ms
MSS	Start master clock (Upon entering this command, moves defined by SD command will start)

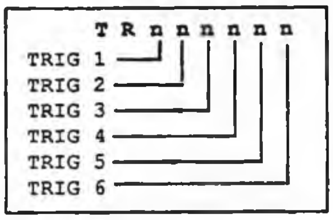
TR Wait for Trigger

Version A

Type	Programming
Syntax	<a>TRnnnnnn
Units	n = function
Range	(0 = open-on/off, 1 = closed, x = don't care)
Default	0
Response	None
See also	IS, TS

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

Triggers are used to synchronize indexer operations with external events. They can be used to implement a handshaking function with other devices. There are six triggers (see below).



When TR command is used in a buffer, the indexer will get to this command and wait until the input pattern is matched before going on to the next command.

Command	Description
TR1ØXXXX	Wait for input #1 to be grounded and input #2 to be opened before going on to the next command. Inputs #3 - #6 are ignored.
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25ØØØ	Set distance to 25,000 steps
G	Execute the move (Go)

TS Report Trigger Input Status

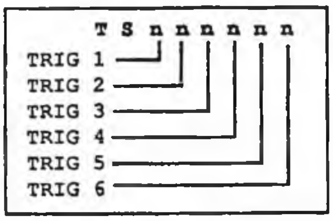
Version A

Type	Status
Syntax	aTS
Units	None
Range	None
Default	None
Response	a:nnnnnn (a = axis number, n = o, 1)
See also	IS, TR

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Trigger Status command retrieves the status of the trigger inputs.

- n= 1 (Input is ON)
- n = 0 (Input is OFF)



Since the TS command is immediate, the host controller can determine the status of the trigger inputs at any time, even during execution of other commands. You can use this command to make sure that your trigger pattern is met, when you have issued the Trigger (TR) command.

Command	Response
1TS	1:1Ø1Ø11

Trigger bits 1, 3, 5 & 6 are active. Trigger bits 2 and 4 are inactive.

U Pause and Wait for Continue

Version A

Type	Programming
Syntax	<a>U
Units	None
Range	None
Default	None
Response	None
See also	C, PS

Attributes
- [ ] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved in sequences

This command causes the indexer to complete the move in progress, then wait until it receives a Continue (c) to resume processing. Since the buffer is saved, the indexer continues to execute the program (at the point where it was interrupted). The indexer continues processing when it receives the c command. This command is typically used to stop a machine while it is unattended.

Command	Description
MN	Set to Normal mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
LØ	Loop indefinitely
D256ØØ	Set distance to 25,600 steps
G	Execute the move (Go)
T1Ø	Wait 10 seconds after the move
N	End loop
U	Halt execution until the indexer receives the Continue command.

This command string pauses at the point where the U command is entered. A Continue (c) command causes execution to resume at the point where it was paused. In this example, the loop stops at the end of a move, and resumes when the indexer receives the c command.

UR Report Scale Factor Status

Version A

Type	Status
Syntax	aUR
Units	None
Range	None
Default	None
Response	a:nnn (nnn = 001 to 255)
See also	US

Attributes
- [x] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved in sequences

Reports the scale factor set using US command. The actual number of steps sent to the motor drive during a Preset move will be the current distance parameter setting multiplied by this scale factor. The range for nnn is 001 to 255.

Command	Description
MN	Set to Continuous mode
A1Ø	Set acceleration to 10 rps^2
V5	Set velocity to 5 rps
US2Ø	Set scale factor to 20
1UR	Request scale factor. The response is 1:020 (verifies that scale factor is set to 20)
D2ØØØØ	Set distance to 20,000 steps
G	Execute the move (Go)

Axis #1 sends out 20,000 • 20 = 400,000 steps.

US Set Position Scale Factor

Version A

Type	Set-Up
Syntax	<a>USn
Units	n = distance multiplier
Range	1 - 255
Default	1
Response	None
See also	UR

Attributes
- [x] Buffered
- [ ] Device specific
- [ ] Saved independently
- [ ] Saved in sequences

This command sets the distance scale factor from 1 to 255. Any distance value set with the D command will be multiplied by the value set by US command. This value will remain valid until the indexer is reset or until a new US value is defined.

For the US command to be effective, you must issue a distance (D) command after the US command.

If 200 steps will yield 0.001 inches of rotary movement, you can issue a US2ØØ command and program distance in 0.001-inch increments.

Command	Description
MN	Set to Continuous mode
A1Ø	Set acceleration to 10 rps^2
V5	Set velocity to 5 rps
US2ØØ	Set scale factor to 200
D1ØØØØ	Set distance to 10,000 steps
G	Execute the move (Go)

V Set Velocity

Version A

Type	Motion
Syntax	<a>Vn
Units	n = rps
Range	0.01 to 160.000 (Motor dependant)
Default	1
Response	None
See also	A, D, G, MR

Attributes
[x] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Velocity (V) command defines the maximum speed at which the motor will run when given the Go (G) command. The actual speed of the motor or output frequency of the indexer will vary, depending on the motor drive resolution. The following formula is used to determine the output frequency of the indexer:

The motor resolution is a function of the motor/drive. However, you may match the PC23 to the motor/drive's resolution using the Motor Resolution (MR) command.

```
        Frequency = (n) • (Motor Resolution) in steps/rev.
```

The top speed of the motor drive is limited by the motor type. Entering a velocity higher than the top speed of a motor drive system will cause the motor to stall and may cause the drive to fault.

Command	Description
MC	Set to Continuous mode
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
G	Execute the move (Go)

In preset mode, Mode Normal (MN) the maximum velocity may also be limited when the resulting move profile is triangular. In Mode Continuous (MC), a Go (G) command is completed—the indexer moves on to the next command in the buffer once the specified velocity is reached.

W Report Immediate Position

Version A

Type	Status
Syntax	aWn
Units	n = form of response
Range	1 = binary, 3 = hex ASCII
Default	1
Response	1W1 = a:nnnnnn or xxxxxxxx (n = position in binary) 1W3 = a:xxxxxxxx (n = axis number, x = position in hex ASCII)
See also	FSB, P, PB, PR, PX, PXB

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Immediate Position Request (W) command provides a position report of a specified axis while the motor is moving. The report indicates position relative to the start of the current move or the completion of the last move. This command works in both encoder step mode (FSB1) and motor step mode (FSBØ).

If you specify the variable n as 1, the response format will be a 5-byte binary number (with no carriage return). The first byte is the axis number and the remaining bytes is the position in 2's compliment notation.

If you specify the variable n as 3, the response format will be an optional axis number, a colon, and eight hex ASCII characters in two's complement signed notation (with a carriage return).

Interpreting Hexadecimal Position Reports

This form of position report (a:xxxxxxxx) is generated by the W3 command. It consists of an an optional axis number and a colon, followed by eight hexadecimal characters: 0 - 9 and A - F. The position report is followed by a carriage return.

The decimal value of the hexadecimal expression can be determined using the technique demonstrated in Tables 5-1 and 5-2. The response is in 2's complement notation reflecting direction; negative numbers imply CCW motion. Both commands are designed for computer controlled situations where the computer can translate the hexadecimal.

Interpreting Binary Position Reports

This form of position report (`nnnnn`), consists of five bytes. The first is the axis number, followed by four bytes that must be linked together (concatenated) to form a 32-bit binary number. A typical PC23 communications algorithm expects to handle characters rather than binary numbers and may have problems with this kind of response. Assume that a response equivalent to the ASCII characters `^@`, `#`, `0`, and `/` (`^@` refers to the CTRL key and `@`, an unprintable character) is given. The binary code for this response should be:

```
    ^@         #          0          /

00000000  00100011  00110000  00101111
```

This code has to be interpreted by the computer. The four characters must be converted to their ASCII code numbers and multiplied by the appropriate power of 256. The first character received is the most significant byte. Refer to the following table for a conversion technique for `^@ # 0 /`. The formula used for the binary conversion is:

```
ASCII  Value  •  Character  Multiplier  =  Character  Value
```

Response	ASCII Hex Value	ASCII Decimal Value	Character Multiplier	Conversion (steps)
^@	Ø	0	16,777,216 (256^3)	0
#	23	35	65,536 (256^2)	2,293,760
0	3Ø	48	256 (256^1)	12,288
/	2F	47	1 (256^0)	47
			Position Total:	2,306,095

If the position total is ≥2,147,483,648, then the position total is showing a 2's compliment negative number. To establish the correct position total, subtract 4, 294, 967, 296 (= 2^{32} = 16^8) from your result to receive the correct negative number. For example, if the response is `1(smiley face)+Σa8`, the conversion would be as follows:

Response	ASCII Hex Value	ASCII Decimal Value	Character Multiplier	Conversion (steps)
+	F6	246	16,777,216 (256^3)	4,127,195,136
Σ	E4	228	65,536 (256^2)	14,942,208
a	61	97	256 (256^1)	24,832
8	38	56	I (256^0)	56
			Position Total:	4,142,162,232

To establish the correct position, subtract 4, 294, 967, 296 from 4,142,162,232. The result is -152,805,064. Therefore, the last position total indicated 152,805,064 steps were moved in the CCW direction.

Positive w3 Response Interpretation

The system provides responses in the following format:

```
 MSD       LSD
xxxxxxxx
```

The first digit is the most significant digit (MSD). The last digit is the least significant digit (LSD). Refer to the following table for the value of each digit.

Digit	Digit Multiplier	
x (MSD)	x $\cdot 16^7$ = h • 268,435,456 =	________
x	x $\cdot 16^6$ = h • 16,777,216 =	________
x	x $\cdot 16^5$ = h • 1,048,576 =	________
x	x $\cdot 16^4$ = h • 65,536 =	________
x	x $\cdot 16^3$ = h • 4,096 =	________
x	x $\cdot 16^2$ = h • 256 =	________
x	x $\cdot 16^1$ = h • 16 =	________
h (LSD)	x $\cdot 16^0$ = h • 1 =	________

The decimal (h) may have one of the values shown in Table 5-3.

Decimal Value	Hexadecimal Value	Decimal Value	Hexadecimal Value
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Using the previous tables, review the decimal value that would be calculated if the hexadecimal response was ØØØ433AE.

Hexadecimal	Character Multiplier	Conversion (steps)
Ø	0 • 268,435,456	0
Ø	0 • 16,777,216	0
Ø	0 • 1,048,576	0
4	4 • 65,536	262,288
3	3 • 4,096	12,288
3	3 • 256	768
A (= 10)	10 • 16	160
E (= 14)	14 • 1	14
	Total Steps:	275,374

Negative W3 Response Interpretation

If the first digit of the position portion of the response a:xxxxxxxx is 8, 9, A, B, C, D, E, or F, the response represents a *two's complement* negative number. Any other response should be interpreted per Table 5-1. There are several ways to convert an 8-digit two's complement hexadecimal number to decimal.

The Binary Approach

① Convert the hexadecimal response to binary form.

② Complement the binary number.

③ Add 1 to the binary result.

④ Convert the binary result to decimal value.

Response to 1W3 is 1:EFFE271Ø.

①	E	F	F	E	2	7	1	Ø	
	1110	1111	1111	1110	0010	0111	0001	0000	
②	0001	0000	0000	0001	1101	1000	1110	1111	
③	0001	0000	0000	0001	1101	1000	1111	0000	
④	1	Ø	Ø	1	D	8	F	Ø	1ØØ1D8FØ hex = 268,556,528 decimal.

Therefore, the result is 268,556,528 steps in the CCW direction.

The Computer Approach

① Convert the hexadecimal response to decimal.

② Subtract 4,294,967,296 (= 2^{32} = 16^{8}) from the decimal number.

Response to 1W3 is 1:EFFE271Ø.

① Convert hex to decimal as follows:

Hexadecimal	Character Multiplier	Conversion (steps)
E (= 14)	14 • 268,435,456	3,758,096,384
F (= 15)	15 • 16,777,216	251,658,240
F (= 15)	15 • 1,048,576	15,728,640
E (= 14)	14 • 65,536	917,504
2	2 • 4,096	8,192
7	7 • 256	1,792
1	1 • 16	16
Ø	Ø • 1	0
	Total Steps:	4,026,410,768

② 4,026,410,768 - 4,294,967,296 = ⁻268,556,528. Therefore, the result is 268,556,528 steps in the CCW direction.

Y Stop Loop

Version A

Type	Programming
Syntax	<a>Y
Units	None
Range	None
Default	None
Response	None
See also	L, N

Attributes
[ ] Buffered
[ ] Device specific
[ ] Saved independently
[ ] Saved in sequences

The Stop Loop (Y) command takes you out of inner loop when the loop completes its current pass. This command does not halt processing of the commands in the loop until the indexer processes reach the last command of the current loop. At that time, the indexer executes the command that follows the End Loop (N) command.

Command	Description
L	Loop indefinitely
A5	Set acceleration to 5 rps^2
V5	Set velocity to 5 rps
D25000	Set distance to 25,000 steps
T2	Wait 2 seconds
G	Execute the move (Go)
N	End loop
Y	Stop loop

The loop requires the motor to move 25,000 steps and then wait for two seconds. The loop terminates at the end of the loop cycle it is executing when it receives the Y command.

Appendix B KS-Drive Commands

All of the commands may be prefixed with a device address. Any command that will cause the drive to transmit information to the RS-232 port MUST be prefixed with a device address. This is to prevent several units from transmitting at the same time.

Responses and reports from the drive will have a * as a leading character to prevent the response from being interpreted as a command by other drives on the communication link.

Invalid commands will be ignored by the drive.

You may send either upper or lower case characters to the K drive, However the Echoed characters from the drive will always be Upper case. Thus in practice it is best to use Upper case to avoid confusion.

GENERAL COMMANDS
****

E Enable the RS-232.

NOTE: IF YOU DO NOT GET ANY RESPONSE ON THE RS-232 LINK YOU PROBABLY DID NOT ISSUE THE E COMMAND.

The E command may be preceded by a device address 1 to 15, for example 1E will enable only device number one. Sending an E without a device address will enable all of the RS-232 ports on all of the drives on the daisy chain at the same time.

Enabling the RS-232 tuning function will disable the pushbutton tuning.

SV Save new values.
SAVE

This is the same as pushing all three mode buttons down and releasing them. The "SV" command will cause the controller to save the gains as they are now adjusted and exit the tuning process, meaning the tuning mode will be exited.

The save command will save any new values you have given the drive. For those values that were not changed the last value to have been saved will be resaved.

F	Exit the RS-232 mode and return to front panel control. The F command will return control to the front panel pushbuttons. Any changes that have been made to the controller's gains are retained in volatile memory. However if a SAVE command is not issued the values will be lost on the next power loss or reset.
OFF ST0	Turns the power amplifier off. No current flows through the motor. AC power to the amplifier remains on.
ON ST1	Turns the power amplifier on. Allows current to flow through the motor.
RFS	Returns the drive to Factory Settings. All settings are as they were when the drive was shipped from the factory. This is the same as pushing the P and I buttons at the same time.
HELP	Returns help menu Provides a list of commands and a brief description of each command. Note that the Help command is the only command that is not device specific. It will respond without a device address. If you have several units on a daisy chain and you type HELP the result will be that all of the units on the chain will transmit at the same time. You will have a great deal of confusion. Don't do it!
ESCAPE KEY	Will take you to the out of the help menu.
DFS	Display all of the servo status flags Syntax: <a>DFSd Type: Status, Immediate, Device Specific Description: Returns all servo status flags as 32 bits where the response is *bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb*[cr] where the order of the bits is *31,30,29,28_ ....._3,2,1,0*

```
bit

31  27  23  19  15   these bits are all reserved for future use
30  26  22  18  14    they all return zero's
29  25  21  17  13
29  24  20  16  12

bit

11      enable circuit   0 = enabled  1 = disabled
10      high voltage problem  no = 0  yes = 1
9       indexer sending pulse at power up no = 0  yes = 1
8       failed crc check no = 0  yes = 1

7       power dump overtemp no = 0  yes = 1
6       average current exceeded  no = 0  yes = 1
5       max position error exceeded no = 0  yes =1
4       remote shutdown from indexer (non X version)  no = 0  1 = yes

3       driver error undefined 0 = no error 1 = pwm hardware shutdown
2       drive over temp   0 = no   1 = yes/shutdown
1       overcurrent  0 = no  1 = yes/shutdown error..
0       RS 232 CMD    0 = on/st1    1 = off/st0
```

RSE Reports Servo Errors.

If an error condition in the servo drive exists, such as excessive following error or an EEPROM failure, it will be reported. Errors are "soft errors" that are indicated with the ERROR LED and display codes. To clear an error one must reset the drive. The possible error messages are listed below

```
#11_amplifier_overheating___
#16_amplifer_off___
#17_indexer_shutdown___
#18
#19_amplifier_overcurrent___
#20_exessive_position_error___
#22_excessive_average_current___
#23_drive_enable_plug_not_inserted___
#24_regen_overheating___
#30_failed_CRC_in_EEPROM___
#60_rs232_command___
#61_indexer_incoming_pulses___
```

Z	Resets the drive. ("1E" must be entered after a "Z") The drive will act as if power was cycled. This command implements a software reset of the system. Any changes that have not been saved before issuing this command will be lost.
RV	Software Revision level reported. This command is for determining the software revision level of the controller software. It will report the part number that is written on the label of the controller's EPROMs. Using this command means it is not necessary to open the Drive amplifier's box in order to determine the revision level of the software.
KILL K STOP S	STOP POWERING THE MOTOR Issuing this command will cause the microprocessor to stop commanding power to the motor until the ON, ST1, or Z command is received. All pulses/position will be lost. These commands remove power from the amplifier and allow the motor to freewheel. These commands were added so that if during the tuning procedure over the RS-232 link the user makes the servo go unstable he has some means of stopping the system. This will normally be a panic situation so the commands were selected to be those most likely to be selected in this situation. The commands function exactly the same as the OFF or ST0 command. It should be noted that in the KX versions the S and Stop commands are controlled by the acceleration settings. So the function of these commands change between the indexer and non indexer versions.

CONFIGURATION COMMANDS
****

The following configuration commands are designed to let you the user set up the system to meet your requirements. Normally the factory settings for the motor driver combination are all that you would need. However, to allow for your special situation we have added the following commands. In the event that you have to replace a motor the first command that you would execute is the CMTR if and only if you have changed the type of motor that the drive was origionally set up for. After the drive is properly set up for the motor size, The FMCA command will Find the Motor Commutation Angle of the motor/resolver combination for the particular motor you have. This is seldom a requirement but it is necessary before any of the other commands can be expected to execute in a normal fashion. CAUTION THE FMCA COMMAND CAUSES MOTION, SEE COMMAND LIST.

The next command needed will be the CMR command. This selects the motor resolution that you desire to work with. (The drive must be in shutdown mode before the CMR can be executed). The motor resolution will affect any command or report that is in motor steps. Therefore you should choose the motor resolution before you do much else.

CMTR Configure/report Motor Type

Use this command to configure the drive to the motor size that is being used with the drive. Normally this is done for you at Compumotor's factory. This command selects the proper current values and factory defaults for the various motor sizes used with the drive. The possible configurations are:

Example: 1CMTR This will report the present set up as *CMTR

1CMTR210	Sets the drive up for the KS210 motor
1CMTR220	Sets the drive up for the KS220 motor
1CMTR230	Sets the drive up for the KS230 motor
1CMTR240	Sets the drive up for the KS240 motor
1CMTR250	Sets the drive up for the KS250 motor
1CMTR260	Sets the drive up for the KS260 motor

Like the FMCA (see below) command it is not necessary to do a save after issuing a CMTR command with a parameter as it will automatically save the new motor type when the command is issued.

FMCA Find Motor Commutation Angle

Use this command if a new motor is being used or if a new resolver has been mounted on the motor, or if a resolver has been re-mounted on the motor. The use of this command is highly unusual, it will cause the motor to rotate under program control in the drive. WHILE IT IS ROTATING THE USER HAS NO CONTROL OVER THE MOTOR. ONCE YOU START THIS SEQUENCE OF EVENTS YOU WILL HAVE TO CUT OFF THE AC POWER IF YOU WISH TO STOP THE MOTOR BEFORE THE COMMUTATION PROCEDURE IS COMPLETED. (For safety reasons you would normally find the commutation angle with the motor disconnected from the load). It is not necessary to save after a this command as the offsets are automatically saved.

CMR Configure Motor Resolution definition/report numeric parameter expected (1000 to 32,768 for KS250 and 260 motors, 1000 to 16,384 for all other KS-Series motors). The drive must be shut down bafore the CMR cmd can be executed.

Define/report motor resolution. Enter a number between 1000 and the maximum allowable resolution for the particular motor size (32,768, 16,384, or 8,192 depending on resolver configuration) the value entered represents the number of steps of resolution you want the resolver reading to have per revolution of the motor. If you choose a binary value the positioning will be slightly more accurate than a non binary value will provide. The math is done as an interger value so truncation error within a single revolution can occur. This error is not cumulative.

If a valid integer number is sent then the new resolution will be that number of steps per motor revolution. If no value is sent then the current resolution is reported. Factory Default is 16384 (KS210 - 240) or, 25000 (KS250,260) steps per revolution.

The 250 and 260 motors actually function at 32,768 steps per rev, and the 210 through 240 at 16,384 steps per rev, in all cases. So changing the motor resolution does not affect dynamic performance. The microprocessor simply converts the number of incoming indexer pulses to the appropriate absolute resolver position mathematically. Thus a binary number converts to an exact resolver position, whereas a non binary number may be rounded off for a given position. However since the KS converts the absolute indexer count into the position there is no accumulation of error.

FOR EXAMPLE[1]: If the resolution is set to 5000 steps per rev we compute the scale factor as (32,768*65536)/5000 = 429496.7296 but because we have 16 bit precision the .7296 is truncated.

If you command a move with D500000 Then the conversion from user friendly units back to resolver units is done as follows:

(500,000 *429496)/65536 = 3276794.434 but because we can't move to a fractional position the motor actually moves to 3276794 counts of the resolver. So where you would expect the motor to go exactly 100 revolutions (500,000 / 5000) for this move the motor will actually go 3276794/32768 = 99.99981689 revolutions. This error does not accumulate because if you gave a second move of the same distance the calculation will use the absolute distance to calculate the next move.

If all of this turns out to be a bother and the truncation error is a problem to you. You can simply choose a resolution that does Divide evenly into (32,768 * 65536) or 2147483648. e.g. 4096, 8192, 16384 etc.

---

[1] This example assumes the use of a KS260

Since a change in resolution could cause major dynamic discontinuities the motor resolution can not be changed while the system is active. You must issue the STO or OFF command to disable the drive before you can change the motor resolution.

Be certain to save any changes you wish to retain before cycling power to the drive. **NOTE:THIS COMMAND CHANGES THE VALUES AND RESPONSES OF ANY OTHER COMMAND THAT USES MOTOR STEPS. YOU SHOULD CONFIGURE MOTOR RESOLUTION BEFORE USING ANY OF THE OTHER CONFIGURE COMMANDS.**

Before the CMR command is active you must issue either the OFF or STO command to shut down the drive. This is to prevent the motor from making large unexpected moves when the Resolution is changed. After the CMR is implimented you will need to issue the ON or ST1 to reenable the drive. The new Resolution you have just issued will not take effect until you issue a Distance command after the CMR command.

CCA   Configure Current, Average, limit- Definition/report.
       Numeric parameter expected.(0 to 10.000 Amps)

If no parameter is supplied this command will report the currently defined maximum allowed average current, in amps. If a parameter is supplied that approximate number will be used as the new maximum average current. The acutal resolution of the control is .00122 amps so integer values are rounded to the nearest approximation. The controller continuously computes an average current to the amplifiers over a 2.56 second time span. If the average current command exceeds the value defined by the this command the controller will disable the amplifier and indicate an error. The factory default setting is motor dependant.

CCP   Configure Current Peak definition/report.
       Numeric parameter expected.(0 to 20.000 Amps)

Define/report the peak current limit. This number defines the maximum current command that will be sent to the motor. This command is included for diagnostic and other special purposes. If a valid number in amperes is entered, the approximation will become the new peak current limit.The actual resolution of the value is .1568 amps so integers are rounded to the nearest value. If no value is sent this command will report the present value defined for the maximum peak current, in amps. The maximum current is the absolute maximum current that will be sent to the motor, it is not the maximum average current command but the maximum transient current. It is the upper limit of how large the current can ever be. The Factory setting will depend upon the motor size. This is not an error or shutdown limit. This command sets the maximum current that the drive will put out. In effect it is a torque limit.

CDB   Configure DeadBand definition/report.
       Numeric parameter expected.(0 to 32,767)

If no parameter is supplied this command will report the current value of the deadband, in motor steps. If a number is supplied that number becomes the new deadband. The slip fault line to the indexer is used to indicate when the absolute value of the following error is within the deadband region. The slip fault line to the indexer connector will be on to indicate that the following error exceeds the deadband and off to indicate that the absolute value of the following error is within the deadband. (The slip fault line is active high). The default factory setting is zero. This is useful in situations when you need to know if the motor rotor is within a certain tolerance range with respect to the indexer command. If you wish to use this output to an external computer, you will need to connect wires from pin 10 and pin 22 of the indexer connector on the KS to your computer.

CPE   Configure Position Error definition/report
       Numeric parameter expected (0 to 65,535 Steps)

Define/report maximum following error. If the absolute position error is greater than this number, the amplifier will shut itself off. If a valid number in steps is entered, it will become the new maximum following error. Otherwise, the current setting is reported. Exceeding the maximum following error is an error condition that will cause the amplifier to be shutdown. If the maximum following error is defined as zero the "shutdown motor on following error exceeded" function is disabled and no amount of following error will generate an error condition or shutdown the motor. The factory default setting is one revolution of the motor.

The value of the following error is only calulated when the CPE command is given. The stored number is in terms of motor revolutions. So that Changes of the CMR resolution will leave the following error actual distance unchanged unless a new CPE command is issued.

This command differs from the CDB command in that being outside deadband only affects the slip fault output. The CPE settings will shut off the drive.

You should be certain to SAVE your settings to the EEPROM if you wish it to be permanent.

Example: 1CMR5000 1CPE1000 will set the following error to one fifth of a revolution of the motor. issuing 1CMR25000 1CPE will get a response of *position error 5000 since it is still one fifth of a rev. 1CPE1000 issued after the 1CMR25000 would set the position error to 1/25th of a rev.

TUNING COMMANDS
****

The following tuning commands are accessed via pushbutton tuning or the RS-232 communication link. The values represent a percentage of the maximum value that the term is allowed to achieve. The following commands change the percentage only and the range is limited from 1 to 99.To change the maximum value of the term it is necessary to use the configuration maximum commands. push buttons allow only integer values of the percentage. The display will show only the integer.

the form of the equation for the tuning set up is:
Gain value = term maximum x term percentage

CVG Configure Velocity Gain definition/report.
numerical parameter expected (0 to 99)

The velocity gain is related to the error in the motor speed with respect to the commanded velocity from the indexer. If a valid numerical parameter is enter then the Velocity gain will be recalculated using the new percentage of the maximum. Otherwise, the current setting will be reported.

CDG Configure Differential Gain definition/report.
numerical parameter expected (0 to 99)

The differential gain is related to position error changes with respect to time. If a valid number is entered, a new differential gain will be calculated using the percentage of the maximum. Otherwise, the current setting is reported.

CIG Configure Integral Gain definition/ report.
numerical parameter expected (0 to 99)

The integral gain is related to position error with respect to time. If a valid number is entered, a new integral gain will be calculated using the percentage. Otherwise, the current setting is reported.

CPG Configure Proportional Gain Definition/report.
numerical parameter expected (0 to 99)

The proportional gain is related to the position error. If a valid number is entered, a new proportional gain will be calculated based on the percentage. Otherwise, the current setting is reported.

CONFIGURE TERM MAXIMUMS
****

The following tuning commands are accessed only via the RS-232 communication link. The values:

CVM   Configure Velocity Maximum.   Definition/report.
         numerical parameter expected (1 to 32767)

This command allow you to change the factory value for the Maximum gain that the Velocity term can achieve. This would normally be changed only if the values provided for your motor were not satisfactory for your application.The default values are motor dependant. If a valid parameter is sent that value will become the new maximum, otherwise the current setting is reported.

CDM   Configure Differential Maximum Definition/report.
         numerical parameter expected (1 to 32767)

This is the gain of the portion of the controller which amplifies the derivative of the position error with respect to time. If a valid number is entered, it will become the new differential gain maximum. Otherwise, the current setting is reported.

CIM   Configure Integral Maximum Definition/report.
         numerical parameter expected (1 to 32767)

Defines the maximum of the integral of the position error with respect to time. If a valid number is entered, it will become the new integral gain maximum. Otherwise, the current setting is reported.

CPM   Configure Proportional Maximum Definition/report.
         numerical parameter expected (1 to 32767)

Defines the maximum of the term which amplifies the position error. If a valid number is entered, it will become the new proportional gain maximum. Otherwise, the current setting is reported.

NOTES:
If you supply an invalid parameter to a command the command will not be performed. Invalid commands are simply ignored.

Any changes made to parameters using these commands are NOT permanent UNTIL THEY ARE SAVED. To make a change permanent the SAVE command (SV) must be issued. The SAVE command will save all changes that have been made into the EEPROM. Thus, if changes are made with these commands and then the machine is reset the changes are lost unless the SAVE command was issued before resetting the machine.

DISPLAY/REPORT COMMANDS

****

Each of the display/report commands results in the drive periodically reporting the appropriate parameter to the terminal where it is displayed Approximately every 130 milliseconds as the value is reported to the terminal. When any of these commands is issued no other command may be issued until the reporting is terminated. To terminate a report hit any key to send a character to the drive. All display report commands must be prefixed with a device address. This is to prevent several units from trying to report at once. These commands do NOT display values to the two number display on the pushbutton tuning panel.

DCA periodically Displays/reports Current (average) in Amperes.

The average current flowing through the motor is reported and repeatedly updated until a key is pressed. To get this number 128 data points of the instantaneous current are read at 512 microsecond intervals. These 128 points are averaged to provide a value computed at 65 millisecond intervals. This value is averaged with the last 256 similar values to provide the average current over the last 16.67 seconds. The new average current is reported on every other calculation to be displayed at 130 millisecond intervals.

DCI periodically Displays/reports Current on an Instantaneous basis

This number is reported in Amperes and is repeatedly updated until a key is pressed. This number is a single sampling of the current at read and reported at a 128 millisecond rate.

DCP periodically Displays/reports the Peak Current

This number is reported in Amperes and is repeatedly updated until a key is pressed. This command samples the instantaneous current at 500 microsecond intervals. Each reading is compared to the largest previous reading, if the new value is larger then it will become the new value. This value is reported at 128 millisecond intervals. This reading accumulates from the time the command is sent. so that the highest instantaneous current ever seen by the motor over long periods of time may be captured.

DPA periodically Displays/reports the Position Actual

This number is reported in steps and is repeatedly updated until a key is pressed. This number is the absolute resolver position scaled in motor steps since the drive was enabled. The resolver of the KS actually counts at 32,768 counts per revolution of the motor shaft. The value that is displayed is calculated by creating a scale factor when the CMR command is issued. (32,768*65536)/CMR - scale factor. The scale factor is carried out to 16 bits. The rest is truncated. this can cause a small non cumualative error in position if the CMR resolution does not evenly divide into 2147483648.

The value is read and reported at about 150 millisecond intervals

DPE   periodically Displays/reports Position Error.

This command reports the difference between setpoint and actual position in steps. This number is used by the position control algorithm to determine what sort of current should be sent to the motor.The difference between the command setpoint and the actual position is also used to determine if the motor is within the Deadband specified in the CDB command. This number is reported in motor steps and is repeatedly updated until a key is pressed. This number is a single instantaneous value read and reported at about 150 millisecond intervals.

DPR   periodically Displays/reports the Position Resolver.

This number is reported in steps and is repeatedly updated until a key is pressed.This value is the resolver position in motor steps within a single revolution of the motor shaft. This is an absolute value with the zero referenced to how the resolver is mounted on the shaft. This data would be useful to diagnose a resolver problem. This number is a single data point read and displayed at about 150 millisecond intervals

DPS   periodically Displays/reports the Position Setpoint

This number is reported in steps and is repeatedly updated until a key is pressed. This number is the absolute number of pulses sent to the drive from the indexer since the drive was enabled (or reset). This counter is read and reported at about a 150 millisecond rate.

DIC   periodically Displays/reports Indexer Counter

Periodically reports the contents of the indexer counter in steps. This number is reported and repeatedly updated until a key is pressed. This is the raw number of pulses sent to the drive from the indexer. It differs from the setpoint in that the Indexer counter only displays the last 4096 counts from the indexer. This command is used to detect problems in the indexer to drive interface. The counter is read and reported at about a 150 millisecond rate.

DVA   periodically Displays/reports Velocity Actual

This number is reported in steps per second and is repestedly updated until a key is pressed. This value is the actual velocity being read from the resolver, it is the actual shaft velocity over a 500 microsecond period. It is displayed every 28 miliseconds.

DVS periodically Displays/reports Velocity Setpoint

This number is reported in steps per second and is repeatedly updated until a key is pressed.This number is calculated at 500 microsecond intervals. and reported at 28 millisecond intervals. This value is the velocity being sent to the velocity part of the servo loop by the PID loop.

Appendix C AutoLISP Simulation Program

```
; This AutoLISP program is used to simulate the additive protoyping process in
; AutoCAD/AutoSurf environment.
; The radius of the initial cylinder core is 5 mm.
; The solid model is a cone with four internal cylindrical cavities.

; Creating the 10 mm diameter cylindrical core.
(entmake
'((0 . "3DSOLID") (5 . "9A") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi lo n o         ") (1 . "=0;& {n {m {rn {rn l") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k l") (1 . "3*2/ {rn {rn {j {o l") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m l") (1 . "):-+:'@+:2/3>+: {rn l o n g l") (1 . ",7:33 {rn {rn {rn {i {m l") (1 .
"9><: {rn {h {g {j {rn {f 90-(>-; ,6183: l") (1 . "9><: {no {nn {nm {j {rn {nl 90-(>-; ,6183: l") (1 . "300/
{rn {nk {nj {i l") (1 . "<01:r,*-9><: {rn noo njo o o n o j o o n V V o n o V V V V l") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {h {l {rn l") (1 . "9><: {ni {rn {nh {j {rn {ng 90-(>-; ,6183: l") (1 . "300/
{rn {rn {nf {h l") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V l") (1 . "300/ {rn {rn {mo {i
l") (1 . "<0:;8: {rn {nj {nj {nf {mn n {g {rn l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {nn {l {rn l") (1 .
"300/ {rn {rn {mm {nn l") (1 . "/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V l") (1 . "<0:;8: {rn {nf
{nf {nj {mn o {nm {rn l") (1 . "<0:;8: {rn {mo {mo {mm {ml n {nk {rn l") (1 . ":;8: {rn {mk {mk {nf {mj
o l") (1 . "<0:;8: {rn {mm {mm {mo {ml o {nh {rn l") (1 . ":;8: {rn {mi {mi {mm {mh o l") (1 . "):-+:' {rn
{mn {mg l") (1 . ":336/,:r<*-): {rn noo noo o o rn o j o o n V V l") (1 . "):-+:' {rn {ml {mf l") (1 .
":336/,:r<*-): {rn noo moo o o n o j o o n V V l") (1 . "/061+ {rn noj noo o l") (1 . "/061+ {rn noj moo o
l"))
)

; Creating additive prototyping tool head.
(entmake
'((0 . "3DSOLID") (5 . "A2") (100 . "AcDbEntity") (67 . 0) (8 . "TEXT") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi jo n o         ") (1 . "=0;& {n {m {rn {rn l") (1 .
"9@=0;&r3(;r>++-6= {rn {l {rn {o l") (1 . "3*2/ {k {rn {j {o l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {n {o {i
{h l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {m {i {h l") (1 . ",7:33 {g {rn {rn {f {m l") (1 . "-:961:2:1+ {rn
o o o o oqhmjgigfkoljllfljj o o o m l") (1 . "):-+:'@+:2/3>+: {rn l o n g l") (1 . "-:9@)+r3(;r>++-6= {rn {rn
{rn {j {i {h l") (1 . "9><: {no {nn {nm {j {rn {nl 90-(>-; ,6183: l") (1 . "99><:r3(;r>++-6= {rn {nk {rn {f
l") (1 . "9><: {nj {ni {nh {j {rn {ng 90-(>-; ,6183: l") (1 . "300/ {rn {nf {mo {f l") (1 . "<01:r,*-9><: {rn
noo moo imqj o o rn mqj o o n V V roqlnimmhhiioniglhfk oqfkgiglmfgojojnlhh o V V V V l") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {no {f {i {rn l") (1 . "99><:r3(;r>++-6= {rn {mn {rn {nn l") (1 . "9><: {mm
{rn {ml {j {rn {mk 90-(>-; ,6183: l") (1 . "300/ {rn {mj {mi {nn l") (1 . "<01:r,*-9><: {rn noo moo fj o o
rn j o o n V V o n o V V V V l") (1 . "300/ {rn {rn {mh {f l") (1 . "<0:;8: {mg {mo {mo {mf {lo o {nm
{rn l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {nj {nn {i {rn l") (1 . "99><:r3(;r>++-6= {rn {ln {rn {ni l") (1 .
"300/ {rn {rn {lm {ni l") (1 . "/3>1:r,*-9><: {rn noo moo nmo o o n rn o o o V V V V l") (1 . "300/ {rn
{rn {mf {nn l") (1 . "<0:;8: {ll {mi {mi {lm {lk n {nh {rn l") (1 . "<0:;8: {lj {mh {mh {rn {li n {nf {rn l")
(1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mo l") (1 . "<0:;8: {lh {mf {mf {mo {lo n {mj {rn l") (1 . ":;8: {lg {lf
{lf {mf {ko o l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {mm {ni {i {rn l") (1 . "<0:;8: {kn {lm {lm {mi {lk o
{ml {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mi l") (1 . ":;8: {km {kl {kl {lm {kk o l") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {mh l") (1 . ":;8: {kj {ki {ki {mh {rn o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mf
l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {lo l") (1 . "):-+:' {rn {lo {kh l") (1 . ":336/,:r<*-): {rn noo moo ho o o
rn j o o n V V l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {lm l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {lk l") (1 .
"):-+:' {rn {lk {kg l") (1 . ":336/,:r<*-): {rn noo moo nmo o o n j o o n V V l") (1 . ":/>-r3(;r>++-6= {rn
{rn {rn {li l") (1 . "):-+:' {rn {li {kf l") (1 . "/061+ {rn noj moo ho l") (1 . "/061+ {rn noj moo nmo l") (1 .
"/061+ {rn noo moo jj l"))
)

; Pause for about 3.5 seconds.
```

```
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ) ;| The function will not return the 500001 at the command line upon completion. |;

; Render the objects.
(command "_avrender")

; Pause for about 2 seconds.
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ) ;| The function will not return the 300001 at the command line upon completion. |;

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")
```

```
; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 20 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "99") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi fk n o        ") (1 . "=0;& {n {m {rn {rn |") (1 .
"9@=0;&r3(;r>++-6= {rn {1 {rn {o |") (1 . "3*2/ {k {rn {j {o |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {n {o {i
{h |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {m {i {h |") (1 . ",7:33 {g {rn {rn {f {m |") (1 . "-:961:2:1+ {rn
o o o o oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn
{rn {rn {j {i {h |") (1 . "9><: {no {nn {nm {j {rn {nl -:):-,:; ,6183: |") (1 . "99><:r3(;r>++-6= {rn {nk {rn
{f |") (1 . "9><: {nj {ni {nh {j {rn {ng -:):-,:; ,6183: |") (1 . "300/ {rn {rn {nf {f |") (1 . "/3>1:r,*-9><: {rn
nniqollmfilhglhmfn njo mk o n o n o o o V V V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {no {f {i {rn |") (1
. "99><:r3(;r>++-6= {rn {mo {rn {nn |") (1 . "9><: {mn {mm {ml {j {rn {mk 90-(>-; ,6183: |") (1 . "300/
{rn {rn {mj {nn |") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n V V o n o V V V V
|") (1 . "<0:;8: {mi {mh {mh {mg {mf o {nm {rn |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {nj {nn {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {lo {rn {ni |") (1 . "9><: {ln {lm {ll {j {rn {lk 90-(>-; ,6183: |") (1 . "300/ {rn {rn
{lj {ni |") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V |") (1 . "<0:;8: {li {lh {lg {lf {ko o
{nh {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nf |") (1 . "<0:;8: {kn {nf {nf {lh {km o {nm {rn |") (1 .
"<0:;8: {kl {lf {kk {nf {mf n {ll {rn |") (1 . ":;8: {kj {ki {kh {mg {kg o |") (1 . "-:9@)+r3(;r>++-6= {rn
{rn {mn {ni {i {rn |") (1 . "99><:r3(;r>++-6= {rn {kf {rn {mm |") (1 . "9><: {jo {rn {jn {j {rn {jm 90-(>-;
,6183: |") (1 . "300/ {rn {jl {jk {mm |") (1 . "<01:r,*-9><: {rn noo njo o o n o no o o n V V o n o V V V V
|") (1 . "<0:;8: {jj {ji {ji {lg {jh n {ml {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mj |") (1 . "<0:;8: {jg {jf
{mj {mh {km n {nh {rn |") (1 . "<0:;8: {io {mj {jf {lj {jh o {nh {rn |") (1 . "<0:;8: {in {jk {mg {mj {ko n
{ll {rn |") (1 . ":;8: {im {il {ki {lf {ik o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mh |") (1 . ":;8: {ij {kh {ki
{lh {ii o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mg |") (1 . "<0:;8: {ih {mg {jk {jf {ig n {ll {rn |") (1 .
":/>-r3(;r>++-6= {rn {rn {rn {mf |") (1 . "):-+:' {rn {ko {if |") (1 . "):-+:' {rn {ig {ho |") (1 . ":336/,:r<*-):
{rn noo njo o o rn o no o o n V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {ln {mm {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {hn {rn {lm |") (1 . "300/ {rn {rn {hm {lm |") (1 . "/3>1:r,*-9><: {rn noo moo o o
n o n o o o V V V V |") (1 . "300/ {rn {rn {hl {mm |") (1 . "<0:;8: {hk {kk {lf {ji {hj n {ll {rn |") (1 .
"<0/>-r3(;r>++-6= {rn {rn {rn {lj |") (1 . "<0:;8: {hi {lj {lj {jk {hj o {ml {rn |") (1 . ":;8: {hh {hg {il {lj
{hf o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {lh |") (1 . "<0:;8: {go {lg {lh {kk {ig o {nh {rn |") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {lg |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {lf |") (1 . ":/>-r3(;r>++-6= {rn {rn {rn
{ko |") (1 . "):-+:' {rn {hj {gn |") (1 . ",+->687+r<*-): {rn nolqglllojliklmnnl noo fqmlinnmmhikgiimmk o
n o V V |") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {km |") (1 . ":336/,:r<*-): {rn non njo mk o rn o
njqollmfilhglhmfog o o n V V |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {kk |") (1 . ":;8: {gm {kh {hg {kk {gl
o |") (1 . "/061+ {rn nolqglllojliklmnnl njo fqmlinnmmhikgiimmk |") (1 . "/061+ {rn fiqfkglmlhjnhfihlj
njo fqjmmfgijnolknhfj |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {jo {lm {i {rn |") (1 . "<0:;8: {gk {hm {hm {hl
{gj o {jn {rn |") (1 . "<0:;8: {gi {hl {hl {hm {gj n {jl {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {jk |") (1 .
":;8: {gh {hg {il {ji {gg o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {ji |") (1 . ":/>-r3(;r>++-6= {rn {rn {rn
{jh |") (1 . "):-+:' {rn {ig {gf |") (1 . ":336/,:r<*-): {rn non noo mk o rn o njqollmfilhglhmfog o o n V V |")
(1 . "<0/>-r3(;r>++-6= {rn {rn {rn {jf |") (1 . "/061+ {rn nolqglllojliklmnnl noo fqmlinnmmhikgiimmk |")
(1 . ":/>-r3(;r>++-6= {rn {rn {rn {ig |") (1 . ",+->687+r<*-): {rn fiqfkglmlhjnhfihlj noo fqjmmfgijnolknhfj
o rn o V V |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {hm |") (1 . ":;8: {fo {fn {fn {hm {fm o |") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {hl |") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {hj |") (1 . ":336/,:r<*-): {rn noo noo o o
rn o no o o n V V |") (1 . "/061+ {rn fiqfkglmlhjnhfihlj noo fqjmmfgijnolknhfj |") (1 . ":/>-r3(;r>++-6=
{rn {rn {rn {gj |") (1 . "):-+:' {rn {gj {fl |") (1 . ":336/,:r<*-): {rn noo moo o o n o no o o n V V |") (1 .
"/061+ {rn nno moo o |"))
)

; Pause for about 3.5 seconds
```

```
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")
```

```
; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 30 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "9A") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi fk n o         ") (1 . "=0;& {n {m {rn {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k |") (1 . "3*2/ {j {rn {i {o |") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k |") (1 . ",7:33 {h {rn {rn {g {m |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k |") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: |") (1 . "9><: {nm {nl {nk {i {rn {nj -:):-,:; ,6183: |") (1 . "300/ {rn {rn {ni
{g |") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n V V o n o V V V V |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {f {l {rn |") (1 . "9><: {rn {nh {ng {i {rn {nf -:):-,:; ,6183: |") (1 . "300/ {rn
{rn {mo {f |") (1 . "<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mn
{mm {ml {mk o {no {rn |") (1 . "9><: {rn {mj {mi {i {rn {mh 90-(>-; ,6183: |") (1 . "300/ {rn {rn {mg
{nl |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V V V V |") (1 . "<0:;8: {rn {mf
{lo {ln {lm o {nk {rn |") (1 . "<0:;8: {rn {ll {ni {lk {lj o {no {rn |") (1 . "<0:;8: {rn {ni {ll {li {lh n {no
{rn |") (1 . "<0:;8: {rn {lg {lf {ni {mk n {mi {rn |") (1 . ":;8: {rn {ko {kn {ml {km o |") (1 . "9><: {kl {kk
{kj {i {rn {ki 90-(>-; ,6183: |") (1 . "300/ {rn {rn {ml {nh |") (1 . "<01:r,*-9><: {rn noo njo o o n o nj o o
n V V o n o V V V V |") (1 . "<0:;8: {rn {li {li {lg {kh o {ng {rn |") (1 . "<0:;8: {kg {kf {mo {jo {jn o {nk
{rn |") (1 . "<0:;8: {jm {mo {kf {jl {jk o {nk {rn |") (1 . "<0:;8: {rn {lf {jj {mo {lm n {mi {rn |") (1 . ":;8:
{rn {ji {jh {ln {jg o |") (1 . "<0:;8: {rn {mm {mn {jf {io o {no {rn |") (1 . "<0:;8: {rn {in {im {mn {lj n
{kj {rn |") (1 . ":;8: {rn {kn {il {lk {ik o |") (1 . "<0:;8: {rn {mg {mg {mm {lh o {ng {rn |") (1 . ":;8: {rn
{ko {ij {mm {ii o |") (1 . "<0:;8: {rn {jf {ml {mg {kh n {mi {rn |") (1 . "<0:;8: {rn {ml {ln {in {ih n {mi
{rn |") (1 . "):-+:' {rn {mk {ig |") (1 . "):-+:' {rn {lj {if |") (1 . ",+->687+r<*-): {rn fnqkfloflikihjhfgk noo
nmqljkkjkklnlgjohf o rn o V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {mj {l {rn |") (1 . "9><: {ho {rn
{hn {i {rn {hm 90-(>-; ,6183: |") (1 . "300/ {rn {rn {lk {mj |") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn
o o o V V V V |") (1 . ":;8: {rn {ij {ko {lg {hl o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mf |") (1 . "<0:;8:
{rn {lo {mf {hk {hj o {nk {rn |") (1 . "<0:;8: {rn {hi {hi {mf {jn n {hn {rn |") (1 . ":;8: {rn {jh {hh {jo
{hg o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {lo |") (1 . "<0:;8: {rn {im {in {lo {jk n {kj {rn |") (1 . ":;8:
{rn {hf {ji {jl {go o |") (1 . "<0:;8: {rn {ln {hk {hi {gn n {mi {rn |") (1 . "):-+:' {rn {ih {gm |") (1 . "):-+:'
{rn {in {gl |") (1 . ",+->687+r<*-): {rn ghqmgmiogifjijmngi njo hqfjknnjgnjgjgffg o n o V V |") (1 .
"<0:;8: {rn {gk {lg {ll {io n {mi {rn |") (1 . ":;8: {rn {il {ij {jf {gj o |") (1 . "<0:;8: {rn {jl {lk {lf {ih o {kj
{rn |") (1 . "<0:;8: {rn {lk {jl {gk {gi o {kj {rn |") (1 . "):-+:' {rn {gi {gh |") (1 . ":336/,:r<*-): {rn non noo
mk o rn o njqollmfilhglhmfog o o n V V |") (1 . "):-+:' {rn {io {gg |") (1 . ":336/,:r<*-): {rn non njo mk o
rn o njqollmfilhglhmfog o o n V V |") (1 . ":;8: {rn {kn {ji {in {gf o |") (1 . "/061+ {rn fnqkfloflikihjhfgk
njo nmqljkkjkklnlgjohf |") (1 . "/061+ {rn fnqkfloflikihjhfgk noo nmqljkkjkklnlgjohf |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {kk {l {rn |") (1 . "300/ {rn {rn {jo {kk |") (1 . "/3>1:r,*-9><: {rn noo moo
o o n o n o o o V V V V |") (1 . ":336/,:r<*-): {rn noo njo o o rn o nj o o n V V |") (1 . "<0:;8: {rn {jj {gk
{kf {hj n {mi {rn |") (1 . ":;8: {rn {hh {hf {hk {fo o |") (1 . "<0:;8: {rn {jo {jo {jj {gn o {hn {rn |") (1 . "):-
+:' {rn {gn {fn |") (1 . ":336/,:r<*-): {rn hh moo o o n o rnl o o n V V |") (1 . "):-+:' {rn {hj {fm |") (1 .
":336/,:r<*-): {rn hh noo o o rn o rnl o o n V V |") (1 . ":;8: {rn {jh {hh {hi {fl o |") (1 . "/061+ {rn
ghqmgmiogifjijmngi noo hqfjknnjgnjgjgffg |") (1 . "/061+ {rn ghqmgmiogifjijmngi moo
hqfjknnjgnjgjgffg |") (1 . "<0:;8: {rn {hk {jf {im {gi n {mi {rn |") (1 . ",+->687+r<*-): {rn
nofqjojnhlmjofgf noo nnqiolfjnnnkjkmnmh o n o V V |") (1 . ":;8: {rn {hf {il {im {gf o |") (1 . "/061+ {rn
nofqjojnhlmjofgf noo nnqiolfjnnnkjkmnmh |") (1 . "/061+ {rn nofqjojnhlmjofgf njo nnqiolfjnnnkjkmnmh
|") (1 . ":336/,:r<*-): {rn noo noo o o rn o nj o o n V V |") (1 . ",+->687+r<*-): {rn ghqmgmiogifjijmngi
njo rhqfjknnjgnjgjgffg o rn o V V |") (1 . "/061+ {rn ghqmgmiogifjijmngi moo rhqfjknnjgnjgjgffg |") (1 .
"/061+ {rn ghqmgmiogifjijmngi noo rhqfjknnjgnjgjgffg |") (1 . ":336/,:r<*-): {rn noo moo o o n o nj o o n
V V |"))
)
```

```
; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
```

```
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 40 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "9C") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi nlm n o        ") (1 . "=0;& {n {m {rn {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k |") (1 . "3*2/ {j {rn {i {o |") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k |") (1 . ",7:33 {h {rn {rn {g {m |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k |") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: |") (1 . "9><: {rn {nm {nl {i {rn {nk -:):-,:; ,6183: |") (1 . "300/ {rn {rn {nj
{g |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V V V V |") (1 . "9><: {ni {nh {ng
{i {rn {nf -:):-,:; ,6183: |") (1 . "300/ {rn {rn {mo {f |") (1 . "<01:r,*-9><: {rn non noo mk o n o
njqollmfilhglhmfog o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mn {mn {mm {ml o {no {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {nm {l {rn |") (1 . "9><: {mk {mj {mi {i {rn {mh -:):-,:; ,6183: |") (1 .
"300/ {rn {rn {mg {nm |") (1 . "<01:r,*-9><: {rn nmm njo o o rn o i o o n V V o n o V V V V |") (1 .
"<0:;8: {rn {mf {lo {ln {lm o {nl {rn |") (1 . "<0:;8: {rn {nj {nj {mf {ll o {no {rn |") (1 . "<0:;8: {rn {ln
{lk {nj {ml n {lj {rn |") (1 . ":;8: {rn {li {lh {mm {lg o |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {nh {l {rn
|") (1 . "9><: {lf {ko {kn {i {rn {km 90-(>-; ,6183: |") (1 . "300/ {rn {rn {kl {nh |") (1 . "<01:r,*-9><: {rn
hh njo o o n o rnl o o n V V o n o V V V V |") (1 . "<0:;8: {rn {kk {kj {ki {kh o {ng {rn |") (1 . "<0:;8: {rn
{kg {mo {mn {ll n {nl {rn |") (1 . "<0:;8: {rn {mo {kg {kf {jo o {nl {rn |") (1 . "<0:;8: {rn {jn {mm {mo
{lm n {lj {rn |") (1 . ":;8: {rn {jm {li {ln {jl o |") (1 . ":;8: {rn {lh {li {mf {jk o |") (1 . "<0:;8: {rn {mm {jj
{kg {ji n {lj {rn |") (1 . "300/ {rn {rn {ln {jh |") (1 . "):-+:' {rn {lm {jg |") (1 . "):-+:' {rn {ji {jf |") (1 .
":336/,:r<*-): {rn noo njo o o rn o mo o o n V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {mj {l {rn |") (1 .
"9><: {io {in {im {i {rn {il 90-(>-; ,6183: |") (1 . "300/ {rn {rn {ik {mj |") (1 . "/3>1:r,*-9><: {rn noo
moo o o n o n o o o V V V V |") (1 . "<0:;8: {rn {ij {ii {ih {ig o {mi {rn |") (1 . "<0:;8: {if {ho {mg {hn
{hm o {ng {rn |") (1 . "<0:;8: {hl {mg {ho {ik {hk o {ng {rn |") (1 . "<0:;8: {rn {hj {jn {mg {kh n {lj {rn
|") (1 . ":;8: {rn {hi {hh {ki {hg o |") (1 . "<0:;8: {rn {lo {mf {lk {ji o {nl {rn |") (1 . "<0:;8: {rn {hf {go
{lo {jo n {im {rn |") (1 . ":;8: {rn {gn {jm {kf {gm o |") (1 . "<0:;8: {rn {ki {ln {go {gl n {lj {rn |") (1 .
"):-+:' {rn {gl {gk |") (1 . ",+->687+r<*-): {rn nnlqnnjmmimlgmjinn noo njqofflijjhlkoiook o n o V V |")
(1 . ":336/,:r<*-): {rn non njo mk o rn o njqollmfilhglhmfog o o n V V |") (1 . "<0:;8: {rn {lk {ih {hf {gj n
{lj {rn |") (1 . ":;8: {rn {lh {gn {lk {gi o |") (1 . "9><: {rn {rn {lj {i {rn {gh 90-(>-; ,6183: |") (1 . "/061+
{rn nnlqnnjmmimlgmjinn njo njqofflijjhlkoiook |") (1 . "/061+ {rn ggqngillljjlhhniok njo
niqnlgoifkljmjfjmk |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ko {l {rn |") (1 . "9><: {rn {jh {gg {i {rn {gh
90-(>-; ,6183: |") (1 . "300/ {rn {rn {kf {ko |") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V
|") (1 . "<0:;8: {rn {gf {fo {kj {hk n {kn {rn |") (1 . "<0:;8: {fn {fm {kl {fl {fk o {mi {rn |") (1 . "<0:;8: {fj
{kl {fm {fi {fh o {mi {rn |") (1 . "<0:;8: {rn {jj {hj {kl {ig n {lj {rn |") (1 . ":;8: {rn {fg {ff {ih {noo o |")
(1 . "<0/>-r3(;r>++-6= {rn {rn {rn {kk |") (1 . "<0:;8: {rn {kj {kk {non {nom o {ng {rn |") (1 . "<0:;8: {rn
{go {nol {kk {hm n {im {rn |") (1 . ":;8: {rn {hh {nok {hn {noj o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn
{kj |") (1 . ":;8: {rn {noi {hi {ik {noh o |") (1 . "<0:;8: {rn {ih {ki {fo {nog n {lj {rn |") (1 . "):-+:' {rn
{nog {nof |") (1 . "):-+:' {rn {hm {nno |") (1 . ",+->687+r<*-): {rn nnfqmhmhmhmhmhmhmh njo
jqlkklknmjifgnkhfh o rn o V V |") (1 . "<0:;8: {rn {fi {kf {jj {gj o {im {rn |") (1 . "<0:;8: {rn {kf {hn {jn
{gl o {im {rn |") (1 . "):-+:' {rn {ji {nnn |") (1 . ":336/,:r<*-): {rn non noo mk o rn o njqollmfilhglhmfog o
o n V V |") (1 . ":;8: {rn {hh {jm {go {nnm o |") (1 . "/061+ {rn nnlqnnjmmimlgmjinn noo
njqofflijjhlkoiook |") (1 . ":;8: {rn {gn {fg {hf {nnm o |") (1 . ",+->687+r<*-): {rn ggqngillljjlhhniok noo
niqnlgoifkljmjfjmk o rn o V V |") (1 . "<01:r,*-9><: {rn noo njo o o n o mo o o n V V o n o V V V V |")
(1 . "300/ {rn {rn {non {in |") (1 . "<0:;8: {rn {fl {ik {nnl {nnk o {kn {rn |") (1 . "<0:;8: {rn {ik {fl {hj
{nog o {kn {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {ij |") (1 . "<0:;8: {rn {ii {ij {nnj {nni o {mi {rn |")
(1 . "<0:;8: {rn {fo {gf {ij {fk n {kn {rn |") (1 . ":;8: {rn {ff {nnh {fl {nng o |") (1 . "<0/>-r3(;r>++-6= {rn
{rn {rn {ii l") (1 . "<0:;8: {rn {nol {hf {ii {fh n {im {rn l") (1 . ":;8: {rn {nnf {fg {fi {nmo o l") (1 . "):-+:'
{rn {gj {nmn l") (1 . "):-+:' {rn {fk {nmm l") (1 . ",+->687+r<*-): {rn glqkhgmiogifjijmnf njo
nnqmhogjllggjljknk o n o V V l") (1 . "<0:;8: {rn {nml {nnl {ho {nom n {gg {rn l") (1 . ":;8: {rn {nok
{noi {non {nmk o l") (1 . "<0:;8: {rn {hn {fi {nml {nmj o {im {rn l") (1 . "):-+:' {rn {nmj {nmi l") (1 .
":336/,:r<*-): {rn nmm noo o o rn o i o o n V V l") (1 . "):-+:' {rn {nom {nmh l") (1 . ":336/,:r<*-): {rn
nmm moo o o n o i o o n V V l") (1 . ":;8: {rn {ff {hi {fo {nmg o l") (1 . "/061+ {rn
nnfqmhmhmhmhmhmhmh moo jqlkklknmjifgnkhfh l") (1 . "/061+ {rn nnfqmhmhmhmhmhmhmh noo
jqlkklknmjifgnkhfh l") (1 . "/061+ {rn ggqngillljjlhhniok noo niqnlgoifkljmjfjmk l") (1 . ":336/,:r<*-): {rn
noo noo o o rn o mo o o n V V l") (1 . "<0:;8: {rn {non {nnj {gf {nnk n {gg {rn l") (1 . ":;8: {rn {noi {nnh
{gf {nmg o l") (1 . "<0:;8: {rn {nnl {nml {fm {nni n {gg {rn l") (1 . ":;8: {rn {nnh {nnf {nnj {nmf o l") (1
. "):-+:' {rn {nnk {nlo l") (1 . ":336/,:r<*-): {rn hh moo o o n o rnl o o n V V l") (1 . "):-+:' {rn {nni {nln
l") (1 . ":336/,:r<*-): {rn hh noo o o rn o rnl o o n V V l") (1 . "/061+ {rn glqkhgmiogifjijmnf noo
nnqmhogjllggjljknk l") (1 . "/061+ {rn glqkhgmiogifjijmnf moo nnqmhogjllggjljknk l") (1 . "<0:;8: {rn
{nnj {non {nol {nmj n {gg {rn l") (1 . ",+->687+r<*-): {rn nnfqmhmhmhmhmhmhmh njo
rjqlkklknmjifgnkhfh o n o V V l") (1 . ":;8: {rn {nnf {nok {nol {nnm o l") (1 . "/061+ {rn
nnfqmhmhmhmhmhmhmh noo rjqlkklknmjifgnkhfh l") (1 . "/061+ {rn nnfqmhmhmhmhmhmhmh moo
rjqlkklknmjifgnkhfh l") (1 . ":336/,:r<*-): {rn noo moo o o n o mo o o n V V l") (1 . ",+->687+r<*-): {rn
glqkhgmiogifjijmnf njo rnnqmhogjllggjljknk o rn o V V l") (1 . "/061+ {rn glqkhgmiogifjijmnf moo
rnnqmhogjllggjljknk l") (1 . "/061+ {rn glqkhgmiogifjijmnf noo rnnqmhogjllggjljknk l"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
```

```
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 50 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "9E") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi nif n o        ") (1 . "=0;& {n {m {rn {rn l") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k l") (1 . "3*2/ {j {rn {i {o l") (1 . "-:961:2:l+ {rn o o o o
oqknfffffgiggifhgnj o o o m l") (1 . "):-+:'@+:2/3>+: {rn l o n g l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k l") (1 . ",7:33 {h {rn {rn {g {m l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k l") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: l") (1 . "9><: {rn {nm {nl {i {rn {nk -:):-,:; ,6183: l") (1 . "300/ {rn {rn {nj
{g l") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V V V V l") (1 . "9><: {rn {ni {nh
{i {rn {ng -:):-,:; ,6183: l") (1 . "300/ {rn {rn {nf {f l") (1 . "<01:r,*-9><: {rn nno noo rmf o n o h o o n V
V o n o V V V V l") (1 . "<0:;8: {rn {mo {mo {mn {mm o {no {rn l") (1 . "9><: {rn {ml {mk {i {rn {mj -
:):-,:; ,6183: l") (1 . "300/ {rn {rn {mi {nm l") (1 . "/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V V V
l") (1 . "<0:;8: {rn {mh {mg {mf {lo o {nl {rn l") (1 . "<0:;8: {rn {nj {nj {ln {lm o {no {rn l") (1 . "<0:;8:
{rn {ll {lk {nj {mm n {lj {rn l") (1 . ":;8: {rn {li {lh {mn {lg o l") (1 . "9><: {lf {ko {kn {i {rn {km -:):-,:;
,6183: l") (1 . "300/ {rn {rn {kl {ni l") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n
V V o n o V V V V l") (1 . "<0:;8: {rn {kk {kk {kj {ki o {nh {rn l") (1 . "<0:;8: {rn {kh {nf {kg {kf o {nl
{rn l") (1 . "<0:;8: {rn {nf {kh {kk {jo n {nl {rn l") (1 . "<0:;8: {rn {kj {jn {nf {lo n {jm {rn l") (1 . ":;8:
{rn {jl {jk {mf {jj o l") (1 . "<0:;8: {rn {ji {kl {mo {lm n {mk {rn l") (1 . ":;8: {rn {lh {li {ln {jh o l") (1 .
"<0:;8: {rn {jg {mn {kl {jf n {lj {rn l") (1 . "<0:;8: {rn {mn {io {ji {in n {lj {rn l") (1 . "300/ {rn {rn {ll
{im l") (1 . "):-+:' {rn {jf {il l") (1 . "):-+:' {rn {in {ik l") (1 . ":336/,:r<*-): {rn noo njo o o rn o mj o o n V
V l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ml {l {rn l") (1 . "9><: {ij {ii {ih {i {rn {ig -:):-,:; ,6183: l") (1
. "300/ {rn {rn {if {ml l") (1 . "<01:r,*-9><: {rn nmm njo o o rn o i o o n V V o n o V V V V l") (1 .
"<0:;8: {rn {ln {ho {ll {jf o {mk {rn l") (1 . "<0:;8: {rn {mi {mi {mg {jo o {nh {rn l") (1 . "<0:;8: {rn {hn
{mf {mi {ki n {jm {rn l") (1 . ":;8: {rn {hm {jl {kj {hl o l") (1 . "<0:;8: {rn {mg {mh {hn {hk o {nl {rn l")
(1 . "<0:;8: {rn {hj {hi {mh {kf n {hh {rn l") (1 . ":;8: {rn {jk {hg {kg {hf o l") (1 . ":;8: {rn {jl {hm {mg
{go o l") (1 . "<0:;8: {rn {mf {gn {hj {gm n {jm {rn l") (1 . "300/ {rn {rn {mf {gl l") (1 . "):-+:' {rn {lo
```

{gk l") (1 . "):-+:' {rn {kf {gj l") (1 . ",+->687+r<*-): {rn nnnqjkhilgjifflffg noo rmmqnhlmmgohfllnolf o
rn o V V l") (1 . "<0:;8: {rn {ho {ln {lk {in o {mk {rn l") (1 . ":336/,:r<*-): {rn non njo mk o rn o
njqollmfilhglhmfog o o n V V l") (1 . "<0:;8: {rn {gi {ll {gh {gg n {lj {rn l") (1 . ":;8: {rn {gf {li {ll {fo o
l") (1 . "<0:;8: {rn {lk {fn {fm {fl n {lj {rn l") (1 . ":;8: {rn {lh {fk {lk {fj o l") (1 . "9><: {rn {rn {lj {i {rn
{fi 90-(>-; ,6183: l") (1 . "/061+ {rn nnjqkomiglhgggjnnf njo nfqifnjjkgkmnlnnfh l") (1 . "/061+ {rn
giqmgggmkononogfjk njo moqfokilmllmfnmnmf l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ko {l {rn l") (1
. "9><: {fh {fg {ff {i {rn {noo 90-(>-; ,6183: l") (1 . "300/ {rn {rn {non {ko l") (1 . "<01:r,*-9><: {rn hh
njo o o n o rnl o o n V V o n o V V V V l") (1 . "<0:;8: {rn {nom {nol {gi {nok o {kn {rn l") (1 . "<0:;8:
{rn {kl {ji {noj {noi o {mk {rn l") (1 . "<0:;8: {rn {noh {kj {kh {hk n {jm {rn l") (1 . "):-+:' {rn {hk {nog
l") (1 . ":336/,:r<*-): {rn noo nhj o o rn o mj o o n V V l") (1 . ":;8: {rn {hg {hm {hn {nof o l") (1 . "<0:;8:
{rn {nno {kg {jn {gm o {hh {rn l") (1 . "<0:;8: {rn {kg {nnn {noh {nnm o {hh {rn l") (1 . "300/ {rn {rn
{noj {fg l") (1 . "):-+:' {rn {nnm {nnl l") (1 . ":336/,:r<*-): {rn nno noo rmf o rn o h o o n V V l") (1 .
":336/,:r<*-): {rn nno nhj rmf o rn o h o o n V V l") (1 . "<0:;8: {rn {jn {nnk {nnj {nni n {jm {rn l") (1 .
":;8: {rn {jk {nnh {hj {nng o l") (1 . "9><: {rn {im {jm {i {rn {fi 90-(>-; ,6183: l") (1 . "/061+ {rn
nnnqjkhilgjifflffg nhj rmmqnhlmmgohfllnolf l") (1 . "/061+ {rn nnnqjkhilgjifflffg noo
rmmqnhlmmgohfllnolf l") (1 . "<0:;8: {rn {nnf {jg {if {nok n {lj {rn l") (1 . "<0:;8: {rn {noj {nno {jg {gg
o {hh {rn l") (1 . ":;8: {rn {nmo {gf {gh {nng o l") (1 . "):-+:' {rn {gg {nmn l") (1 . ",+->687+r<*-): {rn
nnjqkomiglhgggjnnf noo nfqifnjjkgkmnlnnfh o n o V V l") (1 . "<0:;8: {rn {io {nnf {non {nmm n {lj {rn
l") (1 . "<0:;8: {rn {nnn {noj {io {fl o {hh {rn l") (1 . ":;8: {rn {fk {nml {fm {nng o l") (1 . "):-+:' {rn {in
{nmk l") (1 . ",+->687+r<*-): {rn giqmgggmkononogfjk noo moqfokilmllmfnmnmf o rn o V V l") (1 .
"<01:r,*-9><: {rn noo njo o o n o mj o o n V V o n o V V V V l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ii
{l {rn l") (1 . "9><: {nmj {gl {hh {i {rn {nmi 90-(>-; ,6183: l") (1 . "300/ {rn {rn {nmh {ii l") (1 .
"/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V l") (1 . "<0:;8: {rn {nmg {nmf {fn {nmm o {ih {rn
l") (1 . "<0:;8: {nlo {nnj {if {nno {nln o {kn {rn l") (1 . "<0:;8: {nlm {if {nnj {nmh {nll o {kn {rn l") (1 .
":;8: {rn {nlk {nmo {gi {nlj o l") (1 . "<0:;8: {rn {fm {gh {ho {noi n {hh {rn l") (1 . ":;8: {rn {fk {gf {noj
{nli o l") (1 . "<0:;8: {rn {nlh {hn {hi {nnm n {jm {rn l") (1 . "/061+ {rn nokqjhljofnkjmjiim nhj
rmkqjhgnoomfkhlfnom l") (1 . ",+->687+r<*-): {rn nokqjhljofnkjmjiim noo rmkqjhgnoomfkhlfnom o n o
V V l") (1 . "<0:;8: {rn {gh {hj {nom {nln n {hh {rn l") (1 . "<0:;8: {rn {hi {fm {nmf {nlg n {hh {rn l") (1
. ":;8: {rn {nlf {hg {hi {nng o l") (1 . "/061+ {rn nokqjhljofnkjmjiim noo rmkqjhgnoomfkhlfnom l") (1 .
"<0:;8: {rn {gn {nlh {nko {nkn n {jm {rn l") (1 . "<0:;8: {rn {nol {nom {gn {nni o {kn {rn l") (1 . ":;8:
{rn {nnh {nkm {gn {nkl o l") (1 . "):-+:' {rn {gm {nkk l") (1 . ":336/,:r<*-): {rn noo noo o o rn o mj o o n
V V l") (1 . "<0:;8: {rn {fn {gi {nkj {nki n {lj {rn l") (1 . "):-+:' {rn {nln {nkh l") (1 . "/061+ {rn
nnjqkomiglhgggjnnf noo nfqifnjjkgkmnlnnfh l") (1 . ":;8: {rn {nml {nkg {fn {nkf o l") (1 . "):-+:' {rn {fl
{njo l") (1 . "/061+ {rn giqmgggmkononogfjk noo moqfokilmllmfnmnmf l") (1 . "-:9@)+r3(;r>++-6= {rn
{rn {rn {fg {l {rn l") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V l") (1 . "<0:;8: {rn {nko
{nkj {nol {nll n {ff {rn l") (1 . "<0:;8: {njn {njm {non {njl {njk o {ih {rn l") (1 . "<0:;8: {njj {non {njm
{nnn {nlg o {ih {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nom l") (1 . ":;8: {rn {nmo {nnh {nno {nji o
l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nol l") (1 . ":;8: {rn {nkm {nlk {nmh {njh o l") (1 . "):-+:' {rn {nki
{njg l") (1 . ",+->687+r<*-): {rn nmkqlgilililililil njo jqjojommnmkgkjhii o rn o V V l") (1 . ":336/,:r<*-):
{rn non noo mk o rn o njqollmfilhglhmfog o o n V V l") (1 . "<0:;8: {rn {nnk {noh {njm {njf n {jm {rn l")
(1 . ":;8: {rn {nlf {nml {nnn {nio o l") (1 . "):-+:' {rn {njf {nin l") (1 . "<0:;8: {rn {njl {nmh {nnk {nkn o
{ff {rn l") (1 . ":;8: {rn {nkm {nim {nko {nil o l") (1 . "):-+:' {rn {nni {nik l") (1 . ",+->687+r<*-): {rn
nmkqlgilililililil njo rjqjojommnmkgkjhii o n o V V l") (1 . "/061+ {rn nmkqlgilililililil noo
rjqjojommnmkgkjhii l") (1 . "<0:;8: {rn {nmh {njl {nnf {nki o {ff {rn l") (1 . ":;8: {rn {nkg {nlk {nkj {nil
o l") (1 . "/061+ {rn nmkqlgilililililil noo jqjojommnmkgkjhii l") (1 . "):-+:' {rn {njk {nij l") (1 . ",+-
>687+r<*-): {rn hgqjgifjijmnhlfnmj njo nmqfomhhliflfgngmg o n o V V l") (1 . "/061+ {rn
hgqjgifjijmnhlfnmj noo nmqfomhhliflfgngmg l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nmg l") (1 . "<0:;8:
{rn {nmf {nmg {nlh {njf o {ih {rn l") (1 . "<0:;8: {rn {nkj {nko {nmg {njk n {ff {rn l") (1 . ":;8: {rn {nkg
{nim {njl {nii o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nmf l") (1 . ":336/,:r<*-): {rn nmm noo o o rn o i o
o n V V l") (1 . ":336/,:r<*-): {rn nmm moo o o n o i o o n V V l") (1 . "/061+ {rn nmkqlgilililililil moo
jqjojommnmkgkjhii l") (1 . ":;8: {rn {nim {nlf {nlh {nih o l") (1 . ":336/,:r<*-): {rn hh noo o o rn o rnl o o
n V V l") (1 . "/061+ {rn hgqjgifjijmnhlfnmj noo rnmqfomhhliflfgngmg l") (1 . "):-+:' {rn {nkn {nig l") (1
. ":336/,:r<*-): {rn noo moo o o n o mj o o n V V l") (1 . "/061+ {rn nmkqlgilililililil moo
rjqjojommnmkgkjhii l") (1 . "/061+ {rn hgqjgifjijmnhlfnmj moo nmqfomhhliflfgngmg l") (1 . ":336/,:r<*-
): {rn hh moo o o n o rnl o o n V V l") (1 . ",+->687+r<*-): {rn hgqjgifjijmnhlfnmj njo
rnmqfomhhliflfgngmg o rn o V V l") (1 . "/061+ {rn hgqjgifjijmnhlfnmj moo rnmqfomhhliflfgngmg l"))
)

```
; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")
```

```
; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 60 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "9D") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi njm n o        ") (1 . "=0;& {n {m {rn {rn l") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k l") (1 . "3*2/ {j {rn {i {o l") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m l") (1 . "):-+:'@+:2/3>+: {rn l o n g l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k l") (1 . ",7:33 {h {rn {rn {g {m l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k l") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: l") (1 . "9><: {rn {nm {nl {i {rn {nk -:):-,:; ,6183: l") (1 . "300/ {rn {rn {nj
{g l") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V V V V l") (1 . "9><: {ni {nh {ng
{i {rn {nf -:):-,:; ,6183: l") (1 . "300/ {rn {rn {mo {f l") (1 . "<01:r,*-9><: {rn nno noo rmf o n o h o o n V
V o n o V V V V l") (1 . "<0:;8: {rn {mn {mn {mm {ml o {no {rn l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{nm {l {rn l") (1 . "9><: {rn {mk {mj {i {rn {mi -:):-,:; ,6183: l") (1 . "300/ {rn {rn {mh {nm l") (1 .
"<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V l") (1 . "<0:;8: {rn {mg {mf {lo {ln o {nl
{rn l") (1 . "<0:;8: {rn {nj {nj {lm {ll o {no {rn l") (1 . "<0:;8: {rn {lk {lj {nj {ml n {li {rn l") (1 . ":;8: {rn
{lh {lg {mm {lf o l") (1 . "9><: {rn {ko {kn {i {rn {km -:):-,:; ,6183: l") (1 . "300/ {rn {rn {kl {nh l") (1 .
"/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V V V l") (1 . "<0:;8: {rn {kk {kj {ki {kh o {ng {rn l") (1 .
"<0:;8: {rn {kg {mo {kf {jo o {nl {rn l") (1 . "<0:;8: {rn {mo {kg {jn {jm n {nl {rn l") (1 . "<0:;8: {rn {jl
{jk {mo {ln n {li {rn l") (1 . ":;8: {rn {jj {ji {lo {jh o l") (1 . "<0:;8: {rn {jg {jf {mn {ll n {kn {rn l") (1 .
":;8: {rn {lg {lh {lm {io o l") (1 . "<0:;8: {rn {jk {mm {jf {in n {li {rn l") (1 . "<0:;8: {rn {mm {im {jg {il
n {li {rn l") (1 . "300/ {rn {rn {lk {ik l") (1 . "):-+:' {rn {in {ij l") (1 . "):-+:' {rn {il {ii l") (1 . ":336/,:r<*-):
{rn noo njo o o rn o lo o o n V V l") (1 . "9><: {ih {ig {if {i {rn {ho -:):-,:; ,6183: l") (1 . "300/ {rn {rn {jf
{mk l") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n V V o n o V V V V l") (1 .
"<0:;8: {rn {jn {jn {jl {hn o {mj {rn l") (1 . "<0:;8: {hm {hl {mh {hk {hj o {ng {rn l") (1 . "<0:;8: {hi {mh
{hl {hh {hg o {ng {rn l") (1 . "<0:;8: {rn {im {hf {mh {kh n {li {rn l") (1 . ":;8: {rn {go {gn {ki {gm o l")
(1 . "<0:;8: {rn {mf {mg {gl {gk o {nl {rn l") (1 . "<0:;8: {rn {gj {gi {mg {jo n {gh {rn l") (1 . ":;8: {rn {ji
{gg {kf {gf o l") (1 . "<0:;8: {rn {kl {kl {mf {jm o {mj {rn l") (1 . ":;8: {rn {jj {fo {mf {fn o l") (1 .
"<0:;8: {rn {gl {lo {kl {hn n {li {rn l") (1 . "<0:;8: {rn {lo {lk {gj {fm n {li {rn l") (1 . "):-+:' {rn {ln {fl
l") (1 . "):-+:' {rn {jo {fk l") (1 . ",+->687+r<*-): {rn nnjqffmkkfiiglonif noo rmjqlgnfnlfohkgmnhm o rn o
V V l") (1 . "<0:;8: {rn {fj {lm {lj {il o {kn {rn l") (1 . "<0:;8: {rn {lm {fj {lk {in o {kn {rn l") (1 .
":336/,:r<*-): {rn non njo mk o rn o njqollmfilhglhmfog o o n V V l") (1 . ":;8: {rn {fi {lh {lk {fh o l") (1 .
"<0:;8: {rn {lj {ki {fg {ff n {li {rn l") (1 . ":;8: {rn {lg {noo {lj {non o l") (1 . "9><: {rn {rn {li {i {rn
{nom 90-(>-; ,6183: l") (1 . "/061+ {rn nnjqfigmlnkllijlki njo mjqlfhnjhomljfhhhn l") (1 . "/061+ {rn
giqnffghfkgkgfohmf njo miqilhjojomnkimggi l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ko {l {rn l") (1 .
"9><: {nol {nok {noj {i {rn {noi 90-(>-; ,6183: l") (1 . "300/ {rn {noh {nog {ko l") (1 . "<01:r,*-9><: {rn
nmm njo o o rn o i o o n V V o n o V V V V l") (1 . ":;8: {rn {fo {jj {jl {nof o l") (1 . "<0/>-r3(;r>++-6=
{rn {rn {rn {kk l") (1 . "<0:;8: {rn {kj {kk {nno {nnn o {ng {rn l") (1 . "<0:;8: {rn {nnm {nnm {kk {hj n
{nnl {rn l") (1 . ":;8: {rn {gn {nnk {hk {nnj o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {kj l") (1 . "<0:;8: {rn
{gi {fg {kj {hg n {gh {rn l") (1 . ":;8: {rn {nni {go {hh {nnh o l") (1 . "<0:;8: {rn {ki {nno {nnm {nng n
{li {rn l") (1 . "):-+:' {rn {ff {nnf l") (1 . "):-+:' {rn {hj {nmo l") (1 . ",+->687+r<*-): {rn
hmqiogifjijmnhlfoh njo nmqmljgihnffjlflgj o n o V V l") (1 . "<0:;8: {rn {nmn {jl {kg {gk n {li {rn l") (1 .
":;8: {rn {gg {fo {gl {nmm o l") (1 . "<0:;8: {rn {nml {kf {jk {fm o {gh {rn l") (1 . "<0:;8: {rn {kf {hh
{nmn {nmk o {gh {rn l") (1 . "300/ {rn {rn {nml {ig l") (1 . "):-+:' {rn {nmk {nmj l") (1 . ":336/,:r<*-):
{rn nno noo rmf o rn o h o o n V V l") (1 . "):-+:' {rn {gk {nmi l") (1 . ":336/,:r<*-): {rn nno nhj rmf o rn o
h o o n V V l") (1 . ":;8: {rn {ji {fi {gj {nmh o l") (1 . "/061+ {rn nnjqffmkkfiiglonif nhj
rmjqlgnfnlfohkgmnhm l") (1 . "/061+ {rn nnjqffmkkfiiglonif noo rmjqlgnfnlfohkgmnhm l") (1 . "<0:;8:
{rn {jf {jg {nml {nmg o {kn {rn l") (1 . "):-+:' {rn {fm {nmf l") (1 . ",+->687+r<*-): {rn
nnjqfigmlnkllijlki noo mjqlfhnjhomljfhhhn o n o V V l") (1 . "<0:;8: {rn {hh {nml {im {ff o {gh {rn l") (1
```

```
. ":;8: {rn {noo {go {fg {nmh o l") (1 . "):-+:' {rn {il {nlo l") (1 . ",+->687+r<*-): {rn giqnffghfkgkgfohmf
noo miqilhjojomnkimggi o rn o V V l") (1 . "<01:r,*-9><: {rn noo njo o o n o lo o o n V V o n o V V V V
l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ig {l {rn l") (1 . "9><: {nln {ik {nlm {i {rn {nll 90-(>-; ,6183: l")
(1 . "300/ {rn {gh {nlk {ig l") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V l") (1 . "300/ {rn
{rn {nlj {ko l") (1 . "<0:;8: {nli {nog {nog {nlh {nlg o {if {rn l") (1 . ":336/,:r<*-): {rn noo nhj o o rn o lo
o o n V V l") (1 . "<0:;8: {rn {hf {nmn {hl {nnn n {li {rn l") (1 . ":;8: {rn {nnk {nni {nno {nlf o l") (1 .
"<0:;8: {rn {hk {hk {hf {nng o {nnl {rn l") (1 . "300/ {rn {rn {hk {nok l") (1 . "):-+:' {rn {nng {nko l") (1 .
":336/,:r<*-): {rn hh moo o o n o rnl o o n V V l") (1 . "):-+:' {rn {nnn {nkn l") (1 . ":336/,:r<*-): {rn hh
noo o o rn o rnl o o n V V l") (1 . ":;8: {rn {gn {nnk {nnm {nkm o l") (1 . "/061+ {rn hmqiogifjijmnhlfoh
noo nmqmljgihnffjlflgj l") (1 . "/061+ {rn hmqiogifjijmnhlfoh moo nmqmljgihnffjlflgj l") (1 . "<0:;8: {rn
{nno {gl {gi {nmk n {li {rn l") (1 . ",+->687+r<*-): {rn nolqojnnmnoonnfggk noo rmfqgkkkknolkoiflhi o
n o V V l") (1 . "<0:;8: {rn {fg {gj {fj {nmg n {gh {rn l") (1 . ":;8: {rn {nni {gg {gi {nmh o l") (1 . "/061+
{rn nolqojnnmnoonnfggk noo rmfqgkkkknolkoiflhi l") (1 . "/061+ {rn nolqojnnmnoonnfggk nhj
rmfqgkkkknolkoiflhi l") (1 . ":336/,:r<*-): {rn noo noo o o rn o lo o o n V V l") (1 . ":;8: {rn {noo {fi
{nml {nkl o l") (1 . "/061+ {rn nnjqfigmlnkllijlki noo mjqlfhnjhomljfhhhn l") (1 . "/061+ {rn
giqnffghfkgkgfohmf noo miqilhjojomnkimggi l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {nok {l {rn l") (1 .
"300/ {rn {nnl {nlh {nok l") (1 . "/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V l") (1 . "<0:;8: {rn
{nlk {nlk {nlj {nkk n {noj {rn l") (1 . "<0:;8: {nkj {nlj {nlj {nlk {nkk o {noh {rn l") (1 . "<0/>-r3(;r>++-
6= {rn {rn {rn {nog l") (1 . "<0:;8: {rn {nlh {nlh {nog {nlg n {nlm {rn l") (1 . ":;8: {rn {nki {nki {nlh
{nkh o l") (1 . ",+->687+r<*-): {rn hmqiogifjijmnhlfoh njo rnmqmljgihnffjlflgj o rn o V V l") (1 . "/061+
{rn hmqiogifjijmnhlfoh moo rnmqmljgihnffjlflgj l") (1 . "/061+ {rn hmqiogifjijmnhlfoh noo
rnmqmljgihnffjlflgj l") (1 . ":336/,:r<*-): {rn noo moo o o n o lo o o n V V l") (1 . ":336/,:r<*-): {rn non
noo mk o rn o njqollmfilhglhmfog o o n V V l") (1 . ":;8: {rn {nkg {nkg {nlk {nkf o l") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {nlj l") (1 . "):-+:' {rn {nlg {njo l") (1 . ":336/,:r<*-): {rn nmm moo o o n o i o o n
V V l") (1 . "):-+:' {rn {nkk {njn l") (1 . ":336/,:r<*-): {rn nmm noo o o rn o i o o n V V l") (1 . "/061+ {rn
nni moo kqmilmjiknkjioionn:ronk l") (1 . "/061+ {rn nni noo rkqmilmjiknkjioionn:ronk l"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)
```

```
; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 70 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "9F") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi njm n o        ") (1 . "=0;& {n {m {rn {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k |") (1 . "3*2/ {j {rn {i {o |") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k |") (1 . ",7:33 {h {rn {rn {g {m |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k |") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: |") (1 . "9><: {nm {nl {nk {i {rn {nj -:):-,:; ,6183: |") (1 . "300/ {rn {rn {ni
{g |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V V V V |") (1 . "-:9@)+r3(;r>++-
6= {rn {rn {rn {f {l {rn |") (1 . "9><: {rn {nh {ng {i {rn {nf -:):-,:; ,6183: |") (1 . "300/ {rn {rn {mo {f |")
(1 . "<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mn {mn {mm {ml o
{no {rn |") (1 . "9><: {rn {mk {mj {i {rn {mi -:):-,:; ,6183: |") (1 . "300/ {rn {rn {mh {nl |") (1 . "<01:r,*-
9><: {rn nno noo rmf o n o h o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mg {mf {lo {ln o {nk {rn |")
(1 . "<0:;8: {rn {ni {ni {lm {ll o {no {rn |") (1 . "<0:;8: {rn {lk {lj {ni {ml n {li {rn |") (1 . ":;8: {rn {lh {lg
{mm {lf o |") (1 . "9><: {rn {ko {kn {i {rn {km -:):-,:; ,6183: |") (1 . "300/ {rn {rn {kl {nh |") (1 .
"/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V V V |") (1 . "<0:;8: {rn {kk {kj {ki {kh o {ng {rn |") (1 .
"<0:;8: {kg {kf {mo {jo {jn o {nk {rn |") (1 . "<0:;8: {jm {mo {kf {jl {jk o {nk {rn |") (1 . "<0:;8: {rn {jj
{ji {mo {ln n {li {rn |") (1 . ":;8: {rn {jh {jg {lo {jf o |") (1 . "<0:;8: {rn {io {in {mn {ll n {kn {rn |") (1 .
":;8: {rn {lg {lh {lm {im o |") (1 . "<0:;8: {rn {il {mm {in {ik n {li {rn |") (1 . "<0:;8: {rn {mm {jj {io {ij
```

```
n {li {rn l") (1 . "300/ {rn {rn {lk {ii l") (1 . "):-+:' {rn {ik {ih l") (1 . "):-+:' {rn {ij {ig l") (1 . ":336/,:r<*-
): {rn noo njo o o rn o lj o o n V V l") (1 . "9><: {if {ho {hn {i {rn {hm -:):-,:; ,6183: l") (1 . "300/ {rn {rn
{in {mk l") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n V V o n o V V V V l") (1 .
"<0:;8: {rn {hl {hl {hk {hj o {mj {rn l") (1 . "<0:;8: {rn {hi {mh {hh {hg o {ng {rn l") (1 . "<0:;8: {rn {mh
{hi {hl {hf n {ng {rn l") (1 . "<0:;8: {rn {hk {il {mh {kh n {li {rn l") (1 . ":;8: {rn {go {gn {ki {gm o l") (1
. "<0/>-r3(;r>++-6= {rn {rn {rn {mg l") (1 . "<0:;8: {rn {mf {mg {gl {gk o {nk {rn l") (1 . "<0:;8: {rn {gj
{gj {mg {jn n {gi {rn l") (1 . ":;8: {rn {jg {gh {jo {gg o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mf l") (1 .
"<0:;8: {rn {gf {fo {mf {jk n {fn {rn l") (1 . ":;8: {rn {fm {jh {jl {fl o l") (1 . "<0:;8: {rn {lj {lo {fo {fk n
{li {rn l") (1 . "<0:;8: {rn {lo {gl {gj {fj n {li {rn l") (1 . "):-+:' {rn {fk {fi l") (1 . "):-+:' {rn {jn {fh l") (1 .
",+->687+r<*-): {rn ijqjklkhgmiogifjil njo iqnklfkfomijkgfiih o n o V V l") (1 . "<0:;8: {rn {fg {lm {lj {ij
o {kn {rn l") (1 . "<0:;8: {rn {lm {fg {lk {ik o {kn {rn l") (1 . ":336/,:r<*-): {rn non njo mk o rn o
njqollmfilhglhmfog o o n V V l") (1 . "<0:;8: {rn {ki {lk {ff {noo n {li {rn l") (1 . ":;8: {rn {non {lh {lk
{nom o l") (1 . ":;8: {rn {lg {nol {lj {nok o l") (1 . "9><: {rn {rn {li {i {rn {noj 90-(>-; ,6183: l") (1 .
"/061+ {rn nnlqjjjnlohhiioohf njo lmqmigjlimnhiknilf l") (1 . "/061+ {rn gfqnhimlglhknhfohf njo
llqmgklmlkonohjggn l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ko {l {rn l") (1 . "9><: {noi {noh {fn {i {rn
{nog 90-(>-; ,6183: l") (1 . "300/ {rn {nof {nno {ko l") (1 . "<01:r,*-9><: {rn nmm njo o o rn o i o o n V
V o n o V V V V l") (1 . "<0:;8: {rn {kl {kl {kj {hf o {mj {rn l") (1 . "<0:;8: {rn {nnn {ki {kl {hj n {li {rn
l") (1 . ":;8: {rn {nnm {go {hk {nnl o l") (1 . "<0:;8: {rn {kj {kk {nnn {nnk o {ng {rn l") (1 . "<0:;8: {rn
{ff {gf {kk {hg n {fn {rn l") (1 . ":;8: {rn {gn {nnj {hh {nni o l") (1 . ":;8: {rn {go {nnm {kj {nnh o l") (1 .
"):-+:' {rn {kh {nng l") (1 . "):-+:' {rn {hg {nnf l") (1 . ",+->687+r<*-): {rn nniqhghilomhognglj noo
rloqhnnninfhjjhfgh o rn o V V l") (1 . "<0:;8: {rn {ji {nmo {kf {gk n {li {rn l") (1 . ":;8: {rn {gh {fm {gl
{nmn o l") (1 . "<0:;8: {rn {jo {jo {ji {fj o {gi {rn l") (1 . "300/ {rn {rn {jo {noh l") (1 . "):-+:' {rn {fj
{nmm l") (1 . ":336/,:r<*-): {rn hh moo o o n o rnl o o n V V l") (1 . "<0:;8: {rn {hh {jl {nmo {nml o {fn
{rn l") (1 . "<0:;8: {rn {jl {nmk {jj {fk o {fn {rn l") (1 . "300/ {rn {nmj {nmk {ho l") (1 . "):-+:' {rn {gk
{nmi l") (1 . ":336/,:r<*-): {rn hh noo o o rn o rnl o o n V V l") (1 . ":;8: {rn {nol {jh {fo {nmh o l") (1 .
":;8: {rn {jg {gh {gj {nmg o l") (1 . "/061+ {rn ijqjklkhgmiogifjil noo iqnklfkfomijkgfiih l") (1 . "/061+
{rn ijqjklkhgmiogifjil moo iqnklfkfomijkgfiih l") (1 . "<0:;8: {rn {in {io {nmk {nmf o {kn {rn l") (1 .
"<0:;8: {rn {nmk {hh {il {noo o {fn {rn l") (1 . ":;8: {rn {gn {non {ff {nmh o l") (1 . "):-+:' {rn {noo {nlo
l") (1 . ",+->687+r<*-): {rn nnlqjjjnlohhiioohf noo lmqmigjlimnhiknilf o n o V V l") (1 . "):-+:' {rn {ij
{nln l") (1 . ",+->687+r<*-): {rn gfqnhimlglhknhfohf noo llqmgklmlkonohjggn o rn o V V l") (1 .
"<01:r,*-9><: {rn noo njo o o n o lj o o n V V o n o V V V V l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ho
{l {rn l") (1 . "9><: {nlm {ii {nll {i {rn {nlk 90-(>-; ,6183: l") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn
o o o V V V V l") (1 . "300/ {rn {rn {nlj {ko l") (1 . "<0:;8: {nli {nno {nno {nlh {nlg o {hn {rn l") (1 .
"<0:;8: {rn {nmo {hk {hi {nnk n {li {rn l") (1 . "):-+:' {rn {nnk {nlf l") (1 . ":336/,:r<*-): {rn noo nhj o o
rn o lj o o n V V l") (1 . ":;8: {rn {nnj {nnm {nnn {nko o l") (1 . "):-+:' {rn {nml {nkn l") (1 . ":336/,:r<*-):
{rn nno noo rmf o rn o h o o n V V l") (1 . ":336/,:r<*-): {rn nno nhj rmf o rn o h o o n V V l") (1 . "/061+
{rn nniqhghilomhognglj nhj rloqhnnninfhjjhfgh l") (1 . "/061+ {rn nniqhghilomhognglj noo
rloqhnnninfhjjhfgh l") (1 . "<0:;8: {rn {gl {nnn {gf {nml n {li {rn l") (1 . ",+->687+r<*-): {rn
ijqjklkhgmiogifjil njo riqnklfkfomijkgflgl o rn o V V l") (1 . "/061+ {rn ijqjklkhgmiogifjil moo
riqnklfkfomijkgflgl l") (1 . ":;8: {rn {fm {nnj {gf {nmh o l") (1 . "<0:;8: {rn {fo {ff {fg {nmf n {fn {rn l")
(1 . "300/ {rn {rn {nkm {ho l") (1 . "/061+ {rn ijqjklkhgmiogifjil noo riqnklfkfomijkgflgl l") (1 .
":336/,:r<*-): {rn noo noo o o rn o lj o o n V V l") (1 . ":336/,:r<*-): {rn noo moo o o n o lj o o n V V l")
(1 . ":;8: {rn {nol {non {nmk {nkl o l") (1 . "/061+ {rn nnlqjjjnlohhiioohf noo lmqmigjlimnhiknilf l") (1 .
"/061+ {rn gfqnhimlglhknhfohf noo llqmgklmlkonohjggn l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {noh {l
{rn l") (1 . "300/ {rn {gi {nlh {noh l") (1 . "/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V l") (1 .
"<0:;8: {nkk {nlj {nlj {nkm {nkj o {nof {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nno l") (1 . "<0:;8: {rn
{nlh {nlh {nno {nlg n {nll {rn l") (1 . ":;8: {rn {nki {nki {nlh {nkh o l") (1 . "/061+ {rn
nojqhofhnmfgnoknlf nhj rlkqjlnnllkjkgnllmf l") (1 . ",+->687+r<*-): {rn nojqhofhnmfgnoknlf noo
rlkqjlnnllkjkgnllmf o n o V V l") (1 . "/061+ {rn nojqhofhnmfgnoknlf noo rlkqjlnnllkjkgnllmf l") (1 .
"<0:;8: {rn {nkm {nkm {nlj {nkj n {nmj {rn l") (1 . ":336/,:r<*-): {rn non noo mk o rn o
njqollmfilhglhmfog o o n V V l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nlj l") (1 . ":;8: {rn {nkg {nkg {nkm
{nkf o l") (1 . "):-+:' {rn {nlg {njo l") (1 . ":336/,:r<*-): {rn nmm moo o o n o i o o n V V l") (1 . "):-+:' {rn
{nkj {njn l") (1 . ":336/,:r<*-): {rn nmm noo o o rn o i o o n V V l") (1 . "/061+ {rn nni moo
kqmilmjiknkjioionn:ronk l") (1 . "/061+ {rn nni noo rkqmilmjiknkjioionn:ronk l"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
```

```
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
```

```
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 80 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "A1") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi nnk n o        ") (1 . "=0;& {n {m {rn {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k |") (1 . "3*2/ {j {rn {i {o |") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k |") (1 . ",7:33 {h {rn {rn {g {m |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k |") (1  "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: |") (1 . "9><: {rn {nm {nl {i {rn {nk -:):-,:; ,6183: |") (1 . "300/ {rn {nj {ni
{g |") (1 . "<01:r,*-9><: {rn nno noo rmf o n o h o o n V V o n o V V V V |") (1 . "9><: {rn {nh {ng {i
{rn {nf -:):-,:; ,6183: |") (1 . "300/ {rn {rn {mo {f |") (1 . "/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V
V V |") (1 . "300/ {rn {rn {mn {g |") (1 . "<0:;8: {rn {ni {ni {mo {mm n {no {rn |") (1 . "9><: {rn {ml
{mk {i {rn {mj -:):-,:; ,6183: |") (1 . "300/ {rn {mi {mh {nm |") (1 . "<01:r,*-9><: {rn non noo mk o n o
njqollmfilhglhmfog o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mo {mo {ni {mm o {nl {rn |") (1 .
"<0:;8: {rn {mn {mn {mg {mf o {nj {rn |") (1 . ":;8: {rn {lo {lo {ni {ln o |") (1 . "9><: {lm {ll {lk {i {rn
{lj -:):-,:; ,6183: |") (1 . "300/ {rn {rn {li {nh |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n
o o o V V V V |") (1 . "300/ {rn {rn {lh {nm |") (1 . "<0:;8: {rn {mh {mh {li {lg n {ng {rn |") (1 . "<0:;8:
{rn {mg {mg {mn {mf n {lf {rn |") (1 . ":;8: {rn {ko {ko {mg {kn o |") (1 . "):-+:' {rn {mm {km |") (1 .
":336/,:r<*-): {rn nno nhj rmf o rn o h o o n V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ml {l {rn |") (1 .
"9><: {kl {kk {kj {i {rn {ki -:):-,:; ,6183: |") (1 . "300/ {rn {kh {kg {ml |") (1 . "<01:r,*-9><: {rn nmm
njo o o rn o i o o n V V o n o V V V V |") (1 . "<0:;8: {rn {li {li {mh {lg o {mk {rn |") (1 . "<0:;8: {rn {lh
{lh {kf {jo o {mi {rn |") (1 . ":;8: {rn {jn {jn {mh {jm o |") (1 . "300/ {rn {jl {mg {kk |") (1 . "):-+:' {rn
{mf {jk |") (1 . ":336/,:r<*-): {rn nno noo rmf o rn o h o o n V V |") (1 . "/061+ {rn nnh nhj rmf |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {ll {l {rn |") (1 . "9><: {jj {ji {lf {i {rn {jh 90-(>-; ,6183: |") (1 . "300/ {rn
{jg {jf {ll |") (1 . "<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V |") (1 . "300/ {rn {rn {io
{ml |") (1 . "<0:;8: {in {kg {kg {im {il o {lk {rn |") (1 . "<0:;8: {rn {kf {kf {lh {jo n {jl {rn |") (1 . ":;8:
{rn {ik {ik {kf {ij o |") (1 . "):-+:' {rn {lg {ii |") (1 . ":336/,:r<*-): {rn non njo mk o rn o
njqollmfilhglhmfog o o n V V |") (1 . "300/ {rn {ih {kf {kk |") (1 . "/061+ {rn nnh noo rmf |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {kk {l {rn |") (1 . "9><: {ig {if {ho {i {rn {hn 90-(>-; ,6183: |") (1 .
"/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V |") (1 . "300/ {rn {rn {hm {ll |") (1 . "<0:;8: {hl {jf
{jf {hk {hj o {kj {rn |") (1 . "<0:;8: {hi {io {io {hh {hg o {kh {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn
{kg |") (1 . "<0:;8: {rn {im {im {kg {il n {ho {rn |") (1 . ":;8: {rn {hf {hf {im {go o |") (1 . "):-+:' {rn {jo
{gn |") (1 . ":336/,:r<*-): {rn non noo mk o rn o njqollmfilhglhmfog o o n V V |") (1 . "/061+ {rn
nniqollmfilhglhmfn njo mk |") (1 . "300/ {rn {gm {hh {kk |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ji {l
{rn |") (1 . "9><: {rn {rn {gl {i {rn {gk 90-(>-; ,6183: |") (1 . "300/ {rn {gj {im {ji |") (1 . "/3>1:r,*-9><:
{rn noo moo o o n o n o o o V V V V |") (1 . "<0:;8: {gi {hm {hm {gh {gg o {jg {rn |") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {jf |") (1 . "<0:;8: {rn {hk {hk {jf {hj n {gj {rn |") (1 . ":;8: {rn {gf {gf {hk {fo o
|") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {io |") (1 . "<0:;8: {rn {hh {hh {io {hg n {ih {rn |") (1 . ":;8: {rn {fn
{fn {hh {fm o |") (1 . "):-+:' {rn {il {fl |") (1 . ":336/,:r<*-): {rn nmm moo o o n o i o o n V V |") (1 .
"/061+ {rn nniqollmfilhglhmfn noo mk |") (1 . "300/ {rn {fk {gh {kk |") (1 . "300/ {rn {fj {fi {if |") (1 .
"<01:r,*-9><: {rn noo njo o o n o ko o o n V V o n o V V V V |") (1 . "300/ {rn {fh {hk {ji |") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {hm |") (1 . "<0:;8: {rn {gh {gh {hm {gg n {gm {rn |") (1 . ":;8: {rn {fg {fg {gh
{ff o |") (1 . "):-+:' {rn {hj {noo |") (1 . ":336/,:r<*-): {rn hh moo o o n o rnl o o n V V |") (1 . "):-+:' {rn
{hg {non |") (1 . ":336/,:r<*-): {rn nmm noo o o rn o i o o n V V |") (1 . "/061+ {rn nni moo
kqmilmjiknkjioionn:ronk |") (1 . "300/ {rn {rn {nom {kk |") (1 . "300/ {rn {rn {nol {if |") (1 . "<0:;8: {rn
{fi {fi {nom {nok n {gl {rn |") (1 . "300/ {rn {rn {noj {ji |") (1 . "):-+:' {rn {gg {noi |") (1 . ":336/,:r<*-):
{rn hh noo o o rn o rnl o o n V V |") (1 . "/061+ {rn fo moo rgqjmijnmgmfnmnmomm:ronk |") (1 . "/061+
{rn nni noo rkqmilmjiknkjioionn:ronk |") (1 . "<0:;8: {rn {nom {nom {fi {nok o {fk {rn |") (1 . "<0:;8: {rn
{nol {nol {noj {noh n {fj {rn |") (1 . ":;8: {rn {nog {nog {nom {nof o |") (1 . "<0:;8: {rn {noj {noj {nol
{noh o {fh {rn |") (1 . "/061+ {rn fo noo gqjmijnmgmfnmnmomm:ronk |") (1 . ":;8: {rn {nno {nno {noj
```

```
{nnn o l") (1 . "):-+:' {rn {nok {nnm l") (1 . ":336/,:r<*-): {rn noo noo o o rn o ko o o n V V l") (1 . "):-+:'
{rn {noh {nnl l") (1 . ":336/,:r<*-): {rn noo moo o o n o ko o o n V V l") (1 . "/061+ {rn io noo o l") (1 .
"/061+ {rn nko moo o l"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
```

```
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the 90 mm diameter unfinished model.
(entmake
'((0 . "3DSOLID") (5 . "A3") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi nmi n o        ") (1 . "=0;& {n {m {rn {rn |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {o {l {k |") (1 . "3*2/ {j {rn {i {o |") (1 . "-:961:2:1+ {rn o o o o
oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn
{m {l {k |") (1 . ",7:33 {h {rn {rn {g {m |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {i {l {k |") (1 . "9><: {rn
{f {no {i {rn {nn -:):-,:; ,6183: |") (1 . "9><: {rn {nm {nl {i {rn {nk -:):-,:; ,6183: |") (1 . "300/ {rn {nj {ni
{g |") (1 . "<01:r,*-9><: {rn nno noo rmf o n o h o o n V V o n o V V V V |") (1 . "9><: {rn {nh {ng {i
{rn {nf -:):-,:; ,6183: |") (1 . "300/ {rn {rn {mo {f |") (1 . "/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V
V V |") (1 . "300/ {rn {rn {mn {g |") (1 . "<0:;8: {rn {ni {ni {mo {mm n {no {rn |") (1 . "9><: {rn {ml
{mk {i {rn {mj -:):-,:; ,6183: |") (1 . "300/ {rn {mi {mh {nm |") (1 . "<01:r,*-9><: {rn non noo mk o n o
njqollmfilhglhmfog o o n V V o n o V V V V |") (1 . "<0:;8: {rn {mo {mo {ni {mm o {nl {rn |") (1 .
"<0:;8: {rn {mn {mn {mg {mf o {nj {rn |") (1 . ":;8: {rn {lo {lo {ni {ln o |") (1 . "9><: {lm {ll {lk {i {rn
{lj -:):-,:; ,6183: |") (1 . "300/ {rn {rn {li {nh |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n
o o o V V V V |") (1 . "300/ {rn {rn {lh {nm |") (1 . "<0:;8: {rn {mh {mh {li {lg n {ng {rn |") (1 . "<0:;8:
{rn {mg {mg {mn {mf n {lf {rn |") (1 . ":;8: {rn {ko {ko {mg {kn o |") (1 . "):-+:' {rn {mm {km |") (1 .
":336/,:r<*-): {rn nno nhj rmf o rn o h o o n V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {ml {l {rn |") (1 .
"9><: {kl {kk {kj {i {rn {ki -:):-,:; ,6183: |") (1 . "300/ {rn {kh {kg {ml |") (1 . "<01:r,*-9><: {rn nmm
njo o o rn o i o o n V V o n o V V V V |") (1 . "<0:;8: {rn {li {li {mh {lg o {mk {rn |") (1 . "<0:;8: {rn {lh
{lh {kf {jo o {mi {rn |") (1 . ":;8: {rn {jn {jn {mh {jm o |") (1 . "300/ {rn {rn {mg {jl |") (1 . "):-+:' {rn
{mf {jk |") (1 . ":336/,:r<*-): {rn nno noo rmf o rn o h o o n V V |") (1 . "/061+ {rn nnh nhj rmf |") (1 . "-
:9@)+r3(;r>++-6= {rn {rn {rn {ll {l {rn |") (1 . "9><: {jj {ji {jh {i {rn {jg 90-(>-; ,6183: |") (1 . "300/ {rn
{jf {io {ll |") (1 . "<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V |") (1 . "300/ {rn {rn {in
{ml |") (1 . "<0:;8: {im {kg {kg {il {ik o {lk {rn |") (1 . "<0:;8: {rn {kf {kf {lh {jo n {ij {rn |") (1 . ":;8:
{rn {ii {ii {kf {ih o |") (1 . "):-+:' {rn {lg {ig |") (1 . ":336/,:r<*-): {rn non njo mk o rn o
njqollmfilhglhmfog o o n V V |") (1 . "9><: {if {rn {ho {i {rn {hn 90-(>-; ,6183: |") (1 . "/061+ {rn nnh
noo rmf |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {kk {l {rn |") (1 . "9><: {hm {hl {hk {i {rn {hj 90-(>-;
,6183: |") (1 . "300/ {rn {hi {hh {kk |") (1 . "<01:r,*-9><: {rn noo njo o o rn o rkj o o n V V
roqoffjolhnfomoffgfnj oqffjolhnfomoffgfnj o V V V V |") (1 . "300/ {rn {rn {hg {ll |") (1 . "<0:;8: {hf {io
{io {go {gn o {kj {rn |") (1 . "<0:;8: {gm {in {in {gl {gk o {kh {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn
{kg |") (1 . "<0:;8: {rn {il {il {kg {ik n {hk {rn |") (1 . ":;8: {rn {gj {gj {il {gi o |") (1 . "300/ {rn {lf {kf
{jl |") (1 . "):-+:' {rn {jo {gh |") (1 . ":336/,:r<*-): {rn non noo mk o rn o njqollmfilhglhmfog o o n V V |")
(1 . "/061+ {rn nniqollmfilhglhmfn njo mk |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {jl {l {rn |") (1 . "300/
{rn {gg {gf {jl |") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V V |") (1 . "-:9@)+r3(;r>++-6=
{rn {rn {rn {ji {l {rn |") (1 . "9><: {rn {jl {fo {i {rn {fn 90-(>-; ,6183: |") (1 . "300/ {rn {fm {il {ji |") (1 .
"/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V |") (1 . "300/ {rn {rn {fl {kk |") (1 . "<0:;8: {rn {hh
{hh {fk {fj o {jh {rn |") (1 . "<0:;8: {fi {hg {hg {fh {fg o {jf {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {io
|") (1 . "<0:;8: {rn {go {go {io {gn n {fm {rn |") (1 . ":;8: {rn {ff {ff {go {noo o |") (1 . "<0/>-r3(;r>++-6=
{rn {rn {rn {in |") (1 . "<0:;8: {rn {gl {gl {in {gk n {non {rn |") (1 . ":;8: {rn {nom {nom {gl {nol o |") (1 .
"):-+:' {rn {ik {nok |") (1 . ":336/,:r<*-): {rn nmm moo o o n o i o o n V V |") (1 . "/061+ {rn
nniqollmfilhglhmfn noo mk |") (1 . "300/ {rn {non {fh {jl |") (1 . "<0:;8: {rn {gf {gf {fl {noj n {ho {rn |")
```

```
(1 . "300/ {rn {noi {noh {hl |") (1 . "<01:r,*-9><: {rn noo njo o o n o kj o o n V V o n o V V V V |") (1 .
"300/ {rn {nog {go {ji |") (1 . "<0:;8: {nof {fl {fl {gf {noj o {hi {rn |") (1 . "<0:;8: {rn {fk {fk {hh {fj n
{noi {rn |") (1 . ":;8: {rn {nno {nno {fk {nnn o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {hg |") (1 . "<0:;8:
{rn {fh {fh {hg {fg n {gg {rn |") (1 . ":;8: {rn {nnm {nnm {fh {nnl o |") (1 . "):-+:' {rn {gn {nnk |") (1 .
":336/,:r<*-): {rn hh moo o o n o rnl o o n V V |") (1 . "300/ {rn {ij {gl {jl |") (1 . "):-+:' {rn {gk {nnj |")
(1 . ":336/,:r<*-): {rn nmm noo o o rn o i o o n V V |") (1 . "/061+ {rn nni moo kqmilmjiknkjioionn:ronk
|") (1 . ":;8: {rn {nni {nni {gf {nnh o |") (1 . "300/ {rn {rn {fk {hl |") (1 . "<0:;8: {rn {noh {noh {nng {nnf
n {fo {rn |") (1 . "300/ {rn {rn {nng {ji |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {fl |") (1 . "):-+:' {rn {fj
{nmo |") (1 . ":336/,:r<*-): {rn noo njo o o rn o kj o o n V V |") (1 . "):-+:' {rn {fg {nmn |") (1 .
":336/,:r<*-): {rn hh noo o o rn o rnl o o n V V |") (1 . "/061+ {rn fo moo rgqjmijnmgmfnmnmomm:ronk
|") (1 . "/061+ {rn nni noo rkqmilmjiknkjioionn:ronk |") (1 . "):-+:' {rn {noj {nmm |") (1 . ":336/,:r<*-):
{rn noo noo o o n o rko o o n V V |") (1 . "<0:;8: {rn {nng {nng {noh {nnf o {nog {rn |") (1 . ":;8: {rn
{nml {nml {nng {nmk o |") (1 . "/061+ {rn jj njo lqnmilggolhlkkkkog:ronl |") (1 . "/061+ {rn fo noo
gqjmijnmgmfnmnmomm:ronk |") (1 . "/061+ {rn io noo o |") (1 . "):-+:' {rn {nnf {nmj |") (1 . ":336/,:r<*-
): {rn noo moo o o n o kj o o n V V |") (1 . "/061+ {rn nkj moo o |"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
```

```
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Creating the complete model.
(entmake
'((0 . "3DSOLID") (5 . "A5") (100 . "AcDbEntity") (67 . 0) (8 . "CENTRE") (100 .
"AcDbModelerGeometry") (70 . 1) (1 . "noi njj n o        ") (1 . "=0;& {n {m {rn {rn |") (1 .
"9@=0;&r3(;r>++-6= {rn {l {rn {o |") (1 . "3*2/ {k {rn {j {o |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {n {o {i
{h |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {rn {m {i {h |") (1 . ",7:33 {g {rn {rn {f {m |") (1 . "-:961:2:1+ {rn
o o o o oqknfffffgiggifhgnj o o o m |") (1 . "):-+:'@+:2/3>+: {rn l o n g |") (1 . "-:9@)+r3(;r>++-6= {rn
{rn {rn {j {i {h |") (1 . "9><: {no {nn {nm {j {rn {nl -:):-,:; ,6183: |") (1 . "99><:r3(;r>++-6= {rn {nk {rn
{f |") (1 . "9><: {nj {ni {nh {j {rn {ng -:):-,:; ,6183: |") (1 . "300/ {rn {nf {mo {f |") (1 . "<01:r,*-9><: {rn
nno noo rmf o n o h o o n V V o n o V V V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {no {f {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {mn {rn {nn |") (1 . "9><: {mm {ml {mk {j {rn {mj -:):-,:; ,6183: |") (1 . "300/ {rn
{rn {mi {nn |") (1 . "/3>1:r,*-9><: {rn nnh nhj rmf o n o n o o o V V V V |") (1 . "300/ {rn {rn {mh {f |")
(1 . "<0:;8: {mg {mo {mo {mi {mf n {nm {rn |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {nj {nn {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {lo {rn {ni |") (1 . "9><: {ln {lm {ll {j {rn {lk -:):-,:; ,6183: |") (1 . "300/ {rn {lj
{li {ni |") (1 . "<01:r,*-9><: {rn non noo mk o n o njqollmfilhglhmfog o o n V V o n o V V V V |") (1 .
"<0:;8: {lh {mi {mi {mo {mf o {nh {rn |") (1 . "<0:;8: {lg {mh {mh {lf {ko o {nf {rn |") (1 . "<0/>-
r3(;r>++-6= {rn {rn {rn {mo |") (1 . ":;8: {kn {km {km {mo {kl o |") (1 . "-:9@)+r3(;r>++-6= {rn {rn
{mm {ni {i {rn |") (1 . "99><:r3(;r>++-6= {rn {kk {rn {ml |") (1 . "9><: {kj {ki {kh {j {rn {kg -:):-,:;
,6183: |") (1 . "300/ {rn {rn {kf {ml |") (1 . "/3>1:r,*-9><: {rn nniqollmfilhglhmfn njo mk o n o n o o o V
V V V |") (1 . "300/ {rn {rn {jo {ni |") (1 . "<0:;8: {jn {li {li {kf {jm n {mk {rn |") (1 . "<0/>-r3(;r>++-6=
{rn {rn {rn {mi |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {mh |") (1 . "<0:;8: {jl {lf {lf {mh {ko n {jk {rn |")
(1 . ":;8: {jj {ji {ji {lf {jh o |") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {mf |") (1 . "):-+:' {rn {mf {jg |") (1 .
":336/,:r<*-): {rn nno nhj rmf o rn o h o o n V V |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {ln {ml {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {jf {rn {lm |") (1 . "9><: {io {in {im {j {rn {il -:):-,:; ,6183: |") (1 . "300/ {rn {ik
{ij {lm |") (1 . "<01:r,*-9><: {rn nmm njo o o rn o i o o n V V o n o V V V V |") (1 . "<0:;8: {ii {kf {kf
{li {jm o {ll {rn |") (1 . "<0:;8: {ih {jo {jo {ig {if o {lj {rn |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {li |") (1 .
":;8: {ho {hn {hn {li {hm o |") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {lf |") (1 . "300/ {rn {rn {lf {hl |") (1 .
":/>-r3(;r>++-6= {rn {rn {rn {ko |") (1 . "):-+:' {rn {ko {hk |") (1 . ":336/,:r<*-): {rn nno noo rmf o rn o h
o o n V V |") (1 . "/061+ {rn nnh nhj rmf |") (1 . "-:9@)+r3(;r>++-6= {rn {rn {kj {lm {i {rn |") (1 .
"99><:r3(;r>++-6= {rn {hj {rn {ki |") (1 . "9><: {hi {hh {hg {j {rn {hf 90-(>-; ,6183: |") (1 . "300/ {rn
{go {gn {ki l") (1 . "<01:r,*-9><: {rn hh njo o o n o rnl o o n V V o n o V V V V l") (1 . "300/ {rn {rn
{gm {lm l") (1 . "<0:;8: {gl {ij {ij {gk {gj o {kh {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {kf l") (1 .
"<0/>-r3(;r>++-6= {rn {rn {rn {jo l") (1 . "<0:;8: {gi {ig {ig {jo {if n {gh {rn l") (1 . ":;8: {gg {gf {gf {ig
{fo o l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {jm l") (1 . "):-+:' {rn {jm {fn l") (1 . ":336/,:r<*-): {rn non njo
mk o rn o njqollmfilhglhmfog o o n V V l") (1 . "9><: {fm {rn {fl {j {rn {fk 90-(>-; ,6183: l") (1 . "/061+
{rn nnh noo rmf l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {io {ki {i {rn l") (1 . "99><:r3(;r>++-6= {rn {fj {rn
{in l") (1 . "9><: {fi {hl {fh {j {rn {fg 90-(>-; ,6183: l") (1 . "300/ {rn {ff {noo {in l") (1 . "<01:r,*-9><:
{rn noo njo o o rn o rkj o o n V V roqoffjolhnfomoffgfnj oqffjolhnfomoffgfnj o V V V V l") (1 . "300/ {rn
{rn {non {ki l") (1 . "<0:;8: {nom {gn {gn {nol {nok o {im {rn l") (1 . "<0:;8: {noj {gm {gm {noi {noh o
{ik {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {ij l") (1 . "<0:;8: {nog {gk {gk {ij {gj n {fh {rn l") (1 . ":;8:
{nof {nno {nno {gk {nnn o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {ig l") (1 . "300/ {rn {jk {ig {hl l") (1 .
":/>-r3(;r>++-6= {rn {rn {rn {if l") (1 . "):-+:' {rn {if {nnm l") (1 . ":336/,:r<*-): {rn non noo mk o rn o
njqollmfilhglhmfog o o n V V l") (1 . "/061+ {rn nniqollmfilhglhmfn njo mk l") (1 . "99><:r3(;r>++-6=
{rn {nnl {rn {hl l") (1 . "300/ {rn {nnk {nnj {hl l") (1 . "/3>1:r,*-9><: {rn noo noo o o rn o rn o o o V V V
V l") (1 . "-:9@)+r3(;r>++-6= {rn {rn {hi {in {i {rn l") (1 . "99><:r3(;r>++-6= {rn {nni {rn {hh l") (1 .
"300/ {rn {nnh {gk {hh l") (1 . "/3>1:r,*-9><: {rn noo moo o o n o n o o o V V V V l") (1 . "300/ {rn {rn
{nng {in l") (1 . "<0:;8: {nnf {noo {noo {nnj {nmo o {hg {rn l") (1 . "<0:;8: {nmn {non {non {nmm {nml
o {go {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {gn l") (1 . "<0:;8: {nmk {nol {nol {gn {nok n {nnh {rn
l") (1 . ":;8: {nmj {nmi {nmi {nol {nmh o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {gm l") (1 . "<0:;8: {nmg
{noi {noi {gm {noh n {nmf {rn l") (1 . ":;8: {nlo {nln {nln {noi {nlm o l") (1 . "<0/>-r3(;r>++-6= {rn {rn
{rn {gk l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {gj l") (1 . "):-+:' {rn {gj {nll l") (1 . ":336/,:r<*-): {rn nmm
moo o o n o i o o n V V l") (1 . "/061+ {rn nniqollmfilhglhmfn noo mk l") (1 . "-:9@)+r3(;r>++-6= {rn
{rn {fm {hl {i {rn l") (1 . "300/ {rn {nmf {nmm {hl l") (1 . "<0:;8: {nlk {nnj {nnj {noo {nmo n {fl {rn l")
(1 . "-:9@)+r3(;r>++-6= {rn {rn {fi {hh {i {rn l") (1 . "300/ {rn {nlj {nol {hh l") (1 . "<0:;8: {nli {nng
{nng {nlh {nlg n {ff {rn l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {noo l") (1 . ":;8: {nlf {nko {nko {nnj {nkn
o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {non l") (1 . "<0:;8: {nkm {nmm {nmm {non {nml n {nnk {rn l")
(1 . ":;8: {nkl {nkk {nkk {nmm {nkj o l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nol l") (1 . ":/>-r3(;r>++-6=
{rn {rn {rn {nok l") (1 . "):-+:' {rn {nok {nki l") (1 . ":336/,:r<*-): {rn hh moo o o n o rnl o o n V V l") (1 .
"<0/>-r3(;r>++-6= {rn {rn {rn {noi l") (1 . "300/ {rn {gh {noi {hl l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn
{noh l") (1 . "):-+:' {rn {noh {nkh l") (1 . ":336/,:r<*-): {rn nmm noo o o rn o i o o n V V l") (1 . "/061+
{rn nni moo kqmilmjiknkjioionn:ronk l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nnj l") (1 . "300/ {rn {rn
{nlh {hh l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nng l") (1 . "<0:;8: {nkg {nlh {nlh {nng {nlg o {nlj {rn
l") (1 . ":;8: {nkf {njo {njo {nlh {njn o l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {nmo l") (1 . "):-+:' {rn {nmo
{njm l") (1 . ":336/,:r<*-): {rn noo noo o o n o rko o o n V V l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nmm
l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn {nml l") (1 . "):-+:' {rn {nml {njl l") (1 . ":336/,:r<*-): {rn hh noo o o
rn o rnl o o n V V l") (1 . "/061+ {rn fo moo rgqjmijnmgmfnmnmomm:ronk l") (1 . "/061+ {rn nni noo
rkqmilmjiknkjioionn:ronk l") (1 . "<0/>-r3(;r>++-6= {rn {rn {rn {nlh l") (1 . ":/>-r3(;r>++-6= {rn {rn {rn
{nlg l") (1 . "):-+:' {rn {nlg {njk l") (1 . ":336/,:r<*-): {rn noo moo o o n o jo o o n V V l") (1 . "/061+ {rn
io noo o l") (1 . "/061+ {rn fo noo gqjmijnmgmfnmnmomm:ronk l") (1 . "/061+ {rn jo moo o l"))
)

; Pause for about 3.5 seconds
(setq counter 1.0)
(while (<= counter 500000)
  (setq counter (1+ counter))
)
(princ)

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)
```

```
; Change to South-West isometric view.
(command "vpoint" "-1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to South-East isometric view.
(command "vpoint" "1,-1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-East isometric view.
(command "vpoint" "1,1,1")

; Render the objects
(command "_avrender")

; Pause for about 2 seconds
(setq counter 1.0)
(while (<= counter 300000)
  (setq counter (1+ counter))
)
(princ)

; Change to North-West isometric view.
(command "vpoint" "-1,1,1")

; Render the objects
(command "_avrender")
```

Appendix D Extract.c Program Source Codes

```
/* Extract.c is a program that extracts 3D surface points from a dxf file. The extracted points are then
stored inside another text file. All the points in the destination file are consist of X, Y and Z coordinates.
*/

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>

#define OUTPUT_FILE "c:\\temp\\extract.txt"
#define READ "r"
#define WRITE "w"
#define STRING_LENGTH 81
#define STRIKE_2D "AcDb2dVertex"
#define STRIKE_3D "AcDb3dPolylineVertex"
#define TEN "10"
#define TWENTY "20"
#define THIRTY "30"

main()
{
        FILE *data;
        char filename[STRING_LENGTH];
        FILE *new_data;
        char     axis[STRING_LENGTH];
        float x_coor, y_coor, z_coor;
        char string[STRING_LENGTH];
        int  counter;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        /* attempt to open files */
        data = fopen(filename, READ);
        new_data = fopen(OUTPUT_FILE, WRITE);

        if (data == NULL || new_data == NULL)
        {
                printf("file cannot be opened\n");
                exit(errno);
        }

        counter = 0;

        while (! feof(data))
        {
                fscanf(data, "%s", string);

                if ((strcmp(string, STRIKE_2D)) == 0 || (strcmp(string, STRIKE_3D)) == 0)
                {
                        fscanf(data, "%s", axis);
```

```
                    if (counter != 0)
                    {
                              fprintf(new_data, "\n");
                    }
                    else
                    {
                              counter ++;
                    }

                    if ((strcmp(axis, TEN)) == 0)
                    {
                              fscanf(data, "%f", &x_coor);
                    }

                    fscanf(data, "%s", axis);

                    if ((strcmp(axis, TWENTY)) == 0)
                    {
                              fscanf(data, "%f", &y_coor);
                    }

                    fscanf(data, "%s", axis);

                    if ((strcmp(axis, THIRTY)) == 0)
                    {
                              fscanf(data, "%f", &z_coor);
                    }

                    /* the x, y & z value will have 6 decimal point */
                    fprintf(new_data, "%10.6f %10.6f %10.6f", x_coor, y_coor, z_coor);
          }
     }

     fclose(data);
     fclose(new_data);

     printf("The 3D data file is in c:\\temp\\extract.txt");

     return 0;
}
```

Appendix E Convert.c Program Source Codes

```
/* Convert.c is a program that extracts points from an external text file. The extracted points are then
converted from x_coor, y_coor and z_coor into x mm, r mm and Rx degree (from Cartesian co-ordinate
system to Cylindrical co-ordinate system). Assume that the XY plane reference point is (100,100). Z
value will be converted into float number with 1 decimal places. */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<math.h>

#define OUTPUT_FILE "c:\\temp\\convert.txt"
#define READ "r"
#define WRITE "w"
#define STRING_LENGTH 81
#define PI 3.141593
#define XREF 100
#define YREF 100

main()
{
	FILE	*xyz_data;
	char	filename[STRING_LENGTH];
	FILE	*xrRx_data;
	float	x_coor, y_coor, z_coor;
	float	x, r, Rx;
	int	counter;

	printf("Please specify the file with correct path such as\n");
	printf("c:\\folder1\\folder2\\filename.txt\n\n");
	gets(filename);

	/* attempt to open files */
	xyz_data = fopen(filename, READ);
	xrRx_data = fopen(OUTPUT_FILE, WRITE);

	if (xyz_data == NULL || xrRx_data == NULL)
	{
		printf("file cannot be opened\n");
		exit(errno);
	}

	counter = 0;

	while (! feof(xyz_data))
	{
		fscanf(xyz_data, "%f", &x_coor);
		fscanf(xyz_data, "%f", &y_coor);
		fscanf(xyz_data, "%f", &z_coor);

		if (counter != 0)
		{
			fprintf(xrRx_data, "\n");
}
else
{
        counter ++;
}

x = z_coor;

if (x_coor==XREF || y_coor==YREF)
{
        if (y_coor==YREF && x_coor>XREF)
        {
                Rx = 0;
        }

        if (x_coor==XREF && y_coor>YREF)
        {
                Rx = 90;
        }

        if (y_coor==YREF && x_coor<XREF)
        {
                Rx = 180;
        }

        if (x_coor==XREF && y_coor<YREF)
        {
                Rx = 270;
        }
}
else
{
        Rx = ((atan((y_coor-YREF)/(x_coor-XREF)))/PI)*180;

        if (x_coor>XREF && y_coor>YREF)
        {
                Rx = Rx;
        }

        if (x_coor<XREF && y_coor>YREF)
        {
                Rx = 180 + Rx;
        }

        if (x_coor<XREF && y_coor<YREF)
        {
                Rx = 180 + Rx;
        }

        if (x_coor>XREF && y_coor<YREF)
        {
                Rx = 360 + Rx;
        }
}

r = sqrt(((y_coor-YREF)*(y_coor-YREF)) + ((x_coor-XREF)*(x_coor-XREF)));

/* x will have 1 decimal place */
```

```
            fprintf(xrRx_data, "%5.1f %10.6f %10.6f", x, Rx, r);
    }

    fclose(xyz_data);
    fclose(xrRx_data);

    printf("The 3D data file is in c:\\temp\\convert.txt");

    return 0;
}
```

Appendix F Sortadd.c Program Source Codes

```
/* Sortadd.c is a program that rearrange the x mm, Rx degree and r mm so that the Rx will always start at
the lowest angle in each layer(cross sectional profile) of the 3D model. 1 extra point is added at the
beginning and at the end of of each layer so that the 1st point starts at Rx = 0 degree and the last
point ends at Rx = 360 degree.
Assumption - maximum points in each layer are 4000
            - data will be sorted layer by layer
The data will be arranged so that the motion will be continuous from the 1st point to the last point.
The r mm at 0 and 360 degree will be calculated (linear interpolation) based on the 1st and last r mm of
the sorted layer.                                                                                    */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define OUTPUT_FILE "c:\\temp\\sorted.txt"
#define READ "r"
#define APPEND "a"
#define STRING_LENGTH 81
#define FIRST_POINT 0.000000 /* 0 degree */
#define LAST_POINT 360.000000 /* 360 degree */

void insertion_sort(int size);

float           x[4000];
float           Rx[4000];
float           r[4000];

main()
{
        FILE    *xRxr_data;
        char    filename[STRING_LENGTH];
        int     index;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        /* attempt to open files */
        xRxr_data = fopen(filename, READ);

        if (xRxr_data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }

        index = 0;

        while (! feof(xRxr_data))
        {
                fscanf(xRxr_data, "%f", &x[index]);
                fscanf(xRxr_data, "%f", &Rx[index]);
                fscanf(xRxr_data, "%f", &r[index]);

                if (x[0] != x[index]) /* compare subsequent x with the 1st x of the same layer */
                {
                        /* when the subsequent x is not equal to the 1st x then, start
                           sorting */

                        insertion_sort(index);

                        /* assign the array's last element value to the subsequent layer's
                           array's 1st element value */

                        x[0]  = x[index];
                        Rx[0] = Rx[index];
                        r[0]  = r[index];

                        index = 0;
                }
                else
                        if (feof(xRxr_data) && x[0] == x[index]) /* if it is end of file */
                {                                  /* and the 1st x same as */
                        insertion_sort(index);             /* the last x */
                }

                index ++;
        }

        fclose(xRxr_data);

        printf("The sorted file is in c:\\temp\\sorted.txt.\n");

        return 0;
}

/* Sorting the x, Rx and r into ascending order(from less to more) and save the sorted data into another
external file. */
void insertion_sort(int size)
{
        FILE    *sorted_data;
        int     counter, location, idx;
        float current_x, current_Rx, current_r;
        float alpha, beta, distance, angle, dist_per_deg, z, incr_decr_dist;

        sorted_data = fopen(OUTPUT_FILE, APPEND);

        if (sorted_data == NULL)
        {
                printf("sorted_data.txt cannot be opened\n");
                exit(errno);
        }

        if (x[0] != x[size])
        {
                /* sorting procedures for the 1st layer until the 2nd last layer */

                for (counter = 1; counter <= size - 1; counter ++)
                {
                        current_x  = x[counter];
```

```
                    current_Rx = Rx[counter];
                    current_r  = r[counter];

                    location   = counter;

                    while (location > 0 && Rx[location - 1] > current_Rx)
                    {
                              x[location]  = x[location - 1];
                              Rx[location] = Rx[location - 1];
                              r[location]  = r[location - 1];

                              location --;
                    }

                    x[location]  = current_x;
                    Rx[location] = current_Rx;
                    r[location]  = current_r;
          }
}
else
{
          /* sorting procedures for the last layer */

          for (counter = 1; counter <= size; counter ++)
          {
                    current_x  = x[counter];
                    current_Rx = Rx[counter];
                    current_r  = r[counter];

                    location   = counter;

                    while (location > 0 && Rx[location - 1] > current_Rx)
                    {
                              x[location]  = x[location - 1];
                              Rx[location] = Rx[location - 1];
                              r[location]  = r[location - 1];

                              location --;
                    }

                    x[location]  = current_x;
                    Rx[location] = current_Rx;
                    r[location]  = current_r;
          }
}

if (x[0] != x[size])
{
          distance = r[0] - r[size - 1];
}
else
{
          distance = r[0] - r[size];
}

if (distance < 0)
{
          distance = (-1) * distance;
```

```
}

beta = Rx[0] - 0.000000;

if (x[0] != x[size])
{
        alpha = 360.000000 - Rx[size - 1];
}
else
{
        alpha = 360.000000 - Rx[size];
}

angle = alpha + beta;

if (x[0] != x[size])
{
        if ((angle == 0) && (Rx[0] == 0.000000))
        {
                z = r[0];
        }
        if ((angle == 0) && (Rx[size - 1] == 360.000000))
        {
                z = r[size - 1];
        }
}
else
{
        if ((angle == 0) && (Rx[0] == 0.000000))
        {
                z = r[0];
        }
        if ((angle == 0) && (Rx[size] == 360.000000))
        {
                z = r[size];
        }
}

dist_per_deg = distance / angle;

incr_decr_dist = dist_per_deg * beta;

if (x[0] != x[size])
{
        if (r[0] > r[size - 1])
        {
                z = r[size - 1] + incr_decr_dist;
        }
        if (r[size - 1] > r[0])
        {
                z = r[0] + incr_decr_dist;
        }
        if (r[size - 1] == r[0])
        {
                z = r[0];
        }
}
else
```

```
	{
		if (r[0] > r[size])
		{
			z = r[size] + incr_decr_dist;
		}
		if (r[size] > r[0])
		{
			z = r[0] + incr_decr_dist;
		}
		if (r[size] == r[0])
		{
			z = r[0];
		}
	}

	fprintf(sorted_data, "%10.6f %10.6f %10.6f\n", x[0], FIRST_POINT, z);

	if (x[0] != x[size])
	{
		for (idx = 0; idx <= size - 1; idx ++)
		{
			fprintf(sorted_data, "%10.6f %10.6f %10.6f\n", x[idx], Rx[idx], r[idx]);
		}
	}
	else
	{
		for (idx = 0; idx <= size; idx ++)
		{
			fprintf(sorted_data, "%10.6f %10.6f %10.6f\n", x[idx], Rx[idx], r[idx]);
		}
	}

	if (x[0] != x[size])
	{
		fprintf(sorted_data, "%10.6f %10.6f %10.6f\n", x[0], LAST_POINT, z);
	}
	else
	{
		fprintf(sorted_data, "%10.6f %10.6f %10.6f", x[0], LAST_POINT, z);
	}

	fclose(sorted_data);
}
```

Appendix G Vd.c Program Source Codes

```
/* Vd.c will convert axis 1's (x mm), axis 2's (r mm) and axis 3's (Rx degree) into their respective velocity
and incremental or decremental motor step. The new data will be stored into another external text file.
The program will ask the user to key in the file location, file name, desired feed rate and radius
of the raw material block. */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>

#define OUTPUT_FILE "c:\\temp\\vd.txt"
#define STRING_LENGTH 162
#define READ "r"
#define APPEND "a"

float axis_1_incremental_step(float x_mm);
float axis_3_incremental_step(float Rx_deg);
float axis_2_inde_step(float r_mm);

float    axis_1_step; /* axis 1's incremental steps to be moved */
float    axis_3_step; /* axis 3's incremental steps to be moved */
float    axis_2_step; /* axis 2's incremental or decremental steps to be moved */

float radius; /* radius of the raw material block */

main()
{
         float feed_rate, RX_V; /* RX_V is in rev/sec */
         float    axis_1_mm; /* absolute distance from reference point */
         float axis_3_deg; /* absolute angle of rotation */
         float axis_2_mm; /* 3D model thickness at certain angle */
         FILE     *data;
         char     filename[STRING_LENGTH];
         FILE     *vd;
         float    r_v, x_v;
         int  counter;
         float x_step, r_step, Rx_step;

         printf("Please specify the file with correct path such as\n");
         printf("c:\\folder1\\folder2\\filename.txt\n\n");
         gets(filename);

         printf("\nPlease specify the desired feed rate.\n");
         printf("The feed rate is ranging from 0.015 mm/sec to 0.75 mm/sec\n");
         scanf("%f", &feed_rate);

         RX_V = feed_rate / (0.0003 * 5000); /* 0.0003 step/mm, 5000 step/rev */

         printf("\nPlease specify the radius of the raw material block in mm.\n");
         scanf("%f", &radius);
         printf("\n");
```

```
data = fopen(filename, READ);

if (data == NULL)
{
          printf("%s cannot be opened\n", filename);
          exit(errno);
}

/* assumptions : axis 1 coordinate is on the left side of the table
                   axis 3 coordinate is on the middle of the table
                   axis 2 coordinate is on the right side of the table */

counter = 0;

while (! feof(data))
{
          fscanf(data, "%f", &axis_1_mm);
          axis_1_incremental_step(axis_1_mm);

          fscanf(data, "%f", &axis_3_deg);
          axis_3_incremental_step(axis_3_deg);

          fscanf(data, "%f", &axis_2_mm);
          axis_2_inde_step(axis_2_mm);

          x_step = axis_1_step;
          r_step = axis_2_step;
          Rx_step = axis_3_step;


          vd = fopen(OUTPUT_FILE, APPEND);

          if (vd == NULL)
          {
                    printf("%s cannot be opened\n", vd);
                    exit(errno);
          }

          if (Rx_step != 0)
          {
                    if (r_step != 0 && x_step != 0)
                    {
                              r_v = (r_step/Rx_step)*RX_V;

                              x_v = (x_step/Rx_step)*RX_V;
                    }

                    if (r_step != 0 && x_step == 0)
                    {
                              r_v = (r_step/Rx_step)*RX_V;

                              x_v = RX_V;
                    }

                    if (r_step == 0 && x_step != 0)
                    {
                              r_v = RX_V;
```

```
                x_v = (x_step/Rx_step)*RX_V;
        }

        if (r_step == 0 && x_step == 0)
        {
                r_v = RX_V;

                x_v = RX_V;
        }
}
else
        if (Rx_step == 0)
        {
                if (r_step != 0)
                {
                        if (x_step != 0)
                        {
                                x_v = RX_V;

                                r_v = (r_step/x_step)*x_v;
                        }
                        else
                                if (x_step == 0)
                                {
                                        x_v = RX_V;

                                        r_v = RX_V;
                                }
                }
                else
                        if (r_step == 0)
                        {
                                if (x_step != 0)
                                {
                                        x_v = RX_V;

                                        r_v = (r_step/x_step)*x_v;
                                }
                                else
                                        if (x_step == 0)
                                        {
                                                x_v = RX_V;

                                                r_v = RX_V;
                                        }
                        }
        }

if (x_step != 0 && r_step == 0 && Rx_step == 0)
{
        x_v = RX_V;
}

if (x_step == 0 && r_step != 0 && Rx_step != 0)
{
        x_v = RX_V;
}
```

```
		if (x_v < 0)
		{
			x_v = x_v*(-1);
		}

		if (x_v >= 5)
		{
			x_v = 5;
		}

		if (r_v < 0)
		{
			r_v = r_v*(-1);
		}

		if (r_v >= 5)
		{
			r_v = 5;
		}

		if (r_v > 0 && r_v < 0.0001)
		{
			r_v = 0.0001;
		}

		if (r_step == 0 && Rx_step == 0)
		{
			r_v = RX_V;
		}

		if (counter != 0)
		{
			fprintf(vd, "\n");
		}
		else
		{
			counter ++;
		}

		fprintf(vd, "%10.4f %10.0f %10.4f %10.0f %10.4f %10.0f", x_v, x_step, r_v, r_step,
RX_V, Rx_step);

		fclose(vd);
	}

	fclose(data);

	printf("The file is in c:\\temp\\vd.txt");

	return 0;
}

/* Convert axis 1's absolute distance to incremental motor step */
float axis_1_incremental_step(float x_mm)
{  /* assumption : axis 1's reference point is on the other end of the rotary shaft of axis 3
		: the 1st value for axis_1_mm is 0 */

	static float prev_x_mm = 0;
```

```
            axis_1_step = (x_mm - prev_x_mm)/(0.3E-03);

            prev_x_mm = x_mm;

            return (axis_1_step);
}

/* Convert axis 3's absolute angle to incremental motor step */
float axis_3_incremental_step(float Rx_deg)
{           /* assumption : axis 3's reference point(or axis) is the axis itself */

            static float prev_Rx_deg = 0;

            if (Rx_deg == 0)
            {
                        axis_3_step = 0;
            }
            else
            {
                        axis_3_step = (Rx_deg - prev_Rx_deg)/(0.72E-03);
            }

            prev_Rx_deg = Rx_deg;

            return (axis_3_step);
}

/* Calculating machining distance(incremental or decremental motor steps) */
float axis_2_inde_step(float r_mm)
{           /* assumptions : axis 2's reference point is on the axis 3
                                    axis_2_step is the incremental or decremental steps to be move by the motor */

            static float prev_r_mm = 0;
            static float counter = 0;

            if (counter == 0)
            {
                        prev_r_mm = radius;
            }

            axis_2_step = (prev_r_mm - r_mm)/(0.3E-03);

            prev_r_mm = r_mm;

            counter = counter ++;

            return (axis_2_step);
}
```

Appendix H VsvMt.c Program Source Codes

```
/* VsvMt.c send the commands of axis 1, 2 and 3 together in one time. The motions of all the axes are
synchronised. Its functions are to construct commands and send them out to the PC-23 indexer for
execution. It is calculating the total production time as well as each individual machining time. The
remaining time of the production is also calculated.*/

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<errno.h>
#include<string.h>

#define FAIL 0X20
#define BIT2MASK 0X04
#define READY 0X16
#define CB 0X60
#define IDB_M 0X10
#define CHAR_READY 0X70
#define ODB      0X8
#define ACK      0XE0
#define ALDONE 0X02
#define HALT (CB | BIT2MASK)
#define RESTART 0x40 /* byte to restart the pc23 */
#define BADADDR 0XFF

#define ADDRESS 0X300 /* PC23's address */
#define STCTRL_ADDR 0X301 /* Status and Control byte address */

#define STRING_LENGTH 162
#define READ "r"

#define CMD_TRSF_TIME 0.2 /* 0.2 sec for transferring 1 command line */

void initialise(void);

void single_machining_time(float x_step, float r_step, float Rx_step);
void send_axis_123_cmd(float x_v, float x_step, float r_v, float r_step, float Rx_v, float Rx_step);

void write_command(char *c);
void write_character(char gamma);

void read_answer(char *a);
char read_character(void);

float speed; /* steps per second */
float total_time; /* total seconds needed to produce a model */

main()
{
        FILE  *data;
        char  filename[STRING_LENGTH];
        char      *message;
        float step = 0;
        float counter, total_cmd_trsf_time, machining_time, total_hour;
        float axis_1_v, axis_1_step, axis_2_v, axis_2_step, axis_3_v, axis_3_step;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        data = fopen(filename, READ);

        if (data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }

/* The source codes below are for calculating total machining time */

        counter = 0;

        while (! feof(data))
        {
                fscanf(data, "%f", &axis_1_v);
                fscanf(data, "%f", &axis_1_step);
                fscanf(data, "%f", &axis_2_v);
                fscanf(data, "%f", &axis_2_step);
                fscanf(data, "%f", &axis_3_v);
                fscanf(data, "%f", &axis_3_step);

                if (counter == 0)
                {
                        step = axis_2_step;
                }
                else if (axis_1_step == 0)
                        {
                                step = step + axis_3_step;
                        }
                        else if (axis_3_step == 0)
                                {
                                        step = step + axis_1_step;
                                }

                counter = counter ++;
        }

        printf("\n\nTotal step = %10.0f\n", step);
        printf("Total command line = %10.0f\n", counter);

        total_cmd_trsf_time = counter * 0.2; /* 0.2 seconds per 1 command transfer */
        printf("Command transfer time = %10.0f seconds\n", total_cmd_trsf_time);

        speed = axis_3_v * 5000; /* 5000 steps/rev */ /* speed in steps/sec. */
        printf("Machining speed = %10.0f steps/revolution\n", speed);

        machining_time = step / speed;
        printf("Machining time = %10.0f seconds\n", machining_time);

        total_time = total_cmd_trsf_time + machining_time;
        printf("Total time to produce the model = %10.2f seconds\n", total_time);
        total_hour = total_time / 3600; /* 3600 sec = 1 hr */
        printf("Total time to produce the model = %10.4f hours\n", total_hour);

        fclose(data);

/* The above source codes are for calculating total machining time  */

        data = fopen(filename, READ);

        if (data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }

        initialise();

        message=" 1MR6 2MR6 3MR6 ";  /* matching the motor resolution of PC23 and KS drive */
        write_command(message);

        while (! feof(data))
        {
                fscanf(data, "%f", &axis_1_v);
                fscanf(data, "%f", &axis_1_step);
                fscanf(data, "%f", &axis_2_v);
                fscanf(data, "%f", &axis_2_step);
                fscanf(data, "%f", &axis_3_v);
                fscanf(data, "%f", &axis_3_step);

                single_machining_time(axis_1_step, axis_2_step, axis_3_step);
                send_axis_123_cmd(axis_1_v,     axis_1_step,     axis_2_v,     axis_2_step,axis_3_v,
axis_3_step);
        }

        fclose(data);

        return 0;
}

/* Reset the PC23 board. Assuring that the board is ready to accept commands from the user program. */
void initialise(void)
{
        unsigned char statbyte;

        outp(STCTRL_ADDR, HALT); /* initialise procedure */
        while (!((statbyte = inp(STCTRL_ADDR)) & FAIL) );
        if(statbyte == BADADDR)
        {
                printf ("\n\nInvalid address, check PC-23 dipswitches\n\n");
                exit(1);
        }
        outp(STCTRL_ADDR, RESTART);
        outp(STCTRL_ADDR, CB);
        while( ( (statbyte=inp(STCTRL_ADDR)) & READY) != READY );
}

/* Calculating individual machining/motion time */
```

```
void single_machining_time(float x_step, float r_step, float Rx_step)
{
        static float pre_remaining_time = 0;
        static float counter = 0;
              float step = 0;
              float machining_time, remaining_hour, remaining_time;

        if (r_step < 0)
        {
                r_step = r_step * (-1);
        }

        if (x_step == 0)
        {
                if (r_step != 0)
                {
                        if (Rx_step != 0)
                        {
                                step = Rx_step;
                        }
                        else
                        {
                                step = r_step;
                        }
                }
                else
                {
                        step = Rx_step;
                }
        }
        else
        {
                if (r_step != 0)
                {
                        if (Rx_step != 0)
                        {
                                step = Rx_step;
                        }
                        else
                        {
                                step = x_step;
                        }
                }
                else
                {
                        if (Rx_step != 0)
                        {
                                step = Rx_step;
                        }
                        else
                        {
                                step = x_step;
                        }
                }
        }

        machining_time = step / speed;
        printf("\n\nThe machining time for this segment is %10.2f seconds\n", machining_time);
        if (counter == 0)
        {
                pre_remaining_time = total_time;
        }

        remaining_time = (pre_remaining_time - machining_time) - CMD_TRSF_TIME;
        printf("The remaining time for producing the model is %10.2f seconds\n", remaining_time);

        remaining_hour = remaining_time / 3600; /* 3600 seconds per hour */
        printf("The remaining time for producing the model is %10.4f hours\n\n", remaining_hour);

        pre_remaining_time = remaining_time;
        counter = counter ++;
}

/* Constructing and sending command string of axis 1, 2 and 3 to PC23 and getting position response
string and display it on the screen. The axis 2 motor step is incremental or decremental */
void send_axis_123_cmd(float x_v, float x_step, float r_v, float r_step, float Rx_v, float Rx_step)
{
        int sig = 7;

        char axis_123_cmd[STRING_LENGTH];

        char *axis_1_vs = " 1VS";
        char axis_1_vs_str[81]; /* axis 1's initial velocity */
        char *axis_1_v = " 1V";
        char axis_1_v_str[81]; /* axis 1's velocity */
        char *axis_1_d = " 1D";
        char axis_1_step_str[81]; /* axis 1's integer motor step in char/string form */
        char *axis_1_i = " 1I";

        char *axis_2_vs = " 2VS";
        char axis_2_vs_str[81]; /* axis 2's initial velocity */
        char *axis_2_v = " 2V";
        char axis_2_v_str[81]; /* axis 2's velocity */
        char *axis_2_d = " 2D";
        char axis_2_step_str[81]; /* axis 2's integer motor step in char/string form */
        char *axis_2_i = " 2I";

        char *axis_3_vs = " 3VS";
        char axis_3_vs_str[81]; /* axis 3's initial velocity */
        char *axis_3_v = " 3V";
        char axis_3_v_str[81]; /* axis 3's velocity */
        char *axis_3_d = " 3D";
        char axis_3_step_str[81]; /* axis 3's integer motor step in char/string form */
        char *axis_3_i = " 3I";

        char *axis_123_end = " G123 ";

        char *message, *answer;
        answer = "";

/* constructing axis 1's motion commands */

        strcpy(axis_123_cmd, axis_1_vs);
        gcvt(x_v, sig, axis_1_vs_str);
        strcat(axis_123_cmd, axis_1_vs_str);
```

```
            strcat(axis_123_cmd, axis_1_v);
            gcvt(x_v, sig, axis_1_v_str);
            strcat(axis_123_cmd, axis_1_v_str);

            strcat(axis_123_cmd, axis_1_d);
            gcvt(x_step, sig, axis_1_step_str);
            strcat(axis_123_cmd, axis_1_step_str);

            strcat(axis_123_cmd, axis_1_i);

/* constructing axis 2's motion commands */

            strcat(axis_123_cmd, axis_2_vs);
            gcvt(r_v, sig, axis_2_vs_str);
            strcat(axis_123_cmd, axis_2_vs_str);

            strcat(axis_123_cmd, axis_2_v);
            gcvt(r_v, sig, axis_2_v_str);
            strcat(axis_123_cmd, axis_2_v_str);

            strcat(axis_123_cmd, axis_2_d);
            gcvt(r_step, sig, axis_2_step_str);
            strcat(axis_123_cmd, axis_2_step_str);

            strcat(axis_123_cmd, axis_2_i);

/* constructing axis 3's motion commands */

            strcat(axis_123_cmd, axis_3_vs);
            gcvt(Rx_v, sig, axis_3_vs_str);
            strcat(axis_123_cmd, axis_3_vs_str);

            strcat(axis_123_cmd, axis_3_v);
            gcvt(Rx_v, sig, axis_3_v_str);
            strcat(axis_123_cmd, axis_3_v_str);

            strcat(axis_123_cmd, axis_3_d);
            gcvt(Rx_step, sig, axis_3_step_str);
            strcat(axis_123_cmd, axis_3_step_str);

            strcat(axis_123_cmd, axis_3_i);

  strcat(axis_123_cmd, axis_123_end);

            write_command(axis_123_cmd);

            printf("\n The motion command is :\n");

            printf("%s\n", axis_123_cmd);

            while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
            /* Waits for the axes to stop */

            printf("\n The axes positions are :\n ");

            message=" 1P ";      /* AXIS 1 POSITION */
            write_command(message);
```

```
        read_answer(answer);
        printf(answer,"\n\n ");

        message=" 2P ";         /* AXIS 2 POSITION */
        write_command(message);
        read_answer(answer);
        printf("\n ");
        printf(answer);

        message=" 3P ";         /* AXIS 3 POSITION */
        write_command(message);
        read_answer(answer);
        printf("\n ");
        printf(answer);
        printf("\n\n");
}

/* Writes a command string to the PC23 */
void write_command(char *c)
{
        while (*c)
        {
                write_character (*c++);
        }

        write_character (13);

        return;
}

/* Writes a single character to the PC23. PC23 commands are generated by sending multiple characters to
it */
void write_character (char gamma )
{
        while (!(inp(STCTRL_ADDR) & IDB_M));
        outp (ADDRESS,gamma);
        outp (STCTRL_ADDR, CHAR_READY);
        while (inp(STCTRL_ADDR) & IDB_M);
        outp(STCTRL_ADDR,CB);
        while (!(inp(STCTRL_ADDR) & IDB_M));
        return;
}

/* Reads a complete PC23 status request response string */
void read_answer (char *a)
{
        while ((*a++ = read_character()) != 13);
        *a = '\0';
        return;
}

/* Reads one character of a PC23 response to a status request and returns the character response.  */
char read_character()
{
        char gamma=0;
        while (!(inp(STCTRL_ADDR) & ODB));
        gamma = inp(ADDRESS);
        outp (STCTRL_ADDR, ACK);
        while ((inp(STCTRL_ADDR) & ODB));
        outp (STCTRL_ADDR, CB);
        return(gamma);
}
```

Appendix I ExWax.c Program Source Codes

```
/* ExWax.c is a program that extract points from a dxf file. The extracted points are then stored inside
another text file. All the points in the destination file are consist of X, Y and Z coordinates. The
destination file contains all the internal and external points of a three-dimensional surface model with
internal cavities. */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>

#define OUTPUT_FILE "c:\\temp\\extract.txt"
#define READ "r"
#define WRITE "w"
#define STRING_LENGTH 81
#define STRIKE_2D "AcDb2dVertex"
#define STRIKE_3D "AcDb3dPolylineVertex"
#define TEN "10"
#define TWENTY "20"
#define THIRTY "30"

main()
{
        FILE *data;
        char filename[STRING_LENGTH];
        FILE *new_data;
        char    axis[STRING_LENGTH];
        float x_coor, y_coor, z_coor;
        char string[STRING_LENGTH];
        int  counter;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        /* attempt to open files */
        data = fopen(filename, READ);
        new_data = fopen(OUTPUT_FILE, WRITE);

        if (data == NULL || new_data == NULL)
        {
                printf("file cannot be opened\n");
                exit(errno);
        }

        counter = 0;

        while (! feof(data))
        {
                fscanf(data, "%s", string);

                if ((strcmp(string, STRIKE_2D)) == 0 || (strcmp(string, STRIKE_3D)) == 0)
                {
                    fscanf(data, "%s", axis);

                    if (counter != 0)
                    {
                              fprintf(new_data, "\n");
                    }
                    else
                    {
                              counter ++;
                    }

                    if ((strcmp(axis, TEN)) == 0)
                    {
                              fscanf(data, "%f", &x_coor);
                    }

                    fscanf(data, "%s", axis);

                    if ((strcmp(axis, TWENTY)) == 0)
                    {
                              fscanf(data, "%f", &y_coor);
                    }

                    fscanf(data, "%s", axis);

                    if ((strcmp(axis, THIRTY)) == 0)
                    {
                              fscanf(data, "%f", &z_coor);
                    }

                    fprintf(new_data, "%10f %10f %10.1f", x_coor, y_coor, z_coor);
          }
     }

     fclose(data);
     fclose(new_data);

     printf("The 3D data file is in c:\\temp\\extract.txt");

     return 0;
}
```

Appendix J Group.c Program Source Codes

```
/* Group.c is a program that group the x, y, z coordinates based on z coordinates because the arrangement
of the data in the extracted text file is not base on z coordinates.
Assumptions: 1) The model is 100 mm in height.
             2) The step over in the section cuts model is 3 mm .
             3) The last layer is at 99mm.                                                    */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>

#define OUTPUT_FILE "c:\\temp\\group.txt"
#define READ "r"
#define APPEND "a"
#define STRING_LENGTH 81

main()
{
        FILE            *xyz_data;
        char            filename[STRING_LENGTH];
        FILE            *group_data;
        float           x_coor, y_coor, z_coor, z;
        int             counter;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        for (z = 0.0; z <= 99.0; z += 3) /* 0 is the 1st layer value */
        {                                /* 99 is the last layer value */
                                         /* each step over is 3 */
                /* attempt to open files */
                xyz_data = fopen(filename, READ);
                group_data = fopen(OUTPUT_FILE, APPEND);

                if (xyz_data == NULL || group_data == NULL)
                {
                        printf("file cannot be opened\n");
                        exit(errno);
                }

                counter = 0;

                while (! feof(xyz_data))
                {
                        fscanf(xyz_data, "%f", &x_coor);
                        fscanf(xyz_data, "%f", &y_coor);
                        fscanf(xyz_data, "%f", &z_coor);

                        if (z_coor == z && z != 99.0)     /* 99.0 is the largest z value */
                        {
                                fprintf(group_data, "%10f %10f %5.1f\n", x_coor, y_coor, z_coor);
                        }
                        else
                        {
                                if (z_coor == z && z == 99.0)
                                {
                                        if (counter != 0)
                                        {
                                                fprintf(group_data, "\n");
                                        }
                                        else
                                        {
                                                counter ++;
                                        }
                                        fprintf(group_data,  "%10f  %10f  %5.1f",  x_coor,  y_coor,
z_coor);
                                }
                        }
                }

                fclose(xyz_data);
                fclose(group_data);
        }

        printf("The 3D data file is in c:\\temp\\group.txt");

        return 0;
}
```

Appendix K ConWax.c Program Source Codes

```
/* ConWax.c is a program that extract points from an external text file. The extracted points are then
converted from x_coor, y_coor and z_coor into x mm, r mm and Rx degree. Assume that the XY plane
reference point is (100,100). */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<math.h>

#define OUTPUT_FILE "c:\\temp\\convert.txt"
#define READ "r"
#define WRITE "w"
#define STRING_LENGTH 81
#define PI 3.141593
#define XREF 100
#define YREF 100

main()
{
	FILE	*xyz_data;
	char	filename[STRING_LENGTH];
	FILE	*xrRx_data;
	float	x_coor, y_coor, z_coor;
	float	x, r, Rx;
	int	counter;

	printf("Please specify the file with correct path such as\n");
	printf("c:\\folder1\\folder2\\filename.txt\n\n");
	gets(filename);

	/* attempt to open files */
	xyz_data = fopen(filename, READ);
	xrRx_data = fopen(OUTPUT_FILE, WRITE);

	if (xyz_data == NULL || xrRx_data == NULL)
	{
		printf("file cannot be opened\n");
		exit(errno);
	}

	counter = 0;

	while (! feof(xyz_data))
	{
		fscanf(xyz_data, "%f", &x_coor);
		fscanf(xyz_data, "%f", &y_coor);
		fscanf(xyz_data, "%f", &z_coor);

		if (counter != 0)
		{
			fprintf(xrRx_data, "\n");
		}
```

```
        else
        {
                counter ++;
        }

        x = z_coor;

        if (x_coor==XREF || y_coor==YREF)
        {
                if (y_coor==YREF && x_coor>XREF)
                {
                        Rx = 0;
                }

                if (x_coor==XREF && y_coor>YREF)
                {
                        Rx = 90;
                }

                if (y_coor==YREF && x_coor<XREF)
                {
                        Rx = 180;
                }

                if (x_coor==XREF && y_coor<YREF)
                {
                        Rx = 270;
                }
        }
        else
        {
                Rx = ((atan((y_coor-YREF)/(x_coor-XREF)))/PI)*180;

                if (x_coor>XREF && y_coor>YREF)
                {
                        Rx = Rx;
                }

                if (x_coor<XREF && y_coor>YREF)
                {
                        Rx = 180 + Rx;
                }

                if (x_coor<XREF && y_coor<YREF)
                {
                        Rx = 180 + Rx;
                }

                if (x_coor>XREF && y_coor<YREF)
                {
                        Rx = 360 + Rx;
                }
        }

        r = sqrt(((y_coor-YREF)*(y_coor-YREF)) + ((x_coor-XREF)*(x_coor-XREF)));

        fprintf(xrRx_data, "%5.1f %10.6f %10.6f", x, Rx, r);

}
```

```
        fclose(xyz_data);
        fclose(xrRx_data);

        printf("The 3D data file is in c:\\temp\\convert.txt");

        return 0;
}
```

Appendix L SaWax.c Program Source Codes

```
/* SaWax.c is a program that re-arrange the x mm, Rx degree and r mm so that the Rx will always start at
the lowest angle in each layer (cross section profile) of the 3D surface model. 1 extra point is added in the
beginning and the end of of each layer so that the 1st point starts at Rx = 0 degree and the last point ends
at Rx = 360 degree.
Assumption - maximum points in each layer are 4000.
            - data will be sorted layer by layer.
The r mm at 0 and 360 degree will be calculated (linear interpolation) based on the 1st and last r mm of
the sorted layer. r mm will be stored as zero decimal point float value.                        */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define OUTPUT_FILE "c:\\temp\\sorted.txt"
#define READ "r"
#define APPEND "a"
#define STRING_LENGTH 81
#define FIRST_POINT 0.000000
#define LAST_POINT 360.000000

void insertion_sort(int size);

float           x[4000];
float           Rx[4000];
float           r[4000];

main()
{
        FILE    *xRxr_data;
        char    filename[STRING_LENGTH];
        int     index;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        /* attempt to open files */
        xRxr_data = fopen(filename, READ);

        if (xRxr_data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }

        index = 0;

        while (! feof(xRxr_data))
        {
                fscanf(xRxr_data, "%f", &x[index]);
                fscanf(xRxr_data, "%f", &Rx[index]);
                fscanf(xRxr_data, "%f", &r[index]);
                if (x[0] != x[index]) /* compare subsequent x with the 1st x */
                {
                        insertion_sort(index);

                        x[0]  = x[index];
                        Rx[0] = Rx[index];
                        r[0]  = r[index];

                        index = 0;
                }
                else if (feof(xRxr_data) && x[0] == x[index]) /* if it is end of file and */
                        {                                     /* and the 1st x same as */
                                insertion_sort(index);        /* the last x */
                        }

                index ++;
        }

        fclose(xRxr_data);

        printf("\nThe sorted file is in c:\\temp\\sorted.txt.\n");

        return 0;
}

/* Sorting the x, Rx and r into ascending order(from less to more) and save the sorted data into another
external file. */
void insertion_sort(int size)
{
        FILE    *sorted_data;
        int     counter, location, idx;
        float   current_x, current_Rx, current_r;
        float   alpha, beta, distance, angle, dist_per_deg, z, incr_decr_dist;

        sorted_data = fopen(OUTPUT_FILE, APPEND);

        if (sorted_data == NULL)
        {
                printf("sorted_data.txt cannot be opened\n");
                exit(errno);
        }

        if (x[0] != x[size])
        {
                for (counter = 1; counter <= size - 1; counter ++)
                {
                        current_x  = x[counter];
                        current_Rx = Rx[counter];
                        current_r  = r[counter];

                        location   = counter;

                        while (location > 0 && Rx[location - 1] > current_Rx)
                        {
                                x[location]  = x[location - 1];
                                Rx[location] = Rx[location - 1];
                                r[location]  = r[location - 1];
                        location --;
                }

                x[location]  = current_x;
                Rx[location] = current_Rx;
                r[location]  = current_r;
        }
}
else
{
        for (counter = 1; counter <= size; counter ++)
        {
                current_x  = x[counter];
                current_Rx = Rx[counter];
                current_r  = r[counter];

                location   = counter;

                while (location > 0 && Rx[location - 1] > current_Rx)
                {
                        x[location]  = x[location - 1];
                        Rx[location] = Rx[location - 1];
                        r[location]  = r[location - 1];

                        location --;
                }

                x[location]  = current_x;
                Rx[location] = current_Rx;
                r[location]  = current_r;
        }
}

if (x[0] != x[size])
{
        distance = r[0] - r[size - 1];
}
else
{
        distance = r[0] - r[size];
}

if (distance < 0)
{
        distance = (-1) * distance;
}

beta = Rx[0] - 0.000000;

if (x[0] != x[size])
{
        alpha = 360.000000 - Rx[size - 1];
}
else
{
        alpha = 360.000000 - Rx[size];
}
```

```
angle = alpha + beta;

if (x[0] != x[size])
{
        if ((angle == 0) && (Rx[0] == 0.000000))
        {
                z = r[0];
        }
        if ((angle == 0) && (Rx[size - 1] == 360.000000))
        {
                z = r[size - 1];
        }
}
else
{
        if ((angle == 0) && (Rx[0] == 0.000000))
        {
                z = r[0];
        }
        if ((angle == 0) && (Rx[size] == 360.000000))
        {
                z = r[size];
        }
}

dist_per_deg = distance / angle;

incr_decr_dist = dist_per_deg * beta;

if (x[0] != x[size])
{
        if (r[0] > r[size - 1])
        {
                z = r[size - 1] + incr_decr_dist;
        }
        if (r[size - 1] > r[0])
        {
                z = r[0] + incr_decr_dist;
        }
        if (r[size - 1] == r[0])
        {
                z = r[0];
        }
}
else
{
        if (r[0] > r[size])
        {
                z = r[size] + incr_decr_dist;
        }
        if (r[size] > r[0])
        {
                z = r[0] + incr_decr_dist;
        }
        if (r[size] == r[0])
        {
                z = r[0];
                }
        }

        fprintf(sorted_data, "%5.1f %10f %5.0f\n", x[0], FIRST_POINT, z);

        if (x[0] != x[size])
        {
                for (idx = 0; idx <= size - 1; idx ++)
                {
                        fprintf(sorted_data, "%5.1f %10f %5.0f\n", x[idx], Rx[idx], r[idx]);
                }
        }
        else
        {
                for (idx = 0; idx <= size; idx ++)
                {
                        fprintf(sorted_data, "%5.1f %10f %5.0f\n", x[idx], Rx[idx], r[idx]);
                }
        }

        if (x[0] != x[size])
        {
                fprintf(sorted_data, "%5.1f %10f %5.0f\n", x[0], LAST_POINT, z);
        }
        else
        {
                fprintf(sorted_data, "%5.1f %10f %5.0f", x[0], LAST_POINT, z);
        }

        fclose(sorted_data);
}
```

Appendix M Select.c Program Source Codes

```
/* Select.c is a program that select points from an external text file. Only the specified radius value data
set(3 values in a row) will be selected.
Assumption - the largest radius of the model is an integer                    */

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>

#define OUTPUT_FILE "c:\\temp\\select.txt"
#define READ "r"
#define WRITE "w"
#define STRING_LENGTH 81

main()
{
        FILE            *sorted_data;
        char            filename[STRING_LENGTH];
        FILE            *selected_data;
        float           x, Rx;
        int             radius, r, counter;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        printf("\nPlease specify the largest radius of your data file in integer.\n");
        printf("(radius is the third column value of the data file.)\n\n");
        scanf("%d", &radius);

        /* attempt to open files */
        sorted_data = fopen(filename, READ);
        selected_data = fopen(OUTPUT_FILE, WRITE);

        if (sorted_data == NULL || selected_data == NULL)
        {
                printf("file cannot be opened\n");
                exit(errno);
        }

        counter = 0;

        while (! feof(sorted_data))
        {
                fscanf(sorted_data, "%f", &x);
                fscanf(sorted_data, "%f", &Rx);
                fscanf(sorted_data, "%d", &r);

                if (r == radius)
                {
                        if (counter != 0)
                        {
                                fprintf(selected_data, "\n");
```

```
                    }
                    else
                    {
                              counter ++;
                    }

                    fprintf(selected_data, "%5.1f %10f %5d", x, Rx, r);
          }
}

fclose(sorted_data);
fclose(selected_data);

printf("\nThe 3D data file is in c:\\temp\\select.txt");

return 0;
}
```

Appendix N Deposit.c Program Source Codes

```
/* Deposit.c contains a lot of functions. The functions are for
(1)  Reading data from a file. The data is consists of axis 1's travel distance(mm), axis 3's rotational
angle(degree) and axis 2's travel distance(mm). They are the absolute distance and angle.
(2)  Calculating incremental motor step to be travelled by axis 1.
(3)  Calculating incremental motor step to be travelled by axis 3.
(4)  Reset the PC23 board. Assuring that the board is ready to accept commands from the user program.
(5)  Constructing and sending command string of axis 1 to PC23 and getting position response string and
print it on the screen.
(6)  Constructing and sending command string of axis 3 to PC23 and getting position response string and
print it on the screen.
(7)  Constructing and sending command string of axis 2 to PC23 and getting position response string and
print it on the screen.
(8)  The model will be transferred back to its original position once it reached 99 mm in height (total step
over distance).
(9)  Writes a command string to the PC23.
(10) Writes a single character to the PC23. PC23 commands are generated by sending multiple characters
to it.
(11) Reads a complete PC23 status request response string.
(12) Reads one character of a PC23 response to a status request and returns the character response.

Assumption:     Motor at axis 2 is used to push the semi-liquid material/wax onto core cylinder block.
                The rotational motion of the motor will be converted to linear motion of the shaft which
                push the wax/semi-liquid material. Thus, the motor step of axis 2 is directly
                proportional to the quantity of the deposition material.

                There is no stoppage time between axis 3 and axis 2. Stoppage time will have to be
                estimated once the actual process works.

                The total step over distance is 99 mm                                    */

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<errno.h>
#include<string.h>

#define FAIL 0X20
#define BIT2MASK 0X04
#define READY 0X16
#define CB 0X60
#define IDB_M 0X10
#define CHAR_READY 0X70
#define ODB     0X8
#define ACK     0XE0
#define ALDONE 0X02
#define HALT (CB | BIT2MASK)
#define RESTART 0x40 /* byte to restart the PC23 */
#define BADADDR 0XFF

#define ADDRESS 0X300 /* PC23's address */
#define STCTRL_ADDR 0X301 /* Status and Control byte address */
```

```
#define STRING_LENGTH 81
#define READ "r"

float axis_1_incremental_step(float x_mm);
float axis_3_incremental_step(float Rx_deg);

void initialise(void);

void send_axis_1_cmd(float x_step);
void send_axis_3_cmd(float Rx_step);
void send_axis_2_cmd(void);
void send_retract_axis_1_cmd(void);

void write_command(char *c);
void write_character(char gamma);

void read_answer(char *a);
char read_character(void);

float     axis_1_step; /* axis 1's incremental steps to be moved */
float     axis_3_step; /* axis 3's incremental steps to be moved */
float     axis_2_step; /* steps are directly related to the size of droplet of the semi-liquid/wax */

main()
{
        float     axis_1_mm; /* absolute distance from reference point */
        float     axis_3_deg; /* absolute angle of rotation */
        float     radius_mm; /* 3D model thickness at certain angle */
        FILE      *data;
        char      filename[STRING_LENGTH];
        int       radius;
        char       *message;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        printf("\nPlease specify the largest radius(mm) of your data file in integer.\n");
        printf("(radius is the third column value of the data file.)\n\n");
        scanf("%d", &radius);

        printf("\nPlease specify the steps that you want the motor to move so that ");
        printf("\nthe plunger will push out the desired droplet size/amount onto ");
        printf("\nthe core cylinder block.\n\n");
        scanf("%f", &axis_2_step);

        data = fopen(filename, READ);

        if (data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }


        /* assumptions : axis 1 coordinate is on the left side of the table
                         axis 3 coordinate is on the middle of the table
                         axis 2 coordinate is on the right side of the table */
        initialise();

        message=" 1MR6 2MR6 3MR6 "; /* matching the motor resolution of PC23 and KS drive */
        write_command(message);

        while (! feof(data))
        {
                fscanf(data, "%f", &axis_1_mm);
                axis_1_incremental_step(axis_1_mm);
                send_axis_1_cmd(axis_1_step);

                fscanf(data, "%f", &axis_3_deg);
                axis_3_incremental_step(axis_3_deg);
                send_axis_3_cmd(axis_3_step);

                fscanf(data, "%f", &radius_mm);
                send_axis_2_cmd(); /* the motor at the deposition equipment will rotate */
        }

        fclose(data);

  send_retract_axis_1_cmd();

        printf("\n\nYour %dmm radius cylinder block has been built by deposition\n", radius);
        printf("process");

        return 0;
}

/* Convert axis 1's absolute distance to incremental motor step */
float axis_1_incremental_step(float x_mm)
{  /* assumption : axis 1's reference point is on the other end of the rotary shaft of axis 3
                 : the 1st value for axis_1_mm is 0 */

        static float prev_x_mm = 0;

        axis_1_step = (x_mm - prev_x_mm)/(0.3E-03);

        prev_x_mm = x_mm;

        return (axis_1_step);
}

/* Convert axis 3's absolute angle to incremental motor step */
float axis_3_incremental_step(float Rx_deg)
{       /* assumption : axis 3's reference point(or axis) is the axis itself */

        static float prev_Rx_deg = 0;

        if (Rx_deg == 0)
        {
                axis_3_step = 0;
        }
        else
        {
                axis_3_step = (Rx_deg - prev_Rx_deg)/(0.72E-03);
        }
        prev_Rx_deg = Rx_deg;

        return (axis_3_step);
}

/* Reset the PC23 board. Assuring that the board is ready to accept commands from the user program. */
void initialise(void)
{
        unsigned char statbyte;

        outp(STCTRL_ADDR, HALT); /* initialise procedure */
        while (!((statbyte = inp(STCTRL_ADDR)) & FAIL) );
        if(statbyte == BADADDR)
        {
                printf ("\n\nInvalid address, check PC-23 dipswitches\n\n");
                exit(1);
        }
        outp(STCTRL_ADDR, RESTART);
        outp(STCTRL_ADDR, CB);
        while( ( (statbyte=inp(STCTRL_ADDR)) & READY) != READY );
}

/* Constructing and sending command string of axis 1 to PC23 and getting position response string and
print it on the screen. */
void send_axis_1_cmd(float x_step)
{
        int dec = 0;
        int sign = 0;
        int ndig = 0;

        char axis_1_cmd[STRING_LENGTH];
        char *axis_1_front = " 1VS10 1V10 1D";
        char *axis_1_step_str; /* axis 1's integer motor step in char/string form */
        char *go_axis_1 = " 1I 1G ";

        char *answer, *message;
        answer = "";

        strcpy(axis_1_cmd, axis_1_front);
        axis_1_step_str = fcvt(x_step, ndig, &dec, &sign);
        strcat(axis_1_cmd, axis_1_step_str);
        strcat(axis_1_cmd, go_axis_1);

        write_command(axis_1_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_1_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 1 position is :\n\n ");

        message=" 1P ";      /* AXIS 1 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
```

```
}

/* Constructing and sending command string of axis 3 to PC23 and getting position response string and
print it on the screen. */
void send_axis_3_cmd(float Rx_step)
{
        int dec = 0;
        int sign = 0;
        int ndig = 0;

        char axis_3_cmd[STRING_LENGTH];
        char *axis_3_front = " 3VS10 3V10 3D";
        char *axis_3_step_str; /* axis 3's integer motor step in char/string form */
        char *go_axis_3 = " 3I 3G ";

        char *answer, *message;
        answer = "";

        strcpy(axis_3_cmd, axis_3_front);
        axis_3_step_str = fcvt(Rx_step, ndig, &dec, &sign);
        strcat(axis_3_cmd, axis_3_step_str);
        strcat(axis_3_cmd, go_axis_3);

        write_command(axis_3_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_3_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 3 position is :\n\n ");

        message=" 3P ";      /* AXIS 3 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
}

/* The motor will rotate. Hence, the leadscrew will move downward and push the droplet of the semi-
liquid material/wax onto the core cylinder block */
void send_axis_2_cmd(void)
{
        int dec = 0;
        int sign = 0;
        int ndig = 0;

        char axis_2_cmd[STRING_LENGTH];
        char *axis_2_front = " 2VS10 2V10 2D";
        char *axis_2_step_str; /* axis 2's integer motor step in char/string form */
        char *go_axis_2 = " 2I 2G ";

        char *answer, *message;
        answer = "";

        strcpy(axis_2_cmd, axis_2_front);
        axis_2_step_str = fcvt(axis_2_step, ndig, &dec, &sign);
        strcat(axis_2_cmd, axis_2_step_str);
        strcat(axis_2_cmd, go_axis_2);

        write_command(axis_2_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_2_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 2 position is :\n\n ");

        message=" 2P ";      /* AXIS 2 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
}

/* Sending command string of axis 1 to PC23 to retract to the original position and getting position
response string and print it on the screen. */
void send_retract_axis_1_cmd(void)
{
        char *answer, *message, *axis_1_retract_cmd;
        answer = "";

        axis_1_retract_cmd = " 1VS10 1V10 1D-330000 1I 1G ";
        write_command(axis_1_retract_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_1_retract_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 1 position is :\n\n ");

        message=" 1P ";      /* AXIS 1 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
}

/* Writes a command string to the PC23 */
void write_command(char *c)
{
        while (*c)
                write_character (*c++);
        return;
}

/* Writes a single character to the PC23. PC23 commands are generated by sending multiple characters to
it */
void write_character ( char gamma )
{
        while (!(inp(STCTRL_ADDR) & IDB_M));
        outp (ADDRESS,gamma);
        outp (STCTRL_ADDR, CHAR_READY);
        while (inp(STCTRL_ADDR) & IDB_M);
        outp(STCTRL_ADDR,CB);
        while (!(inp(STCTRL_ADDR) & IDB_M));
        return;
}

/* Reads a complete PC23 status request response string */
void read_answer (char *a)
{
        while ((*a++ = read_character()) != 13);
        *a = '\0';
        return;
}

/* Reads one character of a PC23 response to a status request and returns the character response.  */
char read_character()
{
        char gamma=0;
        while (!(inp(STCTRL_ADDR) & ODB));
        gamma = inp(ADDRESS);
        outp (STCTRL_ADDR, ACK);
        while ((inp(STCTRL_ADDR) & ODB));
        outp (STCTRL_ADDR, CB);
        return(gamma);
}
```

Appendix O DpsSlp.c Program Source Codes

```
/* dpsslp.c contains a lot of functions. The functions are for
(1) Reading data from a file. The data is consists of axis 1's travel distance(mm), axis 3's rotational
angle(degree) and axis 2's travel distance(mm). They are the absolute distance and angle.
(2) Calculating incremental motor step to be travelled by axis 1.
(3) Calculating incremental motor step to be travelled by axis 3.
(4) Reset the PC23 board. Assuring that the board is ready to accept commands from the user program.
(5) Constructing and sending command string of axis 1 to PC23 and getting position response string and
print it on the screen.
(6) Constructing and sending command string of axis 3 to PC23 and getting position response string and
print it on the screen.
(7) Writes a command string to the PC23.
(8) Writes a single character to the PC23. PC23 commands are generated by sending multiple characters
to it.
(9) Reads a complete PC23 status request response string.
(10) Reads one character of a PC23 response to a status request and returns the character response.
(11) The model will be transferred back to its original position once it reached 99 mm in height (total step
over distance).

It will enable axis 3 to stop for over 1 second between each motion. So that the external control wax
depostion equipment can do the deposition task by dropping a specific amount/size  of semi-liquid
material/wax onto the core cylinder block.

Assumption:     The deposition equipment is not control by the same program. The program will need
                further integration with the program had handle the deposition equipment.

                The maximum step over distance is 99 mm                                    */

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<errno.h>
#include<string.h>

#define FAIL 0X20
#define BIT2MASK 0X04
#define READY 0X16
#define CB 0X60
#define IDB_M 0X10
#define CHAR_READY 0X70
#define ODB      0X8
#define ACK      0XE0
#define ALDONE 0X02
#define HALT (CB | BIT2MASK)
#define RESTART 0x40 /* byte to restart the pc21 */
#define BADADDR 0XFF

#define ADDRESS 0X300 /* PC23's address */
#define STCTRL_ADDR 0X301 /* Status and Control byte address */

#define STRING_LENGTH 81
```

```
#define READ "r"

float axis_1_incremental_step(float x_mm);
float axis_3_incremental_step(float Rx_deg);

void initialise(void);

void send_axis_1_cmd(float x_step);
void send_axis_3_cmd(float Rx_step);
void send_retract_axis_1_cmd(void);

void write_command(char *c);
void write_character(char gamma);

void read_answer(char *a);
char read_character(void);

float    axis_1_step; /* axis 1's incremental steps to be moved */
float    axis_3_step; /* axis 3's incremental steps to be moved */

main()
{
        float             axis_1_mm; /* absolute distance from reference point */
        float             axis_3_deg; /* absolute angle of rotation */
        float             axis_2_mm; /* 3D model thickness at certain angle */
        FILE              *data;
        char              filename[STRING_LENGTH];
        int               radius;
        unsigned long     i;
        char              *message;

        printf("Please specify the file with correct path such as\n");
        printf("c:\\folder1\\folder2\\filename.txt\n\n");
        gets(filename);

        printf("\nPlease specify the largest radius(mm) of your data file in integer.\n");
        printf("(radius is the third column value of the data file.)\n\n");
        scanf("%d", &radius);

        data = fopen(filename, READ);

        if (data == NULL)
        {
                printf("%s cannot be opened\n", filename);
                exit(errno);
        }


        /* assumptions : axis 1 coordinate is on the left side of the table
                         axis 3 coordinate is on the middle of the table
                         axis 2 coordinate is on the right side of the table */

        initialise();

        message=" 1MR6 3MR6 ";  /* matching the motor resolution of PC23 and KS drive */
        write_command(message);

        while (! feof(data))
        {
                fscanf(data, "%f", &axis_1_mm);
                axis_1_incremental_step(axis_1_mm);
                send_axis_1_cmd(axis_1_step);

                fscanf(data, "%f", &axis_3_deg);
                axis_3_incremental_step(axis_3_deg);
                send_axis_3_cmd(axis_3_step);

                /* stop the motion for over 1 second in a Pentium II 300MHz PC */
                for (i = 0; i < 30000000; i ++)
                ;

                fscanf(data, "%f", &axis_2_mm);
        }

        fclose(data);

   send_retract_axis_1_cmd();

        printf("\n\nYour %dmm radius cylinder block has been built by deposition\n", radius);
   printf("process");

        return 0;
}

/* Convert axis 1's absolute distance to incremental motor step */
float axis_1_incremental_step(float x_mm)
{  /* assumption : axis 1's reference point is on the other end of the rotary shaft of axis 3
                : the 1st value for axis_1_mm is 0 */

        static float prev_x_mm = 0;

        axis_1_step = (x_mm - prev_x_mm)/(0.3E-03);

        prev_x_mm = x_mm;

        return (axis_1_step);
}

/* Convert axis 3's absolute angle to incremental motor step */
float axis_3_incremental_step(float Rx_deg)
{       /* assumption : axis 3's reference point(or axis) is the axis itself */

        static float prev_Rx_deg = 0;

        if (Rx_deg == 0)
        {
                axis_3_step = 0;
        }
        else
        {
                axis_3_step = (Rx_deg - prev_Rx_deg)/(0.72E-03);
        }

        prev_Rx_deg = Rx_deg;

        return (axis_3_step);
```

```
}

/* Reset the PC23 board. Assuring that the board is ready to accept commands from the user program. */
void initialise(void)
{
          unsigned char statbyte;

          outp(STCTRL_ADDR, HALT); /* initialise procedure */
          while (!((statbyte = inp(STCTRL_ADDR)) & FAIL) );
          if(statbyte == BADADDR)
                    {
                    printf ("\n\nInvalid address, check PC-23 dipswitches\n\n");
                    exit(1);
                    }
          outp(STCTRL_ADDR, RESTART);
          outp(STCTRL_ADDR, CB);
          while( ( (statbyte=inp(STCTRL_ADDR)) & READY) != READY );
}

/* Constructing and sending command string of axis 1 to PC23 and getting position response string and
print it on the screen. */
void send_axis_1_cmd(float x_step)
{
          int dec = 0;
          int sign = 0;
          int ndig = 0;

          char axis_1_cmd[STRING_LENGTH];
          char *axis_1_front = " 1VS10 1V10 1D";
          char *axis_1_step_str; /* axis 1's integer motor step in char/string form */
          char *go_axis_1 = " 1I 1G ";

          char *answer, *message;
          answer = "";

          strcpy(axis_1_cmd, axis_1_front);
          axis_1_step_str = fcvt(x_step, ndig, &dec, &sign);
          strcat(axis_1_cmd, axis_1_step_str);
          strcat(axis_1_cmd, go_axis_1);

          write_command(axis_1_cmd);

          printf("\n The motion command is :\n\n");
          printf("%s\n", axis_1_cmd);

          while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
          /* Waits for the axis to stop */

          printf("\n The axis 1 position is :\n\n ");

          message=" 1P ";      /* AXIS 1 POSITION */
          write_command(message);
          read_answer(answer);
          printf(answer,"\n\n ");
}

/* Constructing and sending command string of axis 3 to PC23 and getting position response string and
print it on the screen. */
```

```
void send_axis_3_cmd(float Rx_step)
{
        int dec = 0;
        int sign = 0;
        int ndig = 0;

        char axis_3_cmd[STRING_LENGTH];
        char *axis_3_front = " 3VS10 3V10 3D";
        char *axis_3_step_str; /* axis 3's integer motor step in char/string form */
        char *go_axis_3 = " 3I 3G ";

        char *answer, *message;
        answer = "";

        strcpy(axis_3_cmd, axis_3_front);
        axis_3_step_str = fcvt(Rx_step, ndig, &dec, &sign);
        strcat(axis_3_cmd, axis_3_step_str);
        strcat(axis_3_cmd, go_axis_3);

        write_command(axis_3_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_3_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 3 position is :\n\n ");

        message=" 3P ";     /* AXIS 3 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
}

/* Sending command string of axis 1 to PC23 to retract to the original position and getting position
response string and print it on the screen. */
void send_retract_axis_1_cmd(void)
{
        char *answer, *message, *axis_1_retract_cmd;
        answer = "";

        axis_1_retract_cmd = " 1VS10 1V10 1D-330000 1I 1G ";
        write_command(axis_1_retract_cmd);

        printf("\n The motion command is :\n\n");
        printf("%s\n", axis_1_retract_cmd);

        while((inportb(STCTRL_ADDR) & ALDONE) != ALDONE);
        /* Waits for the axis to stop */

        printf("\n The axis 1 position is :\n\n ");

        message=" 1P ";     /* AXIS 1 POSITION */
        write_command(message);
        read_answer(answer);
        printf(answer,"\n\n ");
}
```

```
/* Writes a command string to the PC23 */
void write_command(char *c)
{
        while (*c)
                write_character (*c++);
        return;
}

/* Writes a single character to the PC23. PC23 commands are generated by sending multiple characters to
it */
void write_character (char gamma )
{
        while (!(inp(STCTRL_ADDR) & IDB_M));
        outp (ADDRESS,gamma);
        outp (STCTRL_ADDR, CHAR_READY);
        while (inp(STCTRL_ADDR) & IDB_M);
        outp(STCTRL_ADDR,CB);
        while (!(inp(STCTRL_ADDR) & IDB_M));
        return;
}

/* Reads a complete PC23 status request response string */
void read_answer (char *a)
{
        while ((*a++ = read_character()) != 13);
        *a = '\0';
        return;
}

/* Reads one character of a PC23 response to a status request and returns the character response.  */
char read_character()
{
        char gamma=0;
        while (!(inp(STCTRL_ADDR) & ODB));
        gamma = inp(ADDRESS);
        outp (STCTRL_ADDR, ACK);
        while ((inp(STCTRL_ADDR) & ODB));
        outp (STCTRL_ADDR, CB);
        return(gamma);
}
```

Proceedings of the IASTED International Conference
Robotics and Automation 2000
August 14-16, 2000 – Honolulu, Hawaii, USA

Precision Robotic Manipulator based Rapid Prototyping System

Tang, S. H.*, Sulaiman, S.[#] and Hashmi, M. S. J.[ϑ]

* School of Mechanical & Manufacturing Engineering, Dublin City University, Ireland, 97970018@tolka.dcu.ie or saihong5@yahoo.com.

[#] Department of Mechanical & Manufacturing Engineering, Faculty of Engineering, Universiti Putra Malaysia, Malaysia, suddin@eng.upm.edu.my.

[ϑ] School of Mechanical & Manufacturing Engineering, Dublin City University, Ireland, HashmiS@DCU.IE.

ABSTRACT

Prototyping technologies can be categorised into subtractive and additive processes. Additive prototyping processes are stereolithography (SLA), fused deposition modelling (FDM), selective laser sintering (SLS), and others. NC subtractive prototyping process was created in 1960s and continues to evolve into CNC processes. Additive prototyping technologies can reduce the prototyping time. However, introducing a robotic-based subtractive prototyping process can also shorten the prototyping time significantly. Moreover, the new configuration is more versatile, cheaper and occupies less floor space than the conventional CNC process.

The present paper describes the hardware and software design of a robotic manipulator based subtractive prototyping system. A four degrees of freedom robotic manipulator was powered by a. c. brushless servomotors, which are controlled by separate drives. The drives accept direction and distance signals from an indexer, which in turn is controlled by the user-developed software through a high specification PC. The manipulator is used to hold and manipulate the model material (polystyrene) for ball nosed end milling process. A commercial CAD software is used to develop the NURBS surface engineering models. User-developed C programs are used to communicate and control the robotic manipulator. Several prototypes produced using this facility have shown satisfactory results.

Keywords: Manufacturing, Mechatronics, Robot Control, Rapid Prototyping.

1. INTRODUCTION

There are two important challenges for product manufacturing industry at present. One of it is to find a better way in reducing the product development time substantially. Another challenge is the improvement on flexibility for manufacturing small batch size products and a variety of types of product [1]. Decreasing product development time will shorten the lead-time to market and subsequently, an organisation can secure a bigger market share earlier than its competitors. In fact, more than 70% of senior management staffs rate the lead-time to market as one of the three most important criteria that drive them in the businesses. Thus, the key to success for most of the manufacturers is the capability to provide quality products to market, at the shortest possible lead-time with the right cost [2].

One of the ways to shorten the product lead-time to market is to reduce the prototyping time. Some of the advantages of having a shorter prototyping time are:

1) *Visualisation.* The touch of the real life objects can reveal unexpected problems and sometimes lead to a better design modification.
2) *Verification and Optimisation.* With shorter prototyping time, the design concept verification and optimisation tasks can be accomplished faster.
3) *Iteration.* With shorter prototyping time, it is possible to go through numerous design iterations within a short time.
4) *Planning and Tooling.* By having the physical product at an earlier design stage, process planning and tooling design can be speeded up.
5) *Marketing.* A prototype can be used to demonstrate to the customers regarding the concept, design ideas and the company's capability in producing it.

There are two ways of reducing product prototyping time significantly. One of it is to develop new prototyping technologies like stereolithography apparatus (SLA), selective laser sintering (SLS), fused deposition modelling (FDM), and so on [3]. These methods are categorised under additive prototyping processes. Another method is to improve the principal existing technique like integrating a precision robotic manipulator into conventional machining process like ball nosed end milling. The latter method is considered as subtractive prototyping process.

This paper is concerning about the improvement of the CNC based machining method by introducing a precision robotic manipulator into the prototyping system. The improved system will save as much as 40% of floor space with the same size of workspace [8]. The rig is a general-purpose robotic system suitable for additive and subtractive prototyping processes. However, the current research is focusing in the subtractive prototyping aspect. The hardware of the rig includes a ball nosed end milling equipment, a four degrees of freedom precision robotic manipulator [6], a. c. brushless servomotors, KS-drives [5], PC-23 indexer [4] and a high specification PC. The major software used in the project includes the AutoSurf and the ANSI C programming tool. High density extruded polystyrene is used as the model material in the prototyping process. With the above hardware and software configurations, various three-dimensional complex shaped objects were produced.

2. HARDWARE

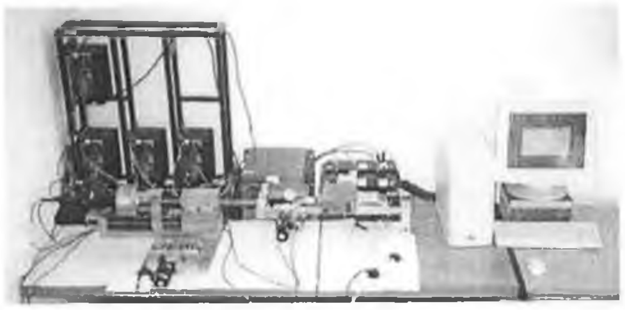


Figure 1. General layout of the rig

General layout of the rig is shown in Figure 1. A Pentium II 300 MHz personal computer is used to control the system by sending commands and receiving responses from the PC-23 indexer. The indexer will in turn communicate with the KS-drives for controlling the a. c. brushless servomotors. The a. c. brushless servomotors drive a four degrees of freedom precision robotic manipulator. The manipulator feeds the polystyrene cylindrical block to the ball nosed cutter for milling three-dimensional complex shaped objects.

2.1 Precision Manipulator

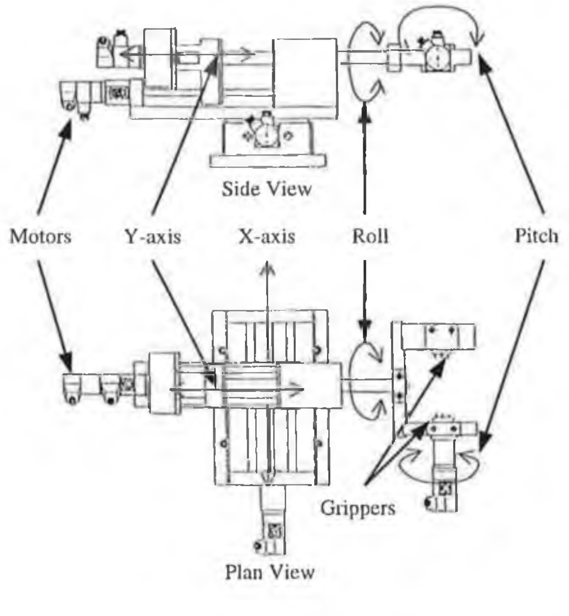


Figure 2. Side and Plan View of the Precision Robotic Manipulator

The precision robotic manipulator is shown schematically in Figure 2. In the diagram, the plan and side views of the manipulator are shown. The length of the manipulator is 1270 mm. Its width and height are 616 mm and 440 mm respectively. The four degrees of freedom are y linear motion axis, x linear motion axis, roll (rotation around y-axis) and pitch (rotation around x-axis). The directions of motion for each motion axis are also shown in the figure by the double arrowhead lines. The weight of the manipulator is about 55 kg. Aluminium alloy (BS HE30 TF) that has the tensile strength of 280 MN/m^2 was used to manufacture the main parts of the manipulator. All the shafts used have hardness of 60 HRC.

The manipulation unit around the x-axis consists of an a. c. brushless servomotor (KS 210) in conjunction with a harmonic drive gearbox (HDUC 14) having a gear ratio of 100:1 for driving a pair of three-pin grippers. The motor for the pitch manipulation can travel an angular distance of 7.2 x 10^{-4} degrees per full motor step at a maximum torque of 15 Nm. The range of the work piece size that can be held by the grippers is 120 mm to 125 mm in length and 40 mm to 150 mm in diameter.

The manipulation unit along the y-axis consists of an a. c. brushless servomotor (KS 220), a flexible coupling (Compumotor No. CPG. 2 – 6), a lead screw with pitch of 1.5 mm and two 16 mm diameter steel shafts. The motor can travel a linear distance of 3.0 x 10^{-4} mm per full motor step. The total linear distance that can be travelled by the unit is 130 mm.

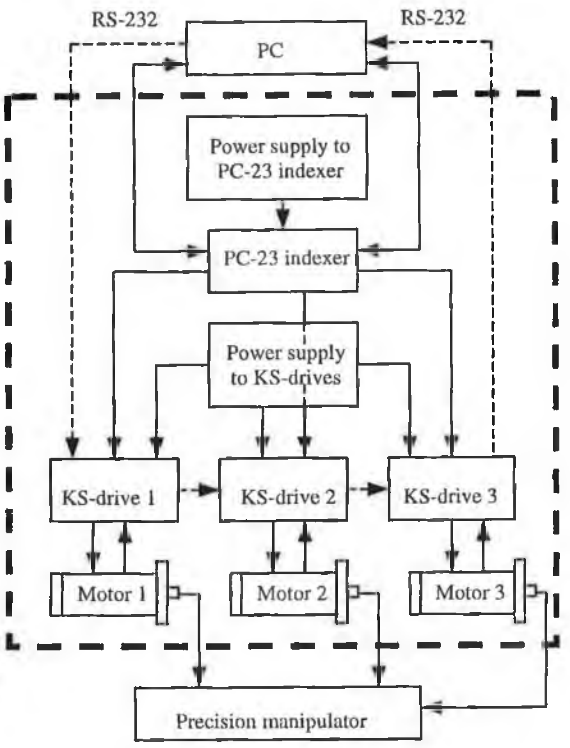
The manipulation unit around the y-axis (roll) consists of an a. c. brushless servomotor (KS 220) in conjunction with a gear box (Drivematic No. SA1002) having a gear ratio of 18:1 for generating the rotational motion around y-axis. The gearbox is attached to a 30 mm diameter steel shaft by a rigid coupling. The motor for roll motion can travel an angular distance of 4.0 x 10^{-3} degrees per full motor step at a maximum torque of 17 Nm. The maximum rotational angle around y-axis is 130 degrees.

The manipulation unit along the x-axis consists of an a. c. brushless servomotor (KS 220), a flexible coupling (Compumotor No. CPG. 2 – 6), a lead screw with pitch of 1.5 mm and a pair of 20 mm diameter steel shafts. A maximum distance of 240 mm can be travelled by this manoeuvring part along x-axis. The motor can travel a minimum linear distance of 3.0 x 10^{-4} mm per full motor step.

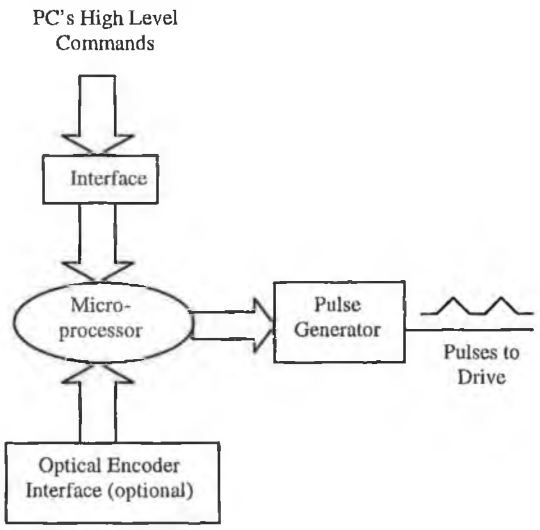
All the prototypes are of cylindrical shapes. As a result, three motion axes are sufficient at the moment. The required motion axes are the manipulation unit along the y-axis, along and around the x-axis.

2.2 Interface System

The successful control of the robotic manipulator based rapid prototyping system requires appropriate interfacing of the machine/device with the PC. The interface system consists of a PC-23 indexer, three KS-drives and three a. c. brushless servomotors. Figure 3(a) illustrated the interface system schematically where the system is bounded by the thickest dotted line.



(a) Interface System



(b) Indexer System Diagram

Figure 3. Interface System and Indexer System Diagram

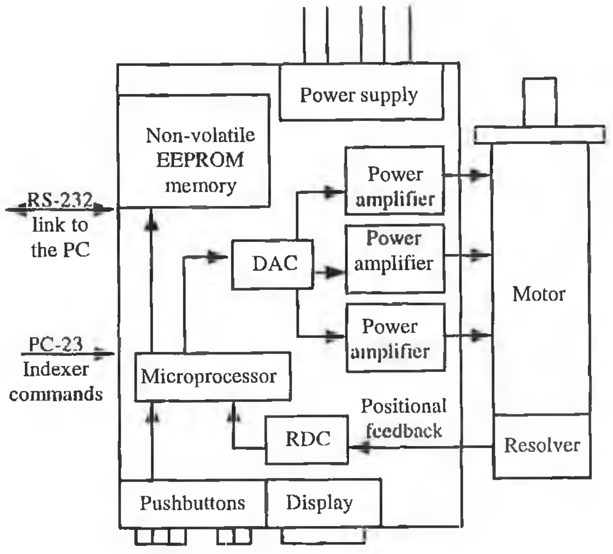
2.2.1 PC-23 Indexer

Figure 3(b) shown the PC-23 indexer system diagram. The indexer uses a 16-bit microprocessor for controlling the motion of up to three motor axes, independently or simultaneously. The indexer is used with an IBM microcomputer (PC, XT or AT) or compatible and suitable for any kind of drive systems that can accept pulsed control signals. There is an indexer main circuit board incorporated into the ISA slot of the PC's motherboard. The indexer operates as an intelligent peripheral in which the onboard microprocessor interprets the PC's high level commands and generates the necessary pulse stream to control the motor velocity, acceleration, position and direction. There are approximately 106 commands for specifying different conditions and operating modes for generating any complex shaped models. Better motion control and responses from the indexer lies in the selection of suitable commands for any particular set of motion sequence.

2.2.2 KS-Drive

The KS-drive consists of an analogue amplifier board and a digital control board. The digital control board handles all the positional compensation PID (proportional, integral and derivative) and V (velocity) gains. The digital control board sends two digitised waveforms from its DAC to the analogue amplifier board. The analogue amplifier board generates its own third phase command and measures the actual motor current to determine the correct pulse width of the voltage to be applied to the motor windings. The drive can be tuned by using either the front panel pushbuttons or the RS-232 port that is link with the PC's serial communication port. There are about 45 commands available for checking, setting, performance optimising and saving the desired parameters into the non-volatile EEPROM memory. Figure 4 illustrated the KS-drive schematically.

For accurate speed control, each indexer axis needs to know the resolution of its controlled motor. The settings of the motor resolution on the KS-drives and the PC-23 indexer must match. The standard motor resolution setting for all the axes on the PC-23 indexer is 25000 steps per revolution. Whereas, the KS-drives for motors KS 210 and KS 220 are configured in the range of 1000 to 16384 steps per revolution. In this project, the KS-drives settings are kept at 5000 steps per revolution. The indexer motor resolution is also set at 5000 steps per revolution although it supports motor or drive resolutions up to 50000 steps per revolution. As a result, the indexer and drives offer the flexibility to change the motor resolutions to suit any particular application.



DAC = digital to analogue converter
RDC = resolver to digital converter

Figure 4. KS-Drive Servo System

2.2.3 A. C. Brushless Servomotor

The a. c. brushless servomotor rotates when the rotor magnetic field tries to follow the stator turning magnetic field created by the three phase a. c. current. By changing the three phase current frequency, the motor achieves different velocities. Step pulses applied first slowly, and then more quickly have the effect of accelerating the motor. The advantages of a brushless motor are:

1) Reduced maintenance.
2) Increased torque/volume ratio.
3) Increased torque at high speed.
4) Simplified in protection compare with more conventional motors.

2.3 Ball Nosed End Milling

The selected subtractive prototyping process is a ball nosed end milling process. A ball nosed cutter has cutting edges at the end and around the cutter. As a result, single point cutting can be accomplished by using the cutting edge at the end of the cutter. The cutting edges at the periphery of the cutter enable multiple cutting operations to be carried out at different intervals of time. The size of the milling chips is relatively small if compare to other machining processes. Hence, the produced surfaces are smoother. The ball nosed end milling can produce virtually any kind of surface if compares to other kind of milling processes. It has the advantage of making holes if compare to the conventional end milling.

The milling material used in the project is an extruded polystyrene. The material is found to be a good choice for milling because minute chips can be removed during the prototyping process and a smooth surface can be created. Its property is not the same as the normal white colour polystyrene that is used in protecting electrical appliances in the packaging industry. The material's minimum density is 32 kg/m^3. Its thermal conductivity is 0.028 W/mK (measured at 10°C). The compressive strength is 300 kN/m^2.

3. SOFTWARE

The AutoSurf package is used to create complex shaped surface models that are section cut into multiple cross sectional layers. The section cut models are then converted from graphic files into neutral format files. The CAM programs are specially devised for extracting surface co-ordinates from the neutral format files, converting the data into different co-ordinate system, sorting, creating motion parameters, communicating with PC-23 indexer and controlling the precision robotic manipulator.

3.1 Surface Modelling

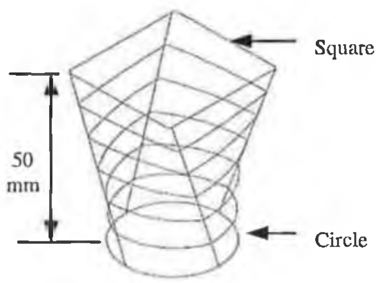
The AutoSurf R3.2 is a window-based, two and three-dimensional mechanical design and drafting software from Autodesk, Inc. Its' modelling system is based on NURBS (non-uniform rational B-spline) curves. A reduced instruction set processor (RISC), with a limited number of instructions is built into the processor to reduce the response time for running some applications on the software development system [7]. Crosshairs and a computer mouse are used to locate geometric shapes within the work area. An X-Y construction plane is used for the two-dimensional mode that uses a three-point origin placed by the user, known as the user co-ordinate system (UCS). In default setting, the Z-axis is perpendicular to the personal computer screen and pointing directly to the user.

AutoSurf has open architecture for easy customisation of menus. The screen menu is the main menu, which includes the drawing editor, configuration, plot, file utility, and operating parameter menus. A dialogue box appears when the selected item is chosen from the pull-down menus to assist the user. Besides using the pull-down menus, the user can type in the commands into the command prompt to call up the functions. The software commands are path dependent. For example, the 'undo' command will remove the screen image and any previous drawing layers up to the earlier drawing level. If compared to the AutoCAD, the AutoSurf has more features and is user-friendlier. In addition, only the AutoSurf can be used to modify the created surface models at a later stage. As a result, the AutoSurf was chosen for the surface modelling process.

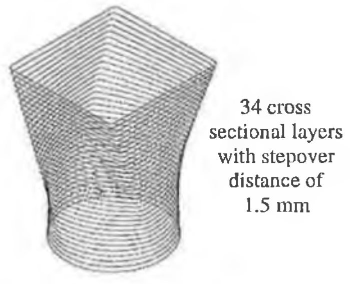
There are four different types of surfaces in terms of the methods used to construct them in AutoSurf:

1) *Surface Primitives*. Created directly by the AutoSurf like cone and cylinder surfaces.
2) *Motion-based Surfaces*. Produced by moving wires through space. Examples are revolved, extruded, tubular and swept surfaces.
3) *Skin Surfaces*. Constructed by applying over a wireframe such as ruled surface.
4) *Derived Surfaces*. Generated from the existing surfaces like blended surface.

Ruled surface modelling is chosen since it is the best method for creating complex shaped surface models. Its models are asymmetric, complex and have predictable bump height and saddle depth (contributed to the predictable milling depth). The created surface models can be section cut into multiple cross sectional layers and converted into DXF (drawing interchange format) entities files. Figure 5(a) and 5(b) illustrates the surface model that was built by using a square and a circle polylines, and its section cut model respectively.



(a) Ruled Surface



(b) Section Cut Surface Model

Figure 5. Ruled Surface and Section Cut Surface Model

The stepover distance of the section cut process determines the amount of layers of the model. At least 400 co-ordinates are generated in each section cut layer. The surface roughness is directly proportional to the amount of co-ordinates in each layer and the number of layers of the models.

3.1 CAM Programs

All computer-aided manufacturing (CAM) programs in this project were written in American National Standards Institute (ANSI) C programming language. The source codes were compiled in Borland C++ version 4.5. ANSI C programming language was used in this project because of the following reasons:

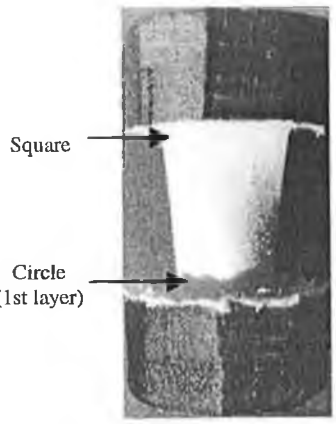
1) It is powerful and flexible. The language itself places no constraints on the user.
2) It is a portable language. A C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification.
3) It is a language of few words. It contains a handful of terms, called keywords, which serve as the base on which the language's functionality is built.
4) It is modular. C code can be written in routines called functions. These functions can be reused in other applications or programs.

Five programs were created so that the data processing procedures can be computerised; path generation process is automated and so on. The programs are listed sequentially as below. All data files have to be processed according to the sequence.

1) *Extract.c*. It is for extracting the entire surface co-ordinates from the DXF entities files. The DXF entities files were transferred from the section cut models.
2) *Convert.c*. It is for converting all the data from Cartesian co-ordinate system to cylindrical co-ordinate system to suit the precision robotic manipulator.
3) *Sortadd.c*. It is for sorting all the co-ordinates according to the height of the model, angles and creating the first and last point for each machining section (layer).
4) *Vd.c*. It is for converting all the distance from millimetre and degree to motor step. It is also calculating all the synchronised velocity for each motion axes.
5) *VsvMt.c*. It is the ultimate motion control program. Its functions include machining time estimation, command construction, communication with the PC-23 indexer, sending commands and receiving responses.

4. RESULT

A number of polystyrene prototypes were produced by using the system. One of the them is shown in Figure 6 together with its milling parameters, time and measurements.



Machining Parameter & Results

Cutter diameter	: 6 mm
Spindle velocity	: 1000 rev/ sec
Feed rate	: 0.375 mm/sec along x-axis
	: 0.9 degree/sec around x-axis

(The feed rate along y-axis is synchronised with the other two motion axes)

Step over	: 1.5 mm
Machining time	: 3.82 hours
Production time	: 4.39 hours

Properties of Circle's 1st layer

Designed diameter	: 31.2566 mm
Measured diameter	: 31.225 mm (average)
Error	: 0.1 %

Figure 6. Milled Prototype with Machining Parameter, Time and Measurements

5. DISCUSSION

Various polystyrene prototypes were produced by using the subtractive prototyping system with the aid of the precision robotic manipulator. The ruled surface models were generally composed of two or more polyline profiles. The profiles were circle, heart, complex, square, cross, star and pentagon. The surface finish of the products is fairly good. Some of the milling chips were still attached on the model surface due to the natural properties of extruded polystyrene block. The surface finish can be further improved if the section cut surface models have more than 34 cross sectional layers – which will contribute to smaller stepover distance and longer production time.

Most of the milling starting points (0^{o}) and ending points (360^{o}) were observed to have a deeper milling depth. The deeper milling depth may be due to the following causes:

1) The starting points were created by the author by using linear interpolation, which is different from the NURBS curve of the engineering models.
2) The cutter was dwelling at the 0^{o} and 360^{o} location of each cross sectional profile for a longer time. It is due to the nature of the Sortadd.c program that may add redundant co-ordinates at the start and end points of each cross sectional layer.
3) The PID and V gains setting of the KS-drives may not be the optimum although the shapes of the milled models were matching the designed models.

The milled polystyrene models were found to have craters. It might be caused by the vibrations of the ball nosed cutter drive unit. The cutter drive can be made more stable if its base is supported from the ground instead of being overhung in the air as at the moment. Fixing an air cushion to it can also reduce the cutter drive vibration. The dimensional accuracy of the product is very good. The dimension of the circle to square shape has been measured and the result is shown in Figure 6. The percentage of error is only 0.1%. The dimensional difference may be due to the positioning system of the manipulator, the milling equipment and the machining vibration. The actual machining time is different from the overall production time because at least 20% of the production time was used in commands transfer and execution.

The difference in command execution speed between the microprocessors of the indexer and the Pentium II PC has affected the interface smoothness in the initial stage. An evaluation program, Mo'Slo v1.32 was used to slow down the execution speed of the PC to be at par with the indexer, but it failed. The issue was later solved by changing the control program so that the PC will wait until it gets the answer from the indexer before continuing to execute its commands.

The original ball nosed cutter was attached to a Bosch hand drill. The hand drill was found to be not suitable because it cannot continuously run for a few hours. As a result, the IKA drive unit that can stand long machining hours was used to replace the Bosch hand drill as the subtractive prototyping tool. The new drive unit has the advantage of controlling the rotating speed of the cutter.

The postprocessor program (Extract.c) for the DXF file can extract all the surface co-ordinates from the model in seconds. The postprocessor was found to be very efficient since all the surface co-ordinates of the section cut model can be extracted out from the section cut models. However, the post-processor programs are solely for the AutoSurf section cut surface model's DXF entities file. The product files of the Extract.c were verified to be correct by plotting the cross sectional profile data into Microsoft Excel. Due to the memory limitation of the PC, the maximum co-ordinates that can be handled by the Sortadd.c program is set at 4000.

6. CONCLUSION

On the whole, a rapid prototyping system using a precision robotic manipulator has been created. The system is comprised of a personal computer, interfacing system (PS-23 indexer, KS-drives and motors), a four degrees of freedom precision manipulator and a ball nosed end milling equipment. The hardware is integrated with the commercial available CAD software (AutoSurf) and self-developed CAM programs (for data processing and motion control) for producing subtractive prototyping models.

The system has the following advantages:
(1) *Low Cost.* The hardware and software configuration is cheap.
(2) *Effective.* The system can produce complex shaped objects with high accuracy.
(3) *Time Saving.* Complex shaped objects can be produced in hours without sacrificing the surface roughness and accuracy.
(4) *Space Saving.* The needed space is much less than the CNC based machines.
(5) *All in One.* All the CAD/CAM activities can be done in one personal computer.

The system can be further improved by:
(1) Replacing the lead screws with ball screw that will provide backlash free motions.
(2) Reducing the vibration of the ball nosed end milling equipment with air cushion.
(3) CAM programs can be linked or integrated to reduce data processing time and error.
(4) For perfect accuracy, the methods of getting the home position of the manipulator in subtractive and additive prototyping processes has to be carried out.

A hybrid rapid prototyping system that comprises of additive and subtractive processes can be created. Objects with internal cavities and out of line of sight areas can be created layer by layer in additive process. The subtractive process can smoothen up the surfaces after each layer is built.

7. REFERENCES

[1] X. Yan and P. Gu, A Review of Rapid Prototyping Technologies and Systems, *Computer-Aided Design*, 28(4), 1996, 307 – 318.
[2] L. Styger, Rapid Prototyping and Tooling Technologies, *Materials World*, 1(12), 1993, 656 – 658.
[3] J.P. Kruth; M.C. Leu and T. Nakagawa, Progress in Additive Manufacturing and Rapid Prototyping, *Annals of the CIRP*, 47(2), 1998.
[4] Compumotor Division, *PC-23 Indexer User Guide* (U.S.A.; Parker Hannifin Corporation, 1987)
[5] Compumotor Division, *KS-Drive User Guide* (U.S.A.; Parker Hannifin Corporation, 1987)
[6] M.T.L. Bhatti, *Effectiveness of Computer Controlled Robotic Precision Manipulator* (Ireland; Dublin City University, 1991)
[7] R.C. Dorf and A. Kusiak, *Handbook of Design, Manufacturing and Automation* (New York; John Wiley & Sons Inc., 1994)
[8] D. A. Belforte, Robotic Manipulation for Laser Processing, *Proceedings of the SPIE-High Power Lasers and Their Industrial Applications*, Vol. 650, 1986, 262 – 270.