CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

THE APPLICATION OF ROBOTICS TO THE ASSEMBLY OF FLEXIBLE PARTS BY SEWING

BY

DAVID GERSHON

Submitted in fulfilment of the requirements for the degree of Doctor of Philosophy.

Being an account of work carried out under the supervision of Professor I. Porat and Mr. C.A. Pinches.

Department of Textile Industries
University of Leeds

March, 1987

CLASS MARK
T. 22426

Theses

ABSTRACT

This thesis concerns the development of a robotic cell to perform assembly and handling operations on cloth. A flexible automation approach was adopted, in which the robot was required to control the cloth panel during both handling and sewing operations, without the aid of hard automation attachments which might limit the flexibility of the system. The cell consisted of an adaptively controlled robot, a hierarchy of controllers, a conventional sewing machine, a two-fingered fabric steering end-effector, and several sensor systems.

A technique was developed for producing a seam parallel to an edge of arbitrary contour, in which two cameras, a cloth tension sensor and the sewing machine's shaft encoder provided the sensory input. Two sensory servo control systems were required, one control system generated the robot's trajectory to maintain a small constant cloth tension, and the other directed the robot to manipulate the cloth panel to maintain a constant seam width.

The design of the cloth tension control was based on the measured frequency response of the open loop system. The seam width control was designed using simulation studies, which accounted for the control transfer function, and non-linearities such as camera pixel resolution, time delays and robot motion limitations.

Several robotic handling techniques were developed, so that a cloth panel placed arbitrarily on the sewing table could be set up for an edge seaming operation, and the cloth could be rotated about the needle. The system's flexibility was demonstrated in the assembly of an irregularly shaped cloth panel, in which three adjacent sides were sewn up.

To Yvette

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to :-

* my supervisors, Prof. I. Porat and Mr. C.A. Pinches for their valuable help and guidance

* Mr. A. Whitehead and the workshop staff for their assistance in designing and building the rig

* Mr. M. N. Moghaddassi for his help and advice with the electronic circuits

* the Textile and Other Manufactures Requirements Board of the Department of Trade and Industry for their financial assistance

* my parents and parents-in-law whose continued support helped make this Ph.D possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbrev.	Meaning	Section
CAD/CAM	Computer Aided Design and Manufacture	1.3.4
CCD	Charqed Coupled Devices	2.8.1
CIM	Computer Integrated Manufacture	2.2
CMRR	Common Mode Rejection Ratio	4.3.5.1
DNC	Direct Numerical Control	2.2
FIGARO	Flexible Intelligent Garment Assembly Robot	2.1
FMS	Flexible Manufacturinq System	1.3.2.4
GPC	General Purpose Communication	2.2
IRQ3	Interrupt Reqiuest No. 3	2.6.2
ISO	International Standards Organisation	3.3.2
ISP	Interrupt Service Procedure	2.3.2.3
OSI	Open System's Interconnection	3.3.2
MITI	Ministry of International Trade and Industry	1.4.1.1
PID	Proportional-Inteqral-Derivative	2.4.4
PIO	Proqrammable Input/Output controller	2.6.2
R & D	Research and Development	1.4.1.2
SSG	Sewing Strategy Generator	7.2.4
$(TC)^2$	Textile and Clothing Technology Corporation	1.4.2.1
UART	Universal Asynchronous Receiver Transmitter	3.3.2
w.r.t.	with respect to	5.2.1.1

CHAPTER 1

INTRODUCTION

1.1. The Clothing Industry

The Clothing Industry is a major UK industry and makes a significant contribution to the economy. In 1985, it was the fourth largest manufacturing industry in the UK in terms of sales (£4.135 billion), and the second largest in terms of employment (193,300); exports amounted to £763 million [1].

However, the industry is confronted with increasingly difficult trading conditions. Clothing manufacture is labour intensive and consequently the industry has suffered import penetration from "low-cost labour" economies - from 1979 to 1985 clothing imports nearly doubled to £1.53 billion [1]. Increased competition from cheap imports has resulted in lower price levels and reduced profitability. Additional difficulties are caused by changes in the market place; retailers are now demanding a quicker response in manufacture and a greater flexibility in design [2].

Clothing industries throughout the industrialized world are facing similar problems, and there is worldwide concern for their future [3,4,5]. The development and implementation of flexible clothing automation has been identified as a vital measure if clothing industries are to meet present day demands [6,7,8].

1.2. Traditional Clothing Manufacturing Processes

There are three main phases in clothing production - preparation of fabric pieces, assembly, and finishing and packaging.

1.2.1. Cloth Preparation

The two-dimensional shapes of the cloth panels are derived from the garment design for the various garment sizes (grading). A drawing is made of the optimized layout of the required cloth panels for the cutting from the cloth roll (lay planning), the cloth is spread out into a multi-ply stack (spreading), and the cloth panel shapes are marked out on the top ply (marking), the cloth panels are cut out in stacks (cutting), and the stacks are tied together in bundles.

1.2.2. Assembly

The cloth panels are assembled using sewing and/or fusion techniques. After each workstation the sub-assemblies are bundled together again before transfer to the next station. During the assembly process, the garment sub-assemblies progress from simple 2-dimensional panels to finish as complex 3-dimensional structures.

An analysis of the sewing operator's productivity showed that on average 20% of the time was spent on sewing, and 66% was spent on work handling (bundle handle, present work to machine, realign, remove and aside etc.) [10].

1.2.3. Finishing and Packaging

The garment is pressed, inspected, labelled and packaged.

1.3. Clothing Automation - State of the Art

1.3.1. Cloth Preparation

When the first automatic cutting system was developed in 1968, there was considerable scepticism as to whether the industry would be prepared to buy such expensive machinery which would require a radical change within clothing firms in order to operate and maintain them [8]. Today, however, computer-controlled cutting systems are commonplace throughout the industry, and many firms have successfully accommodated the needs of complex computerized automation equipment. Computerized systems are now available that fully automate and link the grading, lay-planning, marking and cutting operations [9].

Although multi-ply cutting is still the dominant cutting method, advanced high speed single-ply cutting systems have been developed and they are used in a few specialized applications.

1.3.2. Assembly by Sewing

Four levels of automation can be differentiated as applied to sewing operations [14].

1.3.2.1. Attachments

Labour saving and deskilling attachments can be split into two categories, corresponding to the traditional "sewing versus handling" breakdown of an operator's activities.

Sewing attachments replace or simplify sewing functions. Examples include needle positioning, thread cutting, backtacking, edge guides, photo cells for detecting start/end of cloth, pullers, edge trimmers etc. These devices are usually closely integrated with the sewing machine.

Examples of handling attachments include stackers, ply separation devices, feeders, parts mating devices, parts manipulation or folding devices, etc. These "add-on" handling attachments tend to be more complex, less flexible and less reliable than integrated sewing attachments. This is of course related to the difficulties inherent in handling limp fabric.

1.3.2.2. Semi-Automation

The majority of "automatic sewing units" available today fit into this category, in which conventional sewing machines are combined with selected sewing and peripheral attachments to produce an "engineered work-station". Most sewing functions and some simple handling functions are performed automatically, but most handling activity, including ply separation, parts mating, parts loading etc., are still performed by the operator.

These units are specialized to perform specific sewing operations only and most have limited flexibility to accommodate style changes. They require frequent manual adjustments to accommodate different garment sizes and fabric types. Examples include contour seamers, profile stitchers, pocket setters, dart sewers, button sewers, button-holers, trouser sergers, etc.

Some recent models are computer controlled and therefore do offer a certain degree of programmability. Examples of functions that can be under programmable control are seam length, sewing sequence with time delays, backtacking, stitch condensation, fullness, X-Y pattern sewing, etc. Many semi-automatic sewing machines have been developed based on jig systems, i.e. the cloth panels are clamped in a special-purpose jig and the sewing machine's X-Y table is driven by a contoured groove on the jig.

The Shirley Institute measured the productivity improvements from the use of attachments and semi-automatic sewing units [10].

1.3.2.3. Full Operational Automation

This refers to a cell that performs all cyclic work functions automatically, including ply separation, parts mating, parts feeding, parts manipulation and guidance during the sewing and stacking of completed parts. The operator is required to load the machine with a stack of cut parts, remove completed bundles and transfer bundles between operations.

Several ply separation devices are commercially available, and ply separation devices have been combined with semi-automatic sewing units to produce fully automatic sewing units. Of course these machines still have the disadvantages of limited programmability, frequent manual adjustments and high specialization.

1.3.2.4. Full Sequential Automation

This refers to a completely automatic sequential assembly process in which a series of machines perform both cyclic and non-cyclic work functions, and automatically transfer parts from one automated operation to the next. When a mass production system is required, the assembly line can be based on linking "hard automation" machines and loading and unloading devices.

In a Flexible Manufacturing System (FMS), multi-function programmable production machines are flexibly linked and integrated into a system, optimized for small batch production. Robots and other programmable devices are usually required in an FMS to obtain the desired flexibility.

The automatic sequential assembly of cuffs and collars has been demonstrated by several manufacturers on equipment which is flexible enough to accommodate different styles and sizes. However, flexible automation systems for larger sub-assemblies, which are much more difficult to handle than small stiff cuffs and collars, are not available commercially. There are several research projects to develop flexible clothing automation, which are discussed in section 1.4.

1.3.3. Other Uses of Automation in Clothing Manufacture

The traditional bundle transfer system has been replaced by many garment manufacturers with "Unit Production Systems". In these systems, all the cloth pieces that are required for a sub-assembly (e.g. the two panels for a trouser leg and the waistband) are suspended on a hanger, which is suspended from an overhead conveyor. The hanger is directed to the operator's workstation, under the control of a central computer, and the operator removes the cloth pieces for sewing and then replaces the finished sub-assembly onto the hanger.

The control system permits buffering of hangers and can select different paths as the production circumstances change. The system can track different sub-assemblies using bar codes on the hangers and the conveyor control system can be interfaced to an overall production control system.

In addition to reducing operator handling time, the adoption of conveyor systems provides the facility to link up isolated units of production, both manual and automatic, and it is an essential feature of any FMS concept for the sewing room.

Labour-saving devices and attachments have been developed for fusing and finishing operations, and some fully automatic systems are available for packaging.

1.3.4. Summary

Integrated CAD/CAM (Computer Aided Design and Manufacturing) systems have been introduced into the cutting room and the design office which are comparable in sophistication to the CAD/CAM systems in use in other industries. The sewing room, however, has not yet benefited from flexible automation technology and it is a generation behind the current FMS systems in other industries.

One of the major problems that has held up the development of flexible automation for the sewing room is the fundamental difficulty in handling limp cloth.

1.4. Flexible Clothing Automation Developments

There are several Government and industry sponsored programmes for research and development of flexible clothing automation, throughout the industrialized world.

1.4.1. Japan

1.4.1.1. Automated Sewing System Project

In 1982, the Ministry of International Trade and Industry (MITI) announced an 8-year Large-Scale Project under the title "Automated Sewing System". The objectives of the project, which was funded at ¥13 billion (about £40 million), were to "develop the necessary technologies for an efficient, diversified, small quantity production system" and to produce a working pilot plant by 1989 [11,19].

The Automated Sewing System philosophy is based on flexible assembly of simple 2-dimensional sub-assemblies such as collars and cuffs on a flexible production line, followed by 3-dimensional assembly of all the cloth pieces on a dummy. This approach minimizes the amount of 3-D fabric handling but it relies on some form of temporary fabric stiffening and pre-assembly adhesion.

The project was divided into 4 sections;

a) Sewing preparation - covers all operations from design through to cutting. Research is being undertaken to investigate fabric characteristics, temporary stiffening of the fabric, temporary adhesion of parts before sewing.

b) Sewing and Assembly - covers the development of sewing technology such as 3-dimensional sewing using a small sewing machine on the end of a robot arm, and a multi-functional sewing unit which has different sewing heads on a rotating turret, and attachments stored on a magazine.

c) Material Handling - covers the development of techniques for picking up, mating and transferring fabric pieces. Devices are to be developed for dressing and undressing the dummies.

d) Production control - covers the production control system, related integration systems and automatic recognition of cut pieces.

Although the outline and scope of the project has been reported as described above, no detailed descriptions or technical progress reports have been released, so far.

1.4.1.2. Other Research Projects

In addition to the officially sponsored R & D programme, several Japanese companies are carrying out in-house projects aimed at near-term commercial exploitation. Mitsubishi have demonstrated an automated production line for manufacturing two sides of a travel bag based on jigs [3], and Brother have developed a robotic cell for the assembly of shirt cuffs.

Innovative non-automation production methods have been developed by Toyota. The Toyota Sewing System comprises a line of sewing machines which can be operated in a standing position; each operator controls four to six machines. The system provides flexibility using a combination of group-working practices, manual skills and careful line-planning [12].

1.4.2. U.S.A.

1.4.2.1. The $(TC)^2$ Project

The $(TC)^2$ corporation was set up in 1979 by a consortium of American firms in conjunction with the US government to develop automation for the apparel industry [17,18]. Their first step was to sponsor a study to determine the R & D requirements of the industry. Fabric handling was identified as the category of operations that most urgently required automating. Instead of sponsoring generic research on clothing automation, they decided to take the manufacture of a specific sub-assembly (the sleeve of a man's coat) and automate it.

In 1981, the Draper Laboratories was selected for carrying out the initial R & D. Funding, which was gradually increased, stood at $7.7 million per year in 1986. A prototype machine was completed in 1985 which consisted of the following modular units :

* an automatic loader
* a viewing table and vision system for recognizing parts
* a robot and end-effector which can fold and align the edges
* a transfer door that transfers the parts to the sewing station
* a sewing unit.

An end-effector was developed for a SCARA type robot, which can pick up a single ply, fold and unfold it and orientate it. The end-effector, which comprises three jointed sections, has a degree of programmable configurability. The robot in conjunction with an overhead camera constitutes a fabric handling module.

Two distinct approaches were considered for transporting the fabric during sewing, either a foam backed presser foot or a series of foam backed belts. The presser foot idea was rejected because a different presser foot would be required for each garment size. In the belt system the fabric is held over most of its surface transforming the fabric piece into a rigid object.

In the sewing unit, the fabric is controlled by two upper belt systems, one before and one after the sewing head. The two belt systems are arranged in an interlocking manner which permits the sewing head to move perpendicular to the direction of sewing. Contoured sewing is achieved by generating sewing head position data from a video scan of the fabric piece taken before it enters the sewing unit.

The sewing head's conventional intermittent feed mechanism was replaced with a continuously moving belt top feed system. Fabric fullness was achieved by placing an additional series of belts below the fabric just before the sewing head, so that the two plies could be moved at different speeds.

In 1985 the Draper technology was transferred to the Singer Sewing Company for commercial exploitation. After a preliminary evaluation, they decided to develop a transfer line production system with multiple handling and sewing modules permitting sequential flow down the line. The Draper prototype machine, which had only one sewing and one handling module, had a much lower throughput due to the back and forth flow pattern.

1.4.2.2. The Singer Sewing Corp.

Independent of the $(TC)^2$ project, Singer have developed three ranges of robotic systems for sewing applications. The 100 and 200 MARS robotic systems comprise a four-axis electrically driven gantry robot which can perform fabric pick-up, parts mating and fabric transport during sewing. The 400 MARS robot series are two to five-axis articulated pick and place robots. Singer have provided robotic sewing systems for the manufacture of car seat coverings.

An insight into the Singer approach to the development of flexible clothing automation was given by Lower [8]. Some of the technological breakthroughs that he listed as necessary for flexible garment assembly systems were :-

* Four-axis robots with ability to sew intricately curved seams and ability to pivot smoothly in needle-down position.
* Reliable pick-up and transport end-effectors.

* Accurate stacking systems.
* Prepositioning and orientation systems.
* Preshaping devices for parts mating.
* Sensors for positioning and pick up.
* Vision systems for locating features on cloth panels.

1.4.2.3. Clemson University

Torgerson and Paul reported the development of a vision guidance system for a robot manipulating a fabric panel under a simulated sewing needle, that produced a simulated edge seam [65]. In their experiment, a static overhead camera viewed the panel, which was stationary on a table, and the shape of the panel was extracted from the image using a vision processing algorithm. There was no vision feedback during the simulated sewing operation.

The geometry of a seam around the edge of the panel and 12 mm parallel to the edge, was calculated and a robot trajectory was generated in which the robot, a PUMA 560, moved the panel under a simulated sewing needle. The computed robot motion sequence also rotated the panel about the simulated needle at the end of each seam segment, so that the seam followed the circumference of the panel. The sewing machine was simulated by a pointer which traced out a simulated seam on the panel. The test fabric was heavy denim, which is one of the stiffest fabrics used in garments.

The experiment was performed to determine the accuracy of the vision guided trajectory of the robot, but the interactions between limp cloth, the sewing machine and the robot during sewing were not investigated. Average deviations of 3 to 5 mm were measured between actual and intended seam traces, and Torgerson and Paul attributed these large errors to insufficient resolution in the vision system. However, our experience gained during the research project described in this thesis suggests that the poor accuracy of the PUMA 560 robot is more likely to be the main reason for the large deviations (section 2.4.2.).

An end-effector with four extendable fingers was developed, and the finger configuration could be varied under program control. An algorithm was developed to locate the four fingers and orientate the end-effector optimally over the fabric panel, for any panel shape.

At the end of their paper, Torgerson and Paul recommended further work to investigate the interactions of actual sewing on limp fabric, and to integrate additional vision and force sensors into the system.

1.4.3. Europe

1.4.3.1. The BRITE Project

The European Commission launched the BRITE project in 1985 to promote "pre-competitive technological R & D, including pilot and demonstration projects in new production technologies suitable for products made from flexible materials". In the first three-year phase, 13 projects have been approved which cover the whole spectrum of clothing production.

1.4.3.2. Non-BRITE Research

No details have been published of German clothing automation research although several projects are underway. Semi-automated machinery has been developed by CETIH in collaboration with French shirt manufacturers.

Nilsson, in Sweden, has described in detail a concept for a fully integrated system for manufacturing garments, however no experimental results have been reported, as yet [16]. Nilsson acknowledges that manual assembly of complex three dimensional sub-assemblies will be essential for the foreseeable future, and therefore his production concept incorporates both automated and manual stations linked together within a single CIM environment.

1.4.3.3. UK Research

Hull University have developed a ply separation device and vision systems for parts recognition and for alignment, and they have demonstrated a robot-based transfer line for partial assembly of men's underwear.

Durham University have developed dedicated devices for ply separation and alignment, and they have developed a transfer line for partial assembly of underwear.

Courtaulds Clothing Ltd. have developed a system in which a robot feeding fabric to a sewing machine with synchronization of robot and feed speed, although it has not been demonstrated publicly.

1.5. Comparison of Flexible Clothing Automation Approaches

1.5.1. Introduction

Almost all flexible automation systems include a robot, but the role of the robot in the cell can vary considerably between systems. The robot might have a simple supporting role, e.g. loading a machine tool, or the robot may have the central role in the performance of the manufacturing operation, e.g. a robotic sheep shearing cell [13]. When the robot is required to perform the central function of the cell, the performance of the manufacturing process is limited by the control capability of the robot. Robot control systems are usually categorized into five groups [28], as follows :-

Sequence Control - a sequence of robot motions is determined by mechanical or electrical hardware. The sequence can be reprogrammed by manual adjustments.

Playback Control - an operator guides the robot to a location and the coordinate information is recorded (i.e. on-line programming). When required, the robot can move to the taught location.

Numerical Control - locations can be computed in terms of a coordinate system relative to a frame of reference and the robot can be directed to those locations (off-line programming). Straight line motions and other motions with a defined continuous path can be performed by computing intermediate locations between the start and end points using interpolation schemes.

Adaptive Control - an adaptive robot uses sensory feedback to perform a task in which the desired robot trajectory is not known accurately in advance. For example, some robotic welding systems incorporate a vision system to measure the workpiece geometry ahead of the welding tool, so that the robot's trajectory can be calculated in real time. Thus, different workpieces can be welded without requiring accurate programming of the workpiece's profile or accurate jigging to hold the workpieces and the welding system can accommodate deformation of the workpiece during the weld. This sensor-based real time robot path control is often referred to as "sensory servoing".

Intelligent Control - an intelligent robot can decide how it is going to perform a task, using a world model (which represents the environment, the robot and the task), a knowledge base and an expert system for reasoning and decision making.

1.5.2. $(TC)^2$ Approach

In the $(TC)^2$ project the robot had a numerical control capability. The overhead camera and associated vision processing hardware and software located the initial position of the cloth panel, but during the subsequent handling operation, the robot trajectory was predetermined and there was no real time sensory feedback [64].

The role of the robot was restricted to performing handling operations only, and the sewing operations were performed by the sewing unit. The sewing unit was a programmable device with two degrees of freedom, belt motion and sewing head motion. However, this modular concept, in which the handling and sewing functions were performed by separate devices, limits the flexibility of the system. Some handling operations require intimate co-operation between the handling robot and the sewing machine, e.g. rotating the cloth about the needle between seams.

The sewing unit had only a numerical control capability. The motion of the belts and of the sewing head was predetermined by the visual measurements of the panel's initial orientation and position prior to the sewing operation. Consequently, the accuracy of the sewing process is dependent on the ability of the belt system to hold the cloth rigid, throughout the operation. In practise, however, many fabric materials will buckle or slip during the process in an unpredictable manner, and the sewing unit has no means to detect or correct this. Our experience suggests that the buckling tendency would be worse when sewing along intricately curved contours, due to the shear forces on the cloth created by the perpendicular motions of the belts and the sewing head (section 5.4.4).

1.5.3. The Clemson Approach

In the Clemson project, the robot had a numerical control capability and the robot manipulated the cloth during both sewing and handling operations. The vision system provided the initial position and orientation of the cloth panel only, and no real time sensory feedback was provided. The $(TC)^2$ attempts to solve the problem of slipping and buckling of the cloth by rigidly holding the cloth with a system of belts. The Clemson approach relies on the stiffness of the heavy denim fabric and the multi-fingered support. Torgerson and Paul acknowledged that sensory feedback would be required if flexible fabrics were to be sewn by a real sewing machine.

1.5.4. The Adaptive Robot Approach

A more ambitious approach to the flexible automation of garment assembly operations, is to develop a robot with an adaptive control capability, which can perform the operations based on real time sensory feedback. If the adaptive robot can detect slipping and buckling of the cloth during sewing operations and correct its trajectory accordingly, then neither belts nor any other restraining devices would be necessary to control the unstable cloth. Consequently, the same robot could perform both sewing and handling operations, and the flexibility of the system could be maximized.

In the adaptive robot approach adopted in this project, the robot was given the central role of performing all sewing and handling operations in conjunction with a conventional sewing machine. The limp nature of fabric was accommodated by the real time control of the robot trajectory, derived from sensory measurements taken during the sewing or handling operation.

No hard automation attachments or devices were fitted to the sewing machine which might limit the flexibility of the system, e.g. a cheap edge guide can de-skill production of edge seams, but the attachment would have to be removed before the same machine could be used to sew on a pocket.

The adaptive robot approach is analogous to employing a skilled operator on a basic sewing machine, in place of a semi-skilled operator on a semi-automatic machine. The former is more expensive but can perform a greater range of operations on a greater range of materials. By developing the robot's skills and by keeping the sewing machine simple, a single flexible automation cell should be able to perform the same functions that are currently performed by a wide range of different types of semi-automatic sewing stations.

1.5.5. The Intelligent Robot Approach

The robot sewing and handling skills were provided by its adaptive control capability. An intelligent control capability is also required, if the flexible sewing cell is to adapt itself automatically between batches. Without this reasoning ability, the cell would require extensive reprogramming and testing for each product, which may differ from previous products in its material, shape, size or sequence of operations.

The requirement for an intelligent capability is further discussed in section 7.2.4.

1.6. Clothing Automation Research at Leeds University

The Department of Textile Industries at the University of Leeds has been researching into clothing automation since 1982. In addition to the development of actual devices and techniques for clothing automation, research has been aimed at understanding and analysing the fundamental problems involved in handling limp fabric.

A ply separation device was developed which can pick up a single ply of fabric from a stack, with very high reliability. The device is flexible in terms of shape, size and fabric. A vision system was developed which, when used in conjunction with either a robot or a dedicated device, can align a cloth panel of any shape or size. A technique for accurately placing one ply on top of another is under development.

The development of a flexible sewing station, based on the adaptive approach described in section 1.5.4, is described in the subsequent chapters. Although several clothing automation projects based on adaptive robotics maybe underway elsewhere, this project appears to be the first to be reported in a refereed publication [66] (see Appendix I).

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

CHAPTER 2

FIGARO - A ROBOTIC SEWING DEVELOPMENT SYSTEM

2.1. Overview

A robotic sewing system, referred to by the acronym - FIGARO (Flexible Intelligent Garment Assembly RObot), was developed which comprised a hierarchy of controllers, a robot and a sewing machine. The system was used to investigate robotic sewing and handling techniques in accordance with the flexible automation approach outlined in section 1.5.4.

Fig. 2-1: General View of FIGARO System

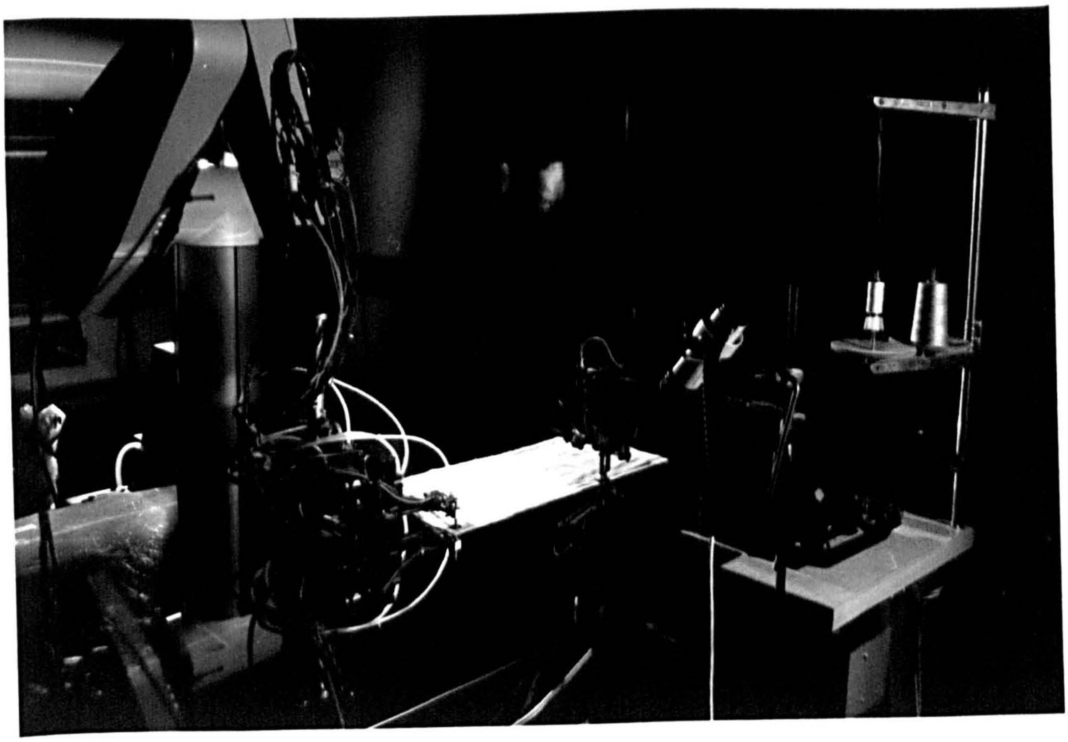


Fig. 2-2: Edge Seaming Operation

A robotic sewing technique was developed, which could produce either a straight seam or a seam parallel to the edge of a cloth panel of arbitrary contour. The sewing technique was based on real time multi-sensory servo control of the robot during the sewing operation. The edge seaming technique involved the following stages :-

a) The robot sets up the cloth panel by sliding it into position, with the sewing machine's needle at the beginning of the seam.

b) The robot repositions its fingers, so that they hold the far end of the cloth panel against the sewing table.

c) The sewing machine is started, and the robot controls the cloth panel throughout the sewing operation. The robot motion is determined in real time by two superimposed servo control systems, a tension control system and a seam width control system :-

(i) The tension control system ensures that the robot moves forward with the cloth and maintains a small cloth tension throughout the sewing operation.

(ii) The seam width control system ensures that the robot rotates the cloth panel about the sewing needle in order to track the edge contour and produce a seam parallel to the edge.

d) When the end of the seam is detected, the sewing machine is stopped.

The straight seaming technique was identical to the edge seaming method but without the seam width control system.

This chapter describes the primary functional units of the FIGARO system, their interfaces and the hierarchical control concept which was implemented. The development of a real time path control capability, on which the cloth tension and seam width control systems were based, is described in Chapter 3. The development of the cloth tension and seam width control systems are described in Chapters 4 and 5 respectively. The techniques that were developed to set up the cloth panel for the sewing operation, and the development of additional cloth handling techniques are described in Chapter 6.

2.2. Hierarchical Control Structure

A hierarchical control structure (fig. 2-3) was chosen to provide an adaptive robot control capability (section 1.5.4). Many development and commercial adaptive robot systems have been based on similar hierarchical control structures [29,30,31,32] rather than on a control structure in which the robot controller controls the entire station.

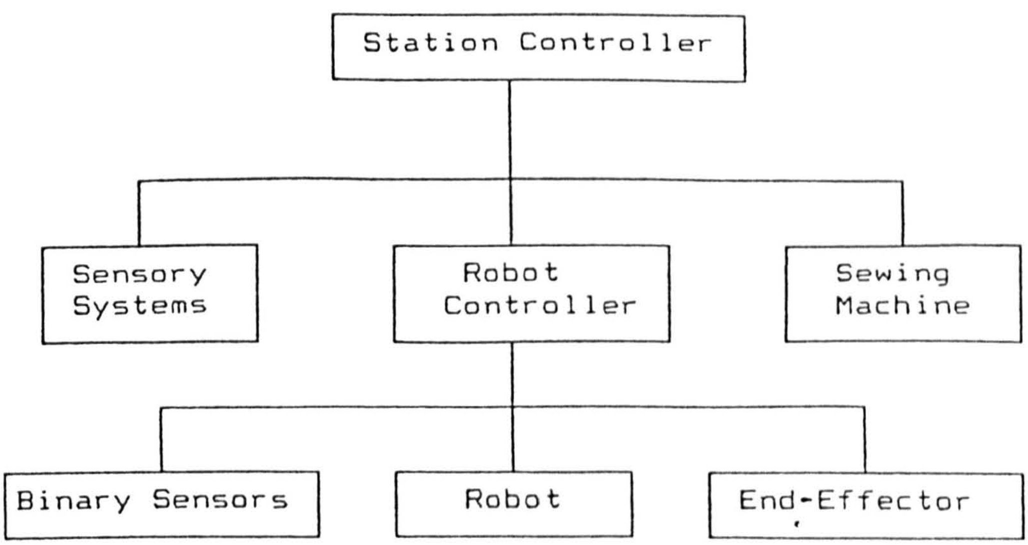


Fig. 2-3: FIGARO Hierarchical Control Structure

In the hierarchical concept, the station controller has equal access to all the major sensors and actuators, and the robot sub-system is regarded as one of the station's actuators. This approach encourages modularity during development of the sub-systems, and facilitates the integration of several complex sub-systems, e.g. more than one robot, vision systems, DNC machines etc. The hierarchy can be readily extended upwards by putting several station controllers under a cell controller (i.e. an FMS cell), which in turn could be controlled by a process controller within a CIM (Computer Integrated Manufacturing) scheme [33,34].

In the FIGARO system, the station controller accepts sensory data in real time, computes a robot trajectory and transmits the robot coordinates to the robot controller. The station controller also coordinates robot motions in conjunction with the sewing machine. The robot controller converts the robot coordinates into joint angles and directs the robot along the required path.

For convenience, some of the simple binary sensors (e.g. photo cells, microswitches) and actuators (e.g. pneumatic cylinders) which were integrated into the end effector, were interfaced to the robot controller. All other actuators and sensors were directly interfaced to the station controller.

Two communication channels were developed between the station controller and the robot controller, the GPC channel for General Purpose Communications and the ALTER channel, which was dedicated to the high speed transfer of real time robot trajectory data.

2.3. Station Controller

2.3.1. Hardware

The IBM AT microcomputer, which was selected for the station controller, is a general purpose microcomputer with a large variety of software development tools and hardware options available. Furthermore, IBM have published comprehensive technical manuals for the AT and for its operating system, which facilitate the development and integration of non-proprietary software and hardware.

FIGARO's IBM AT had the following features :-

* Intel 80286 16-bit microprocessor operating at 6MHz
* Intel 80287 math coprocessor
* 512KB of RAM
* 6 spare expansion slots for customized adapters
* 20MB fixed disk drive
* 16 levels of system interrupt
* 7 channels for direct memory access (DMA)
* 3 programmable timers
* real time clock

2.3.2. Software

2.3.2.1. Requirement for Multi-Tasking

The IBM AT was required to perform the following processes :-

* ALTER communications management (section 3.3)
* GPC management (section 2.6.2)

* Read sensors and calculate robot trajectory (section 4.4.1)
* Control sequence of sewing and handling operations
* User/supervisor interface (section 6.1.1)
* Decision making processes (section 6.3.3)
* Display runtime messages on the screen
* Print out performance and debugging data

These processes were executed in real time and required concurrent execution, therefore a multi-tasking environment was necessary.

2.3.2.2. AMX-86 Multi-Tasking Executive

The AMX-86 multitasking executive [37], on which the FIGARO software was based, provides software facilities which are required in complex real time applications. The AMX-86 is a program which can schedule the pseudo-concurrent execution of application Tasks on a single microprocessor. Additional considerations in selecting the AMX-86 system were that its compatibility with the IBM AT, and C language interface, permit the development of real time software with a high level language. The operation of an AMX-based system is described in fig. 2-4.

2.3.2.3. Tasks

In a multi-tasking system, the software is split up into independent application modules called Tasks. Each Task is treated as a separate program, executing independently of other Tasks. A major distinction between multi-tasking and single-tasking systems is the way in which a Task is called. When a Task is called, a request is passed to the scheduler which will eventually execute the Task in conjunction with other real time demands on the processor, according to a priority scheme. The caller is not suspended after making a call, but may continue irrespective of the status of the called Task. Pending calls to a Task can be queued and given different priorities.

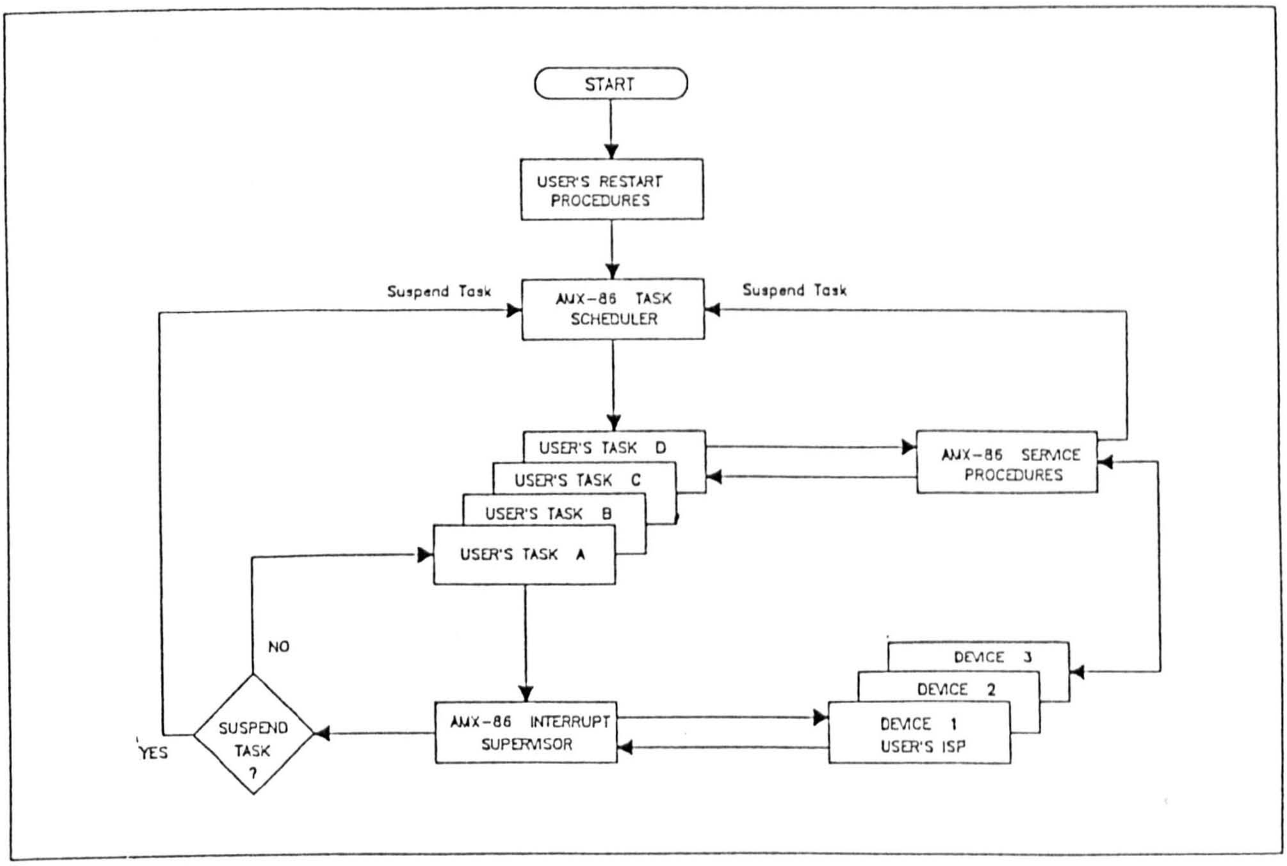


Fig. 2-4: AMX-86 Multi-Tasking Executive

Each Task should perform a clearly defined function, and the logical breakdown of a complex real time problem into independent Tasks is a crucial step in the development of real time software.

A Task can be initiated in one of several ways :-

* It can be started immediately after AMX has completed its initialisation phase by a Restart Procedure.
* It can be started after a time interval has elapsed, by a Timer.
* It can be started by a software or hardware interrupt, by an Interrupt Service Procedure, or ISP.
* It can be started by another Task.

Parameters can be passed to a Task from the caller, e.g. the PRNT Task, which displays or prints out messages, is passed a pointer to a message string when it is called. Concurrent execution of Tasks can be controlled and synchronized through various wake/wait facilities. A Task can suspend itself unconditionally until a Task, Timer or an ISP awakes it, or the "wait" can be conditional on the execution of a called Task, or a timeout limitation can be specified.

Definitions of and relationships between Tasks concerned with FIGARO's adaptive capability are described in section 3.3. The software organisation for the overall system is described in section 6.1.1.

2.3.2.4. Scheduling and Priorities

The main function performed by the AMX system is the scheduling of the processor resources between Tasks, ISP's, Timers, etc. At any given time a number of Tasks may be "active", i.e. waiting for access to the processor, but only one can have access at a time.

Each Task is given a Task number which determines its priority, and the scheduler selects the active Task with the highest priority. Only when that Task has become inactive (i.e. either it has terminated or entered a "wait" state), can the next highest active Task be given access to the processor.

Since a Task which receives parameters can be called several times with different parameters, AMX provides a queueing facility to take care of pending calls to a Task, (e.g. several different messages can be queued to the PRNT Task for printing). Calls to a Task can be given different priority so that, for example, an error message can be given priority over a status message.

2.3.2.5. ISP's and Timers

Immediate response to an external event can be generated through an Interrupt Service Procedure (ISP). When the processor is interrupted by a hardware interrupt, further interrupts are temporarily disabled and then the AMX Interrupt Supervisor directs control to the appropriate user-defined ISP. An ISP should be a short routine that services the interrupt quickly, so that the disabled interrupts may be re-enabled as soon as possible.

For example, the communication ISP, COMISP (section 3.3.2), is invoked when the serial port receives a byte. This interrupt is serviced by reading the byte from the port and putting it onto a circular list. If necessary, it awakes the Task that is waiting for the byte, before returning control to the AMX Interrupt Supervisor.

Timers are user-written procedures which are executed at specified time intervals after they were called.

2.3.2.6. Resource Management

The AMX Resource Manager provides circular lists, buffer pools and other facilities for the orderly use of the computer's resources by concurrent Tasks. Data can be added to or removed from circular lists without any possible collision between Tasks (i.e. the 'mutual exclusion problem' is circumvented by restricting access to critical processes [70]).

2.3.2.7. AMX Configuration Module

To use AMX, a configuration module has to be written, which specifies the names of the Tasks, Restart Procedures and Timers in order of priority; the queue lengths required for each Task; and the storage requirements for stacks, heaps and buffers, etc. A Configuration Builder facility assists in the construction of this module.

2.3.2.8. Languages

The majority of the modules were written in the C programming language, except for some of the communication procedures and the configuration module, which were written in 8086 Assembler. Full listings of all the software modules are given in Appendices A,B,C,D and E.

2.4. PUMA 560 Robot

The PUMA 560 industrial robot was selected because of its advanced VAL II control and programming system; the ALTER function facilitates the development of an adaptive robot capability. In addition, the PUMA 560 has been used extensively in research and development and its performance and characteristics have been widely reported [35,36].

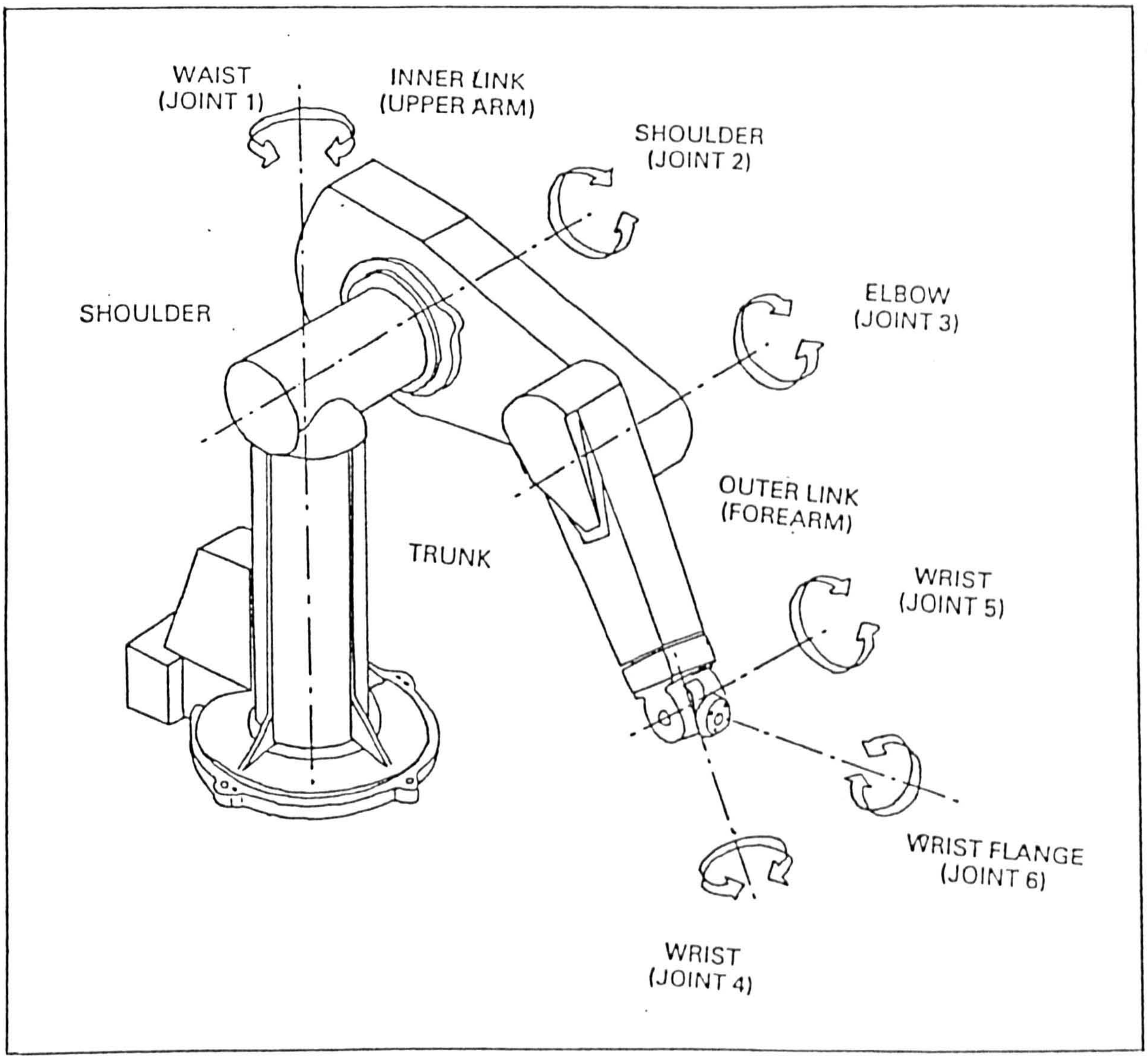


Fig. 2-5: The PUMA 560 Robot

2.4.1. Mechanical Specification

The PUMA 560 is a six-degree-of-freedom, general-purpose, assembly robot with six revolute axes. The configuration, size and proportion of the robot's limbs are imitative of the human arm and torso. The robot has a spherical working volume with a 0.92 m radius, and can carry a maximum load of 2.3 kg including the end-effector. The limbs and joints of the PUMA 560 are named in fig. 2-5.

An anthropomorphic six-axis robot, like the PUMA 560, can reach most points in its workspace by assuming one of eight possible spatial configurations, as follows :

- either RIGHTY or LEFTY, i.e. the first three joints resemble a human's right or left arm

- either ABOVE or BELOW, i.e. the robot's elbow points up or down

- either FLIP or NOFLIP, i.e. the wrist (joint 5) works in negative or positive angles.

With the maximum load, the maximum acceleration of the end-effector is 1 g, the maximum velocity is 1 m/s and the maximum straight-line velocity is 0.5 m/s.

The robot has good repeatability (±0.1 mm) which is dependent on potentiometer resolution, arm stiffness, backlash and servo deadband. This is the relevant precision specification when the robot is programmed using taught locations only ("on-line programming"). However, when the robot is programmed using computed locations ("off-line programming"), then the robot's absolute accuracy is significant. In the FIGARO application, the majority of robot motions involved computed locations rather than taught locations, therefore good absolute accuracy was necessary.

However, the PUMA 560 does not have good absolute accuracy, in common with many other industrial robots, since they were originally intended for on-line programming only. Absolute accuracy is dependent on the accuracy of a matrix transformation which converts a location's coordinates into joint angles. Furthermore, this transformation calculation is sensitive to accumulated round-off errors and to differences between the mathematical model of the robot's geometry and the robot's actual geometry (due to manufacturing tolerances and distortion of the robot's structure).

2.4.2. Calibration

The robot was calibrated in accordance with the manufacturer's instructions using the V2POT5X0.1 program supplied with the robot (section 8.6 of reference [20]). The manual defines two reference positions for the robot arm, the "READY" and "POTCAL" positions, which are used in the calibration procedure. A careful check showed that the manufacturer's alignment marks had been placed inaccurately on the robot arm.

However, even after redrawing the READY and POTCAL alignment marks, and after a further calibration, the absolute accuracy was ±4 mm in the X, Y and Z directions (i.e. 7 mm RMS), and ±0.2° for rotations about the Z axis. This was measured by programming the robot to move to a location at a specific linear or angular offset to the original location. The robot's accuracy deteriorates even further towards the inner and outer limits of its working envelope.

The absolute accuracy of the PUMA 560 is investigated more fully in reference [22], which describes a different method for measuring robot accuracy, where the robot is positioned at an arbitrary location with a RIGHTY configuration. The robot's configuration is then changed to LEFTY and the robot is commanded to move to the same location. This is a particularly stringent test which exaggerates any inaccuracies in the robot system. El-Zorkany [22] reported an RMS error of 16 mm and 1.6° with this test method and the FIGARO robot showed an RMS error of 25 mm.

2.4.3. Electrical Specification

The PUMA 560 robot was supplied together with a system cabinet which provided all the necessary power and control facilities, a VDU terminal with an integral floppy-disk drive, and a manual control unit.

The system cabinet comprises :

* A power tray which provides filtered power supplies.

* A control module which comprises a DEC LSI-11/73 computer, 128 KB non-volatile memory, serial interfaces for peripherals and communications, a digital servo system for each axis, and all the necessary signal interfaces between processors and motors.

* A switch panel which houses the main operator switches.

* An I/O module which contains 40 solid-state relays for binary input/output control signals.

* A power amp module which contain a servo amplifier with monitoring circuitry for each motor.

2.4.4. Robot Control Design

Each axis is driven by a permanent-magnet dc servomotor, and a potentiometer and an incremental encoder are mounted onto each servomotor. The potentiometer provides an absolute position signal and the encoder provides both relative position and velocity signals. Each servomotor is controlled by a digital servo system, based on the 6502 microprocessor, and an analog servo amplifier, using a PID (proportional integral and derivative) control scheme with current feedback [21].

In the PUMA control system, each axis is controlled independently of the other axes, so that coupling effects between joints and the gravity and load effects are ignored. Although some wobble and other dynamic errors are noticeable, the PUMA has satisfactory control, but at the expense of relatively slow speed due to an overdamped system. More sophisticated control methods have been suggested that would account for coupling inertia, friction, gravity and loading effects, and would result in reduced structural stiffness requirements, smaller motors, lower energy inputs and faster speed, as well as improved dynamic control [23,24,25,26,27].

LEEDS UNIVERSITY LIBRARY

2.5. VAL II Robot Control and Programming System

VAL II is one of the most advanced robot programming languages commercially available today [39]. The language has PASCAL-style control structures, manipulation of location transformations, editing and debugging facilities, interrupt handling with priority scheduling, several robot motion control modes, communications support on different levels and a wide range of functions and operators, in addition to the ALTER facility which permits real time trajectory control by an external computer [38].

2.5.1. Robot Motion Control Modes

When the VAL II system processes a robot motion command, an interpolation function is used to automatically generate a series of intermediate locations between specified initial and final locations [45]. This method ensures that the joints move in a coordinated, predictable fashion between the two locations. The programmer may select between two interpolation schemes, as follows :-

- joint interpolated motions are generated by interpolating the joint positions from their initial values to their desired final values so that all the joints complete their motions simultaneously.

- straight-line interpolated motions are generated by interpolating the cartesian tip location and computing the joint positions necessary to move the robot tool tip along a straight line. The maximum speed for straight line motions is only half that of joint interpolated motions.

VAL II includes a continuous path feature, which can control the transition between successive motion segments in a sequence to produce a smooth, continuous motion. VAL II ensures that there is smooth acceleration and deceleration for each motion sequence.

VAL II also permits the user to program the robot to move along a mathematically defined trajectory. In these procedural motions, the robot motion is executed in parallel with a VAL II program loop in which the robot trajectory is computed in small increments. The transitions between computed motion segments are automatically smoothed by the continuous path feature.

In addition to the programmed robot motions described above, VAL II permits real time path control with the ALTER facility. The ALTER facility is described in the next chapter.

2.5.2. Motion Control Parameters

Robot motion along a programmed trajectory can be further specified by the following parameters:

SPEED - tool speeds can be specified either in mm/sec or in terms of a percentage of a maximum speed.

COARSE/FINE - this parameter specifies a low or high tolerance position requirement for the hardware position servos.

NONULL/NULL - final position checking of all the joints can be avoided between consecutive motion segments, if high speed and low accuracy are required.

INTOFF/INTON - the position-error integration feature of the PID control of the servomotors can be switched off, if a steady-state position error is expected, (e.g. if the robot is exerting a force on an object).

2.5.3. Location Transformations

The position and orientation of the robot tool is internally represented in VAL II by homogeneous transformations. Paul [25] gives a complete description of the theory of homogeneous transformations and their application to robotic control. In VAL II, location transformations can be translated, rotated or compounded. Using compound transformations, locations can be related to different frames of reference.

In VAL II, TOOL and BASE transformations can be specified which rotate and offset the TOOL and WORLD coordinate reference frames. The TOOL transformation, which relates the tip of the tool to the end of the robot, is used to accommodate different end-effectors. The WORLD transformation can compensate for the movement of the robot base relative to other fixed objects.

Thus, if the location transformation of an object is known relative to the robot tool, then its transformation with respect to the reference coordinate frame is given by :-

OBJECT = BASE : T6 : TOOL : OFFSET

where OBJECT is the object location w.r.t. WORLD frame
OFFSET is the object location w.r.t. TOOL frame
T6 is the compound transformation,
A1:A2:A3:A4:A5:A6,

for the robot's six links, which relates the tool to the base.

The WORLD and TOOL coordinate systems for the PUMA robot are shown in fig. 2-6 for the default values of BASE and TOOL.

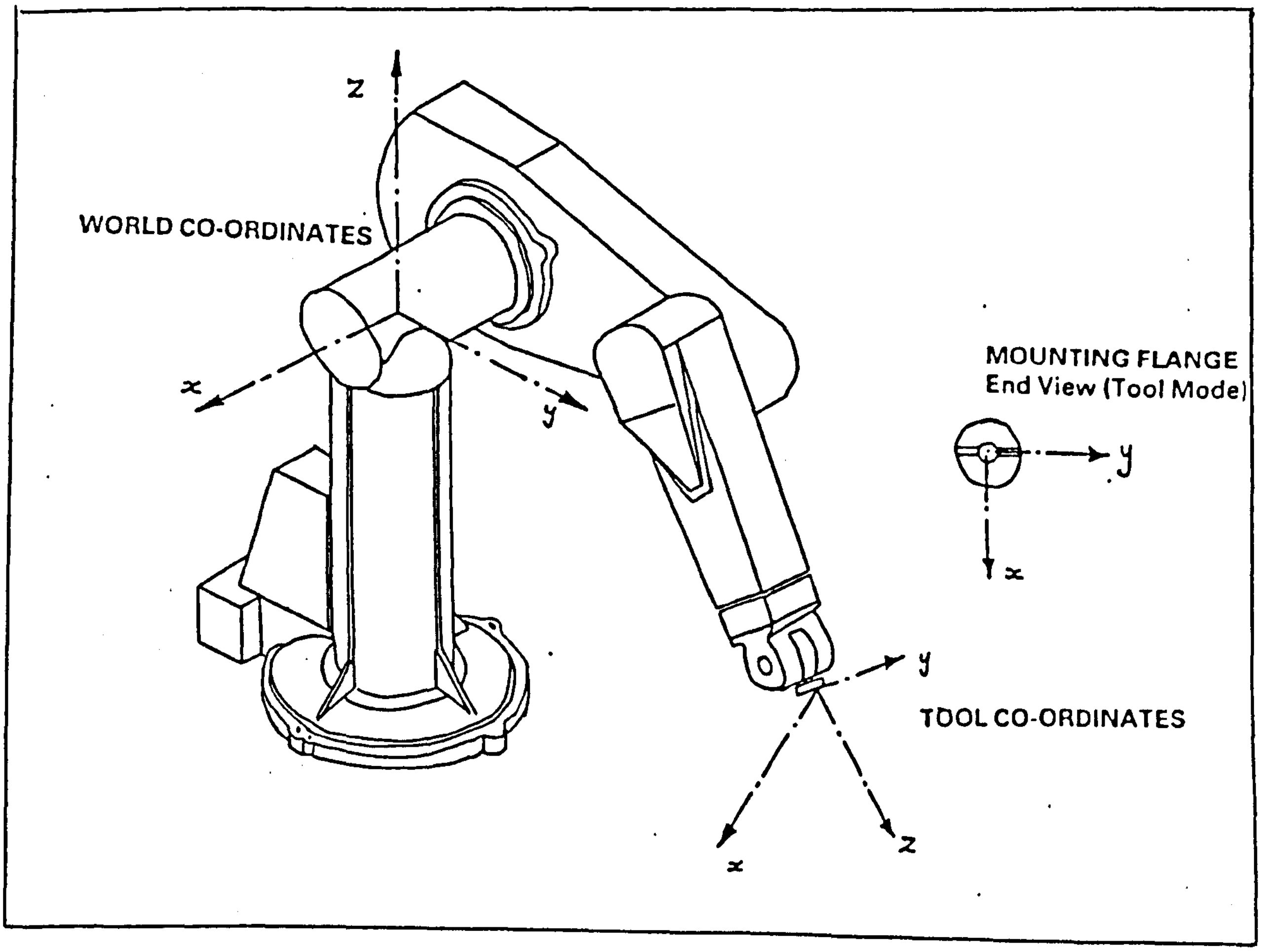


Fig. 2-6: WORLD and TOOL Coordinate Systems

2.6. General Purpose Communication (GPC) Channel

A general purpose communications channel was required between the station controller and the robot controller, in order to permit initialization, control, synchronization, parameter transfers, monitoring and error recovery.

2.6.1. VAL II Supervisory Communications Facility

VAL II provides extensive facilities for communications with a supervisory computer. The supervisory computer can monitor the VAL II system status, and perform all the I/O (input/output) that is normally performed by the VAL II terminal and disk drive. By implementing the supervisory communication channel, the terminal and disk drive can be discarded, and the robot controller remotely operated via a LAN (Local Area Network) by other controllers in the factory.

2.6.2. FIGARO GPC Design

Although, the VAL II supervisory communications facility provides all the functions that would be required by a commercial production system, it was unsuitable for research purposes. The protocol was complex and rigorous, the majority of its features were not necessary in the laboratory set-up and, furthermore the protocol used up considerable processor time and memory storage.

Therefore a communication channel was developed which provided the specific facilities required by the FIGARO system, with minimum processor overheads. In the FIGARO arrangement, the VAL II terminal was not replaced by the communication link, so that VAL II programs could be developed independently of the rest of the system.

The GPC channel consisted of 20 parallel uni-directional lines between the VAL II binary I/O signals and a 8255 PIO (programmable I/O controller) chip on a prototype card in the IBM AT bus. In each direction, 8 lines were used as a data bus (or buffer) and 2 lines were used as handshaking signals. One signal was a "Buffer Full" signal from the Sender to the Receiver, and the second was an "Acknowledge Strobe" signal from the Receiver back to the Sender.

2.6.3. FIGARO GPC Protocol

The handshaking protocol implemented in the GPC link is shown in table 2-1. The protocol was based on the timing diagram for the 8255 PIO (programmable I/O) device [46], which was configured for Mode 2 Operation (viz. Strobed Bi-directional Bus I/O). Thus, the operation of the handshaking signals was performed automatically by the 8255 PIO chip at the IBM AT end.

The 8255 PIO chip was connected to the IRQ3 and IRQ5 interrupt lines respectively and the IBM AT software implementation of the protocol was interrupt-driven so that low priority tasks were not "locked out" during GPC delays.

The software routines and the circuit diagrams developed for the GPC channel are given in Appendix C. The use of the GPC facility is further discussed in Chapter 6.

Seq	SENDER	RECEIVER
1	Put data byte on bus	
2	Set Buffer Full Signal	
3		Detect Buffer Full Signal
4		Read data byte
5		Toggle Acknowledge Strobe
6	Buffer Full Signal off	

Table 2-1: GPC Handshaking Protocol

2.7. Sewing Machine

The Mitsubishi LS2-190 lockstitch sewing machine was selected for the FIGARO development system. The machine has a conventional drop feed mechanism in which a presser foot holds the cloth against a pair of toothed dogs. The dogs pull the cloth forward intermittently in synchronization with the needle motion, so that the cloth is stationary while the needle is in the cloth.

The machine was fitted with the LIMI-STOP Z variable speed, needle-positioning clutch motor, which was controlled by the LE-MF microprocessor-based control unit. The LE-MF unit has two optional connector sockets, that facilitate interfacing the unit to an external computer, and it measures needle position and sewing speed with an optical shaft encoder mounted on the sewing head shaft.

The machine was fitted with an automatic presser foot lifter and an underbed thread trimmer which can be remotely operated to cut the sewing thread after a seam has been sewn. The machine's maximum sewing speed was 5000 rpm, and the maximum stitch length was 4 mm.

The IBM AT was interfaced to the sewing machine so that the following functions could be controlled from the station controller :-

* start and stop sewing
* vary sewing speed
* backtacking (i.e. sewing backwards)
* lift presser foot
* trim sewing thread
* stop machine with needle up or down
* bring needle up

The IBM AT/sewing machine interface is described in detail in Appendix H.

2.8. Work Station Design

At the start of the FIGARO development, the work station consisted of the robot, an end-effector, the sewing machine and a sewing table. Additional features, that were incorporated when the need arose, are described in later chapters.

2.8.1. Sewing Table

The sewing machine was mounted on a large table, 180 mm by 800 mm, with the needle located 330 mm from the end. The table's dimensions were selected so that there would be sufficient room to manipulate large cloth panels (e.g. trouser legs) for the sewing and handling operations.

The sewing table required both a smooth surface, so that cloth panels could be slid into position without buckling, and a reflective surface, so that the edge of the cloth panel could be easily detected by photocells and CCD cameras. Consequently, the sewing table was covered with a thin sheet of highly polished stainless steel, which provided an excellent reflective surface and a relatively low table-to-cloth friction.

However, the table friction proved sensitive to dust, and to combat this, the table surface required periodic cleaning. The table friction would be further reduced by incorporating a flotation system in the sewing table, which would also reduce its sensitivity to dust. Flotation, in which compressed air is expelled via small nozzles drilled in the table surface, is often employed in automatic sewing stations.

2.8.2. End-Effector

The end-effector was designed to perform sewing and handling operations on cloth panels with the simplest possible configuration and minimum interference with the sewing operation, in order to retain maximum system flexibility (section 1.5.4). The first prototype of the end-effector is shown in fig. 2-7. The second prototype is described in section 6.2.

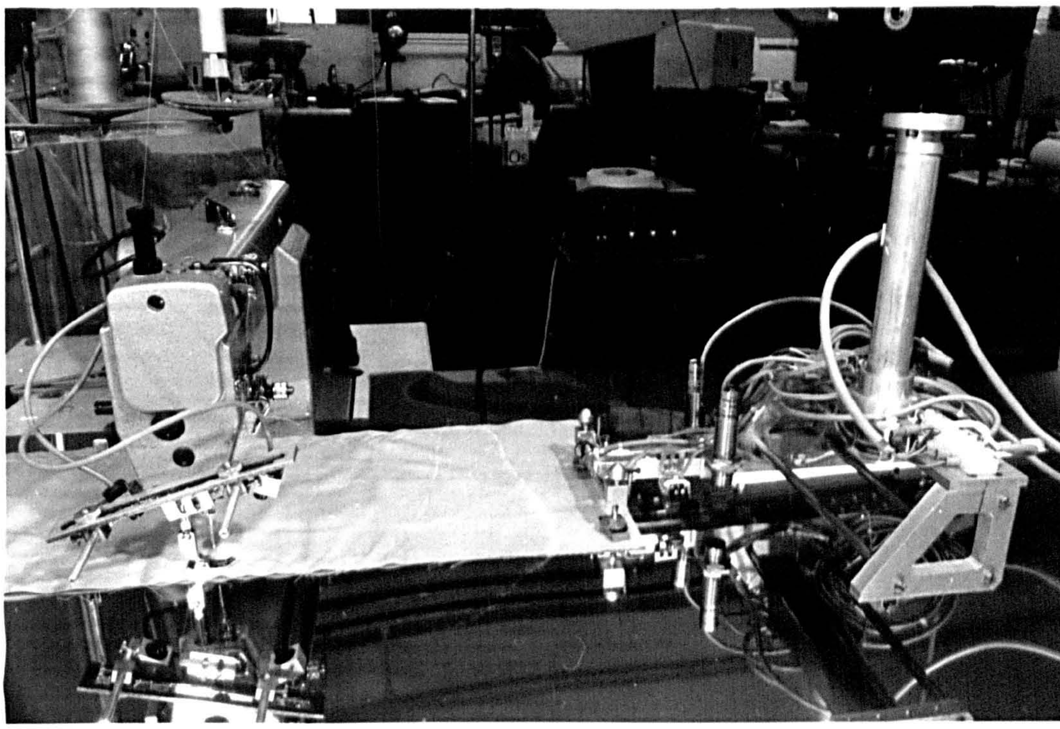


Fig. 2-7: FIGARO End-Effector - First Prototype

2.8.2.1. Number of Fingers

If one end of a cloth panel is held by the sewing machine needle, then a minimum of two fingers is required to rotate the cloth about the needle, when fingers are positioned at the far end of the cloth panel. Similarly, a minimum of two fingers is required to slide a cloth panel across the table, when fingers are positioned at the front edge. Although additional fingers reduce the cloth panel's tendency to buckle, they also restrict the working envelope of the end-effector in the vicinity of the sewing machine.

2.8.2.2. Hand Design

The first prototype end-effector had two spring-loaded fingers supported on the end of cantilevered beams. This low profile design permitted the fingers to operate in close proximity to the sewing needle and move under the arm of the sewing machine without the end-effector hitting the sewing machine.

The distance between the two fingers could be adjusted manually. Several micro-switches were installed on the end-effector to detect collisions between the robot and objects in the workspace (section 4.3.4.5). A photocell was mounted on each finger beam to detect the edge of the cloth panel (section 6.2.2).

2.8.2.3. Finger Pads

To prevent the cloth panel slipping under the fingers during handling and sewing operations, the finger-to-cloth friction had to be greater than the table-to-cloth friction and also greater than the cloth tension during sewing. Consequently, the finger pad material had to exhibit high friction with fabrics at low contact pressure. Card wire pads or needles were rejected since they would scratch the table surface. Pads with nylon needles were found to be unsatisfactory since they required relatively high spring loading before they gripped the cloth.

Rubber pads, with a diameter of 20 mm, were found to give satisfactory performance; the best performance was achieved using thin rubber discs with a contoured surface to increase surface friction.

2.8.2.4. Spring Loading of Fingers

Each finger was spring-loaded, and the finger's vertical travel had to be sufficient to accommodate static and dynamic errors in the height of the end-effector above the table. Static errors up to 10 mm were measured by programming the robot to slide slowly across the table surface; these were due to distortions in the table surface and due to the robot's poor static accuracy. When the robot was programmed to slide across the table at high acceleration and velocity, significant dynamic and inertia effects caused height variations of up to 20 mm.

The finger, its support and spring arrangement were designed to maintain a low profile while still providing 20 mm vertical travel. Various springs, with different spring constants, were tested in the end-effector (see section 4.5.4.4).

2.8.3. Robot Siting

The optimum siting of the robot in a work station is often a major difficulty, especially when the workpieces are large relative to the robot's workspace. In addition to the obvious problem of placing all necessary items within reach of the robot, there is also the need to avoid the robot's singularity regions.

2.8.3.1. Singularities

Six-degree-of-freedom robot arms have a number of singularities in their kinematics, which in practice means that a small change in Cartesian coordinates corresponds to a large change in joint angles. Singularity regions should be avoided since they result in unpredictable and erratic behaviour of the robot arm.

Each singularity is associated with one of the spatial configuration pairs, that is, the arm is at the boundary between either the RIGHTY or LEFTY, the ABOVE or BELOW, or the FLIP or NOFLIP configurations. In physical terms, a singularity occurs when an axis of one joint becomes aligned with an axis of an adjacent link.

Not all robot types suffer from this problem. If the number of joints is less than six there are no singularities, but then there are "holes" or regions within the workspace that the robot cannot reach.

For the FIGARO application, in which the robot's wrist flange was always held parallel to the table's surface, two singularity regions limited the robot's stable workspace. When the wrist flange was too far from the WORLD z axis, the upper arm and forearm approached alignment, i.e. the elbow singularity. When the wrist flange was too close to the WORLD z axis, one of the wrist singularities might be encountered. The FIGARO robot's working envelope is defined in section 5.4.2.

2.8.3.2. Robot Height

The robot was fitted to a pedestal that was 170 mm lower than the table surface. With the end-effector installed, the robot exhibited wrist singularities even when the wrist flange was quite distant from the WORLD z axis. The wrist singularities were minimized by lowering the robot base so that the arm was closer to the table surface.

The problem and its solution can be readily understood by considering the anthropomorphic analogy. If a man tried to slide the palm of his hand over a low table surface while standing up, he would strain his wrist ! However, he would be much more comfortable if he sat down at the table because he would use his elbow and shoulder more and his wrist would not be strained.

The optimum height range for the robot tool flange, that would give maximum reach and also minimize wrist singularities, was found to be between 0 and 200 mm below the centre of the base coordinate origin (assuming ABOVE configuration). Since, the table surface was 490 mm below the base origin and the end-effector was 150 mm high, the robot origin had to be lowered by 150 to 350 mm. Rather than manufacture a new pedestal, a 200 mm long aluminium spacer was made to fit between the end-effector and the tool flange (see fig. 2-7).

2.8.3.3. Limitations Due to End-effector

The second prototype end-effector (section 6.2) was 540 mm wide, and the width of the end-effector significantly limited both the robot's minimum and maximum reach.

a) Minimum Reach

When the end-effector was close to the body, the inner end of the end-effector was liable to hit the robot's trunk. This minimum reach limitation could be removed by suspending the robot from an overhead gantry, so that the robot's trunk would not intrude into the useful workspace. Although, this arrangement was not implemented, overhead mounting is a recommendation for future improvement of the FIGARO system (section 7.4.4).

b) Maximum Reach

When the arm was outstretched, it could not achieve its full mechanical potential, due to a software limitation. Location coordinates are stored internally in VAL II as 16-bit signed integers, scaled by a factor of 32. Hence the maximum distance that can be legal is only :

$$\frac{2^{16}}{2 \times 32} = 1024 \text{ mm} \qquad (2.1)$$

This corresponds approximately to the maximum reach of the PUMA. However, when locations were defined relative to the far finger on the wide end-effector, using the TOOL transformation facility, then the maximum distance was still 1024 mm, even though the arm could physically reach another 270 mm. This software limitation is not present in a more advanced version of VAL II, supplied with the Adept robot, which represents distances internally as real variables (section 7.4.1).

2.8.4. Coordinate Systems

The sewing needle was selected as the origin of the work station coordinate system and the direction of sewing was chosen as the x direction. The robot TOOL transformation was carefully defined so that its origin was at F, the centre of the right hand finger pad, and its x axis was aligned with the workstation x axis. The xy planes of both coordinate systems were defined parallel to the sewing table's surface.

The two coordinate systems are shown in fig. 2-8; the work station's axes are marked x,y,z and the TOOL's axes are marked x',y',z'.

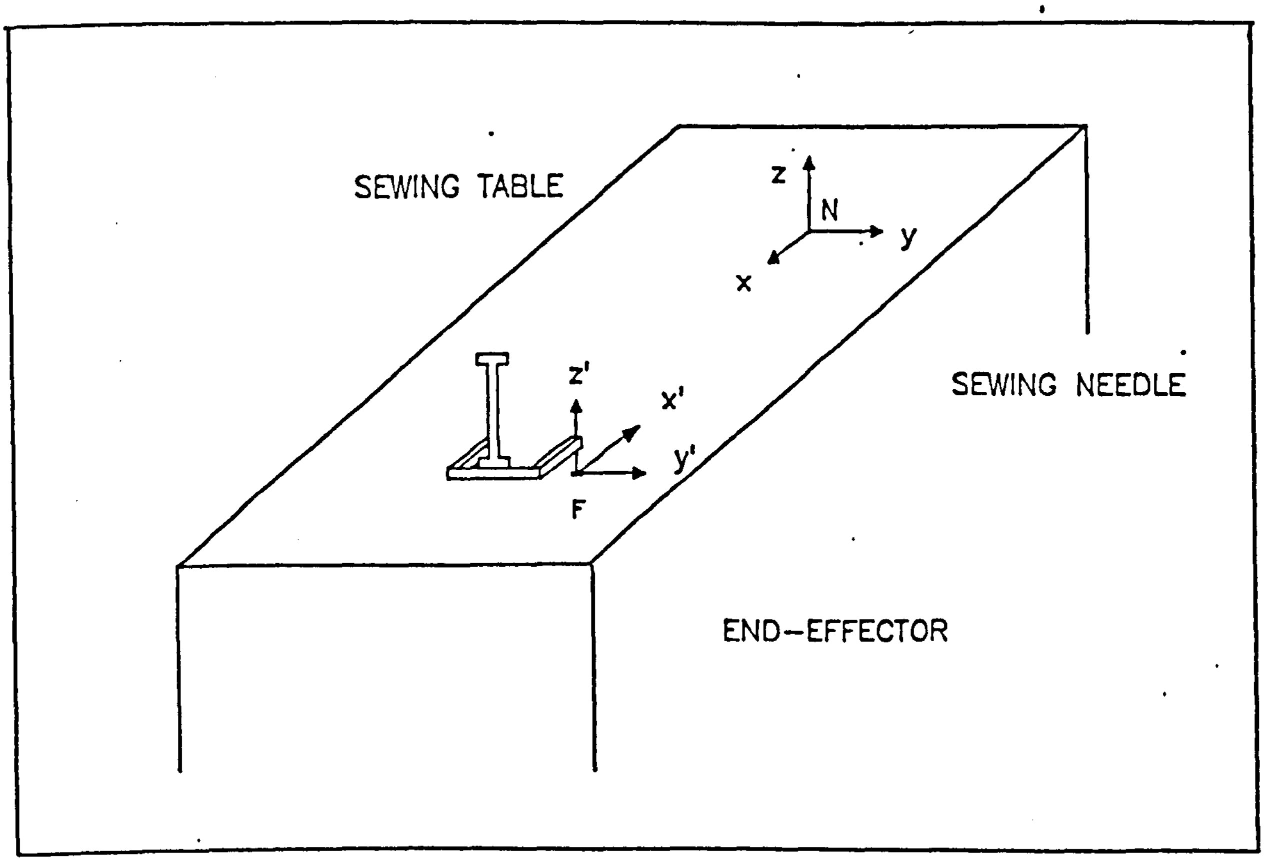


Fig. 2-8: FIGARO Coordinate Systems

CHAPTER 3

THE DEVELOPMENT OF A REAL TIME PATH CONTROL CAPABILITY

A real time path control capability was developed based on the VAL II ALTER facility, which permits an external computer to supply path modification data to the robot controller while the robot arm is in motion. A high speed serial communications link was implemented between the IBM AT and the robot controller, and interrupt-driven multi-tasking software was written to service the link at the IBM AT end.

3.1. VAL II ALTER Facility

The VAL II ALTER facility can be used to modify a pre-programmed motion or it can have total control over the robot's path. The ALTER modification data can be interpreted in TOOL or WORLD coordinates, and the robot motion can be generated by cumulative or non-cumulative application of the modification data.

3.1.1. Partial and Total Real Time Path Control Modes

Robot motion data from the external computer is ignored by the robot controller, unless VAL II is performing a programmed straight-line motion, or if the robot is stationary during a programmed DELAY.

If the robot trajectory is approximately known in advance and sensory feedback is only required to modify the tool path, then the robot should be programmed to follow the nominal path and the ALTER facility would then supply real time sensory corrections. In the case of robotic sewing, the required tool path is entirely unpredictable, and therefore it was simpler to leave the robot nominally stationary during an infinite DELAY and give the external computer exclusive control over the robot's trajectory.

3.1.2. ALTER modes

ALTER data can specify any combination of offsets along and rotations about the x, y, and z axes. When ALTER is initiated, the user must specify either the WORLD or TOOL coordinate systems for the subsequent ALTER data. He must also specify whether the effects of the ALTER data are to be cumulative or non-cumulative.

In cumulative mode, the effect of any data received is accumulated and the robot location is modified by the sum of all past ALTER data. Thus, if the IBM AT sends an ALTER value of 0.1 mm in the x direction, then the robot will move away from its nominal location at the rate of 0.1 mm per 28 ms, i.e. a speed of 3.5 mm/s (see section 3.2). The robot stops when the external computer changes the x value to zero.

In non-cumulative mode, the robot location is modified only by the most recent data. Thus, when the IBM AT sends an x value of 0.1 mm, the robot moves by 0.1 mm and then stops. When the x value is set to zero, the robot returns to the nominal location.

In the FIGARO system, ALTER was always used in the WORLD coordinate mode (section 2.5.3) for convenience. Both cumulative and non-cumulative modes were tested, and their different attributes are discussed further later.

3.2. The ALTER Communication Channel

The ALTER communication channel is dedicated to the transfer of real time path control data from the IBM AT to VAL II. The link is an RS232 serial line operating at 19200 baud, which means that a byte is transmitted every 0.5 ms. The protocol is optimized for high speed with minimal error checking and no automatic retransmission of corrupted data, since any time delay is detrimental to the performance of the path control system.

The ALTER protocol is based on a handshake cycle which is repeated every 28 ms. VAL II initiates the cycle with a short message which requests path control information and contains status information. The IBM AT must complete transmitting its reply within 16 ms of the start of the cycle, otherwise VAL II will abort ALTER with a timeout error message. Simple start and end message codes, a one byte checksum and a byte-stuffing protocol are used in the message packet.

For convenience and clarity, robot motion parameters are often quoted below in handshake units (or hs). For example, an x ALTER data value of 2, in cumulative mode, would result in a robot velocity of 2 mm/hs in the x direction (equivalent to 71 mm/s).

3.3. Implementation of the ALTER Protocol on the IBM AT.

The ALTER protocol was implemented on the IBM AT using the interrupt handling and multi-tasking facilities provided by the AMX-86 executive, in conjunction with the IBM AT serial/parallel adapter.

3.3.1. Hardware Considerations

Although the IBM technical reference manual [40] only recommends operation of their serial port at 9600 baud, when the IBM serial adapter card was installed in FIGARO's IBM AT it ran successfully at 19200 baud. However, the same card with the same software failed when used with an older IBM AT system unit. This suggests that the IBM AT may be operating close to a timing limitation when supporting interrupt-based communications at 19200 baud.

3.3.2. Software Design

The software was organized along the lines of the ISO's OSI (Open Systems Interconnection) Reference Model, which defines a hierarchy of functional levels for computer network communications [41]. The OSI model encourages a modular approach to the design of software and hardware elements. A self-contained AMX-86 task was written for each communication function within the OSI levels, in order to permit parallel execution of the functions. The hierarchical arrangement of the ALTER communication Tasks is shown in fig. 3-1.

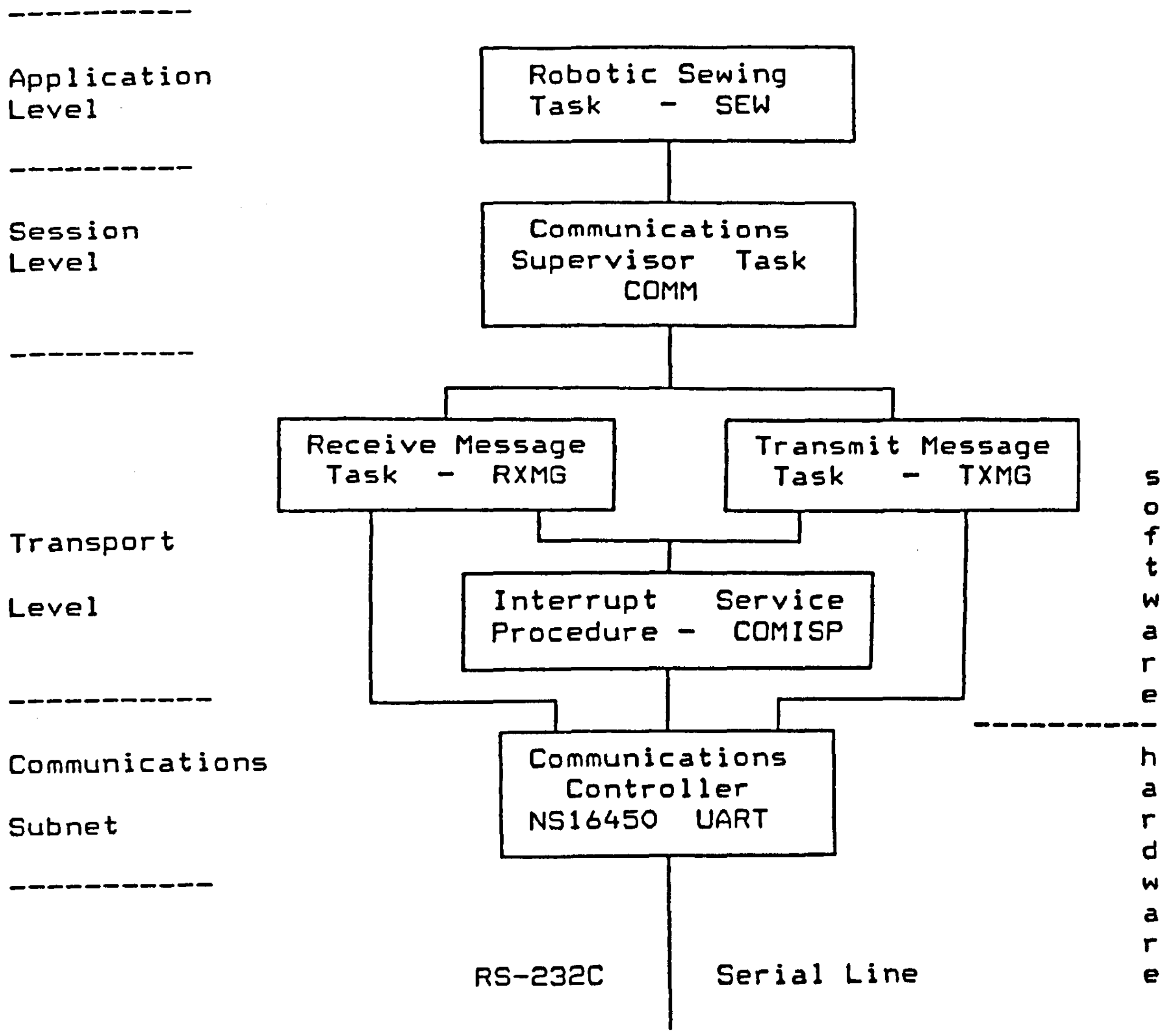


Fig. 3-1: Hierarchical Implementation of ALTER Protocol
on the IBM AT.

The SEW Task, in which the desired robot trajectory is calculated from sensory servo control functions, corresponds to the Applications Level, the highest OSI level. The SEW Task is described in later Chapters 4 and 5.

The COMM Task, which corresponds to the OSI's Session Level, performs the following functions :

* Interpreting the ALTER status message.
* Maintaining the ALTER handshake requirement by immediately acknowledging every VAL II message.
* Passing the ALTER data from the Application level on to the Transport level for transmission to VAL II.
* Terminating the ALTER communication channel.

The RXMG Task performs the following functions :

* Assembles the message packets received by the serial port.
* Removes the header and checksum and any byte stuffing.
* Checks for data corruption.
* Transfers the message to the COMM Task for interpretation.

The TXMG Task performs the following functions :

* Takes the ALTER data for transmission to VAL II.
* Constructs the message packet by adding the header and checksum and by performing the byte-stuffing protocol.
* Loads the message packet onto a circular list.

The communications ISP (Interrupt Service Procedure), COMISP, is executed whenever the UART communications controller generates an IRQ4 interrupt. The COMISP procedure determines whether the interrupt was a byte-received or byte-transmitted interrupt; in the first case it adds the received byte to a circular list, in the

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

second case it loads the port with a byte for transmission from a second circular list. If either RXMG or TXMG is waiting for an interrupt, then COMISP awakes the appropriate Task.

All the software modules associated with the ALTER communication channel are listed and explained in Appendix B. The efficiency of the COMISP procedure has a critical effect on system performance, since it is executed every 0.5 ms. Consequently, COMISP and part of TXMG were written in 8086 assembler; the remainder was written in the C programming language.

3.3.3. Communication Overhead

The support of the ALTER communication channel causes a significant processing overhead for the IBM AT. This overhead was measured by comparing the execution time of a dummy program operating in a normal MS-DOS environment, with the execution time of the same program operating under AMX-86 with the ALTER communication protocol running in the background. The following results were obtained :-

* Execution time under MS-DOS = 19.2 secs
* Execution time under AMX-86 = 27.8 secs
* During the 27.8 secs, 945 ALTER handshakes were completed. Each handshake consisted of 12 bytes received and 8 bytes sent.

Thus, the ALTER communications overheads plus the AMX-86 scheduling overheads, were :-

$$(27.8 - 19.2) / 945 = 9 \text{ ms per handshake}$$

Since the ALTER handshakes occur every 28 ms, the overheads account for a third of the cycle time, which is not very satisfactory.

A more suitable arrangement might be to implement the COMM, TXMG, RXMG and COMISP functions on a microcontroller, such as the Intel 8751, which could be installed together with a block of dual-ported RAM on a card on the IBM AT bus. During real time path control of the robot under sensory feedback, the IBM AT 80286 processor could be dedicated to calculating the robot's trajectory coordinates, while the microcontroller would take care of the ALTER communications. The received and transmitted messages would be transferred between processors via the dual-ported RAM.

Nevertheless, it was decided that the software-oriented single-processor multi-tasking arrangement was more suitable for a development system on which exploratory research was to be carried out. The hardware-oriented multiprocessor arrangement would have improved the performance of the system, but at the expense of reduced flexibility and increased complexity.

3.4. Dynamic Performance Tests on ALTER Control

3.4.1. ALTER Performance Specification

The VAL II manual [38] states that the total time taken between receiving the ALTER data from the IBM AT until the robot reaches the required location is 49 ms. This is made up of 22 ms for the matrix transformation calculations which converts coordinate data into joint angles, and 27 ms for the joint servo controllers to reach the target location.

The VAL II manual does not provide any additional information on the response performance of the ALTER motion control, or on smoothing or interpolation requirements. Initial experiments with the ALTER facility indicated that careful interpolation and limitation should be applied to the ALTER data sequence to prevent erratic or jerky motion. Series of tests were performed to confirm the timings given in the manual, and to investigate the dynamic characteristics and interpolation requirements of ALTER control.

3.4.2. Test Setup

In the test setup the PUMA was attached to the end of a vertical LVDT, with a travel of 150 mm. A timing output signal was produced by the IBM AT, which showed the beginning of the handshake cycle and the end of the IBM AT message transmission. The LVDT's output was filtered at 1 kHz, stored in a data logger and then recorded on an X-Y plotter.

Three test series were performed :-

a) single step change in position
b) ramp demand - i.e. a constant position increment per handshake
c) stepped ramp demand - i.e. every second or third handshake a position increment was transmitted, and a zero increment was transmitted on the other handshakes.

The tests were repeated for a range of increment rates up to 10 mm per handshake. In addition, many of the tests were repeated for both cumulative and non-cumulative ALTER modes, and for the COARSE, NONULL and INTOFF motion control parameter settings.

3.4.3. Results

The following test results *were obtained* :-

* The robot started to move approximately 20 ms after the IBM AT message had been sent. This confirmed the manual's timing specification for the matrix transformation calculation.

* When the robot performed a step change, over 85% of the total distance was covered within the specified 27 ms. The remainder of the distance was gradually achieved over a further 10 to 35 ms. This characteristic appears to be due to a conservative control strategy applied to the joint servomotors, involving coarse and fine motion segments.

* The robot moved very smoothly when given a ramp demand. The robot tool passed through the requested location 65 ms after the IBM transmitted the data. This figure is not quite as good as the one quoted in the manual (49.5 ms), probably due to the intentional position offset applied to the coarse motion segment.

* A stepped ramp demand resulted in an intermittent, staggered robot motion. For ALTER increments above 5 mm, the jerky motion was severe.

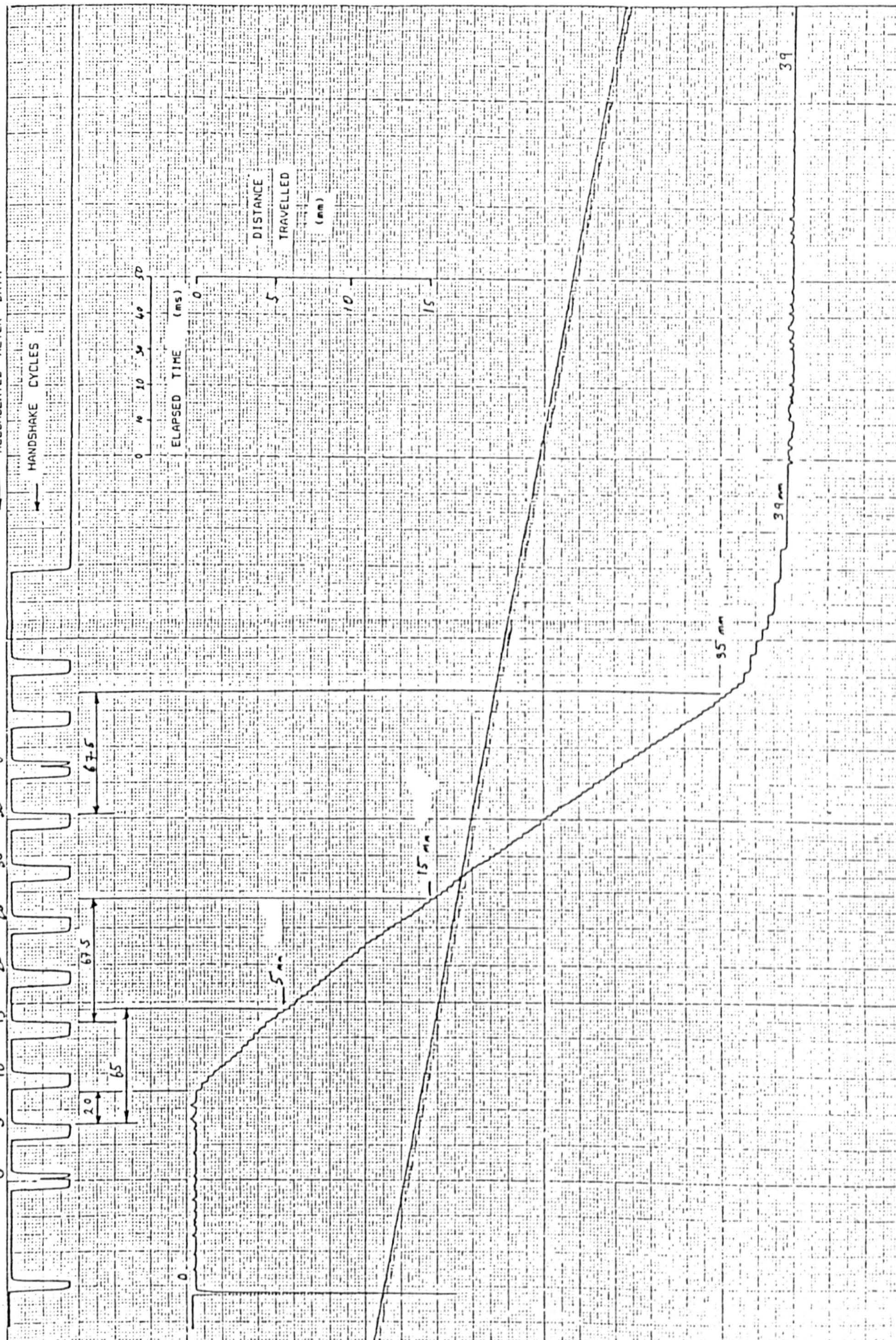


Fig. 3-2: ALTER Dynamic Test Results - Ramp Test

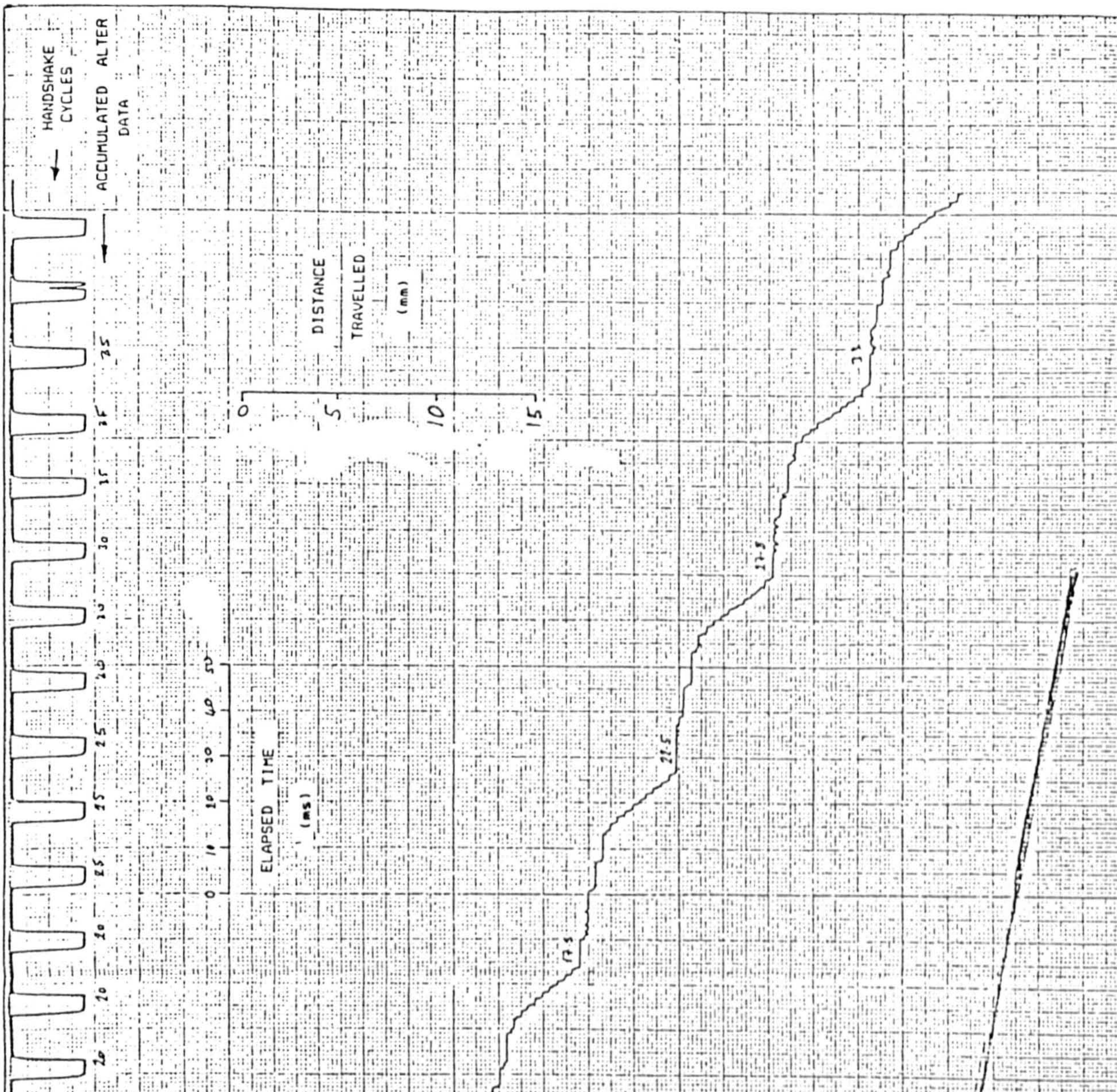


Fig. 3-3: ALTER Dynamic Test Results - Stepped Ramp Test

* Cumulative and non-cumulative modes produced identical results for equivalent tests, confirming that the two modes are provided merely for the user's convenience but they do not imply any difference in the control of the robot.

* The settings of the COARSE/FINE, NULL/NONULL and INTON/INTOFF motion control parameters have no effect on ALTER real time path control (section 2.5.1).

Two examples of ALTER motion traces are shown in figs. 3-2 and 3-3.

3.4.4. Conclusions

a) When VAL II performs a normal programmed robot motion, the LSI 11 processor applies an interpolation and smoothing function in order to achieve a specified tool velocity and smooth acceleration and deceleration. Then, the LSI 11 sends the computed setpoints to each joint controller every 28 ms.

b) When an external computer specifies the intermediate locations every 28 ms, the LSI 11 processor does not apply any smoothing interpolation; it merely converts the ALTER data into joint setpoints and sends the setpoints onto the 6502 joint controllers.

c) Consequently, the external computer is responsible for smoothing out the ALTER data to produce smooth acceleration and deceleration, and for limiting the position increments to achieve a particular speed.

d) The 6502 joint controllers perform a digital PID (proportional-integral-derivative) control algorithm which consists of a coarse and a fine motion phase. A large proportion of the demand is input into the coarse control and is achieved at high speed. The remainder of the demand is achieved more slowly and accurately using integral control. This conservative control strategy was probably adopted to prevent instability due to coupling effects between joints and load and gravity effects (section 2.4.4).

3.5. Generation of ALTER Data

As indicated by the ALTER dynamic performance experiment, care was required in the generation of ALTER data to ensure that the subsequent robot motion was smooth and approximated to the intended motion.

3.5.1. Velocity and Acceleration Limitations

Since the handshake rate is fixed at 35.7 Hz (1 every 28 ms), the magnitude of position increments that the IBM AT demands, determines the speed of the robot motion. Similarly, the rate of change of the position increments determines the robot acceleration. The ALTER data must be limited to sensible values by the external computer, since when a very large position increment was transmitted (i.e. greater than 25 mm/s) the robot arm was flung violently in an uncontrolled motion.

VAL II interprets successive ALTER position demands as point-to-point motions. Therefore, if a straight-line motion was desired, the ALTER data should be limited to small position increments, so that the gross motion would effectively be linear.

In an investigation into ALTER motion control, the robot was programmed to move in a straight line, parallel to and 100 mm above the table surface with an acceleration of only 4 mm/hs/hs (0.52 g) and a velocity of 15 mm/hs (0.42 m/sec). Instead of a linear trajectory, the robot was observed to move in a vertical circular arc such that it would have hit the table top if the intended trajectory had been within 30 mm. Further experimentation showed that the ALTER data had to be limited to within 3 mm/hs/hs and 8 mm/hs in order to maintain satisfactory linear motion.

The maximum velocity and acceleration depended on the distance of the end-effector from the robot's base. When the arm was outstretched, the dynamic errors were more severe due to the arm's reduced stiffness. Consequently, the ALTER data was limited to 1.5 mm/hs/hs and 4 mm/hs when the mounting flange on the robot's wrist was more than 680 mm from the origin of the robot's WORLD coordinates (section 5.4.2).

Additional limitations were applied to the ALTER data before transmission to VAL II, which prevented the end-effector from colliding with either the sewing machine or the base of the robot, or from approaching a singularity region. The implementation of these limitations is discussed in section 5.4.2.

3.5.2. The Non-Cumulative Approach

3.5.2.1. The Need for Smoothing

The ALTER data computed in the SEW Task was derived from the sensory servo control transfer functions. However, due to the processing limitations of the IBM AT and due to speed limitations of the vision system, the SEW Task was not able to compute new ALTER data in time for each handshake. Usually, the ALTER message would be updated only once every two handshakes, and occasionally once in three handshakes.

Initially, the ALTER channel was operated in the non-cumulative mode. When the calculation overhead reduced the ALTER update rate to less than that of the ALTER handshake rate, the robot motion was intermittent and jerky. This undesirable behaviour was due to the stepped ramp form of the ALTER data, which had been investigated in the Dynamic Response Experiment (section 3.4.3).

For example, in non-cumulative form ALTER data for a smooth robot motion between, say, locations 2 and 10 mm away from nominal origin, might be computed as :-

2 4 6 8 10

However due to the slower update rate, VAL II would receive ALTER data in the form of a stepped ramp, as:-

2 2 4 4 6 6 8 8 10 10

Consequently, the resultant robot motion would be jerky. Clearly, some form of interpolation was required to smooth out the infrequently calculated robot path increments among the more frequent ALTER handshakes.

3.5.2.2. The Interpolator Algorithm

A smoothing interpolation algorithm was written for non-cumulative ALTER data, and was executed on the COMM Task level. The algorithm modified ALTER messages that had not yet been updated, based on a prediction of the next ALTER message.

If the COMM Task received from the SEW Task a position demand of, say, 4 mm, and the previous update had been 2 mm, then the interpolater assumed that the next update would be 6 mm, by extrapolation. If an ALTER handshake requested data before a new update had been calculated, then an intermediate position demand would be transmitted, i.e. a value between 4 mm and 6 mm. For the first non-updated handshake 40 % of the increment was transmitted, and if there was a second non-updated handshake then 70 % of the increment was transmitted, and so on.

Although somewhat inelegant in concept, this algorithm was effective in smoothing out robot motions.

3.5.3. The Cumulative Approach

3.5.3.1. Implicit Interpolation

When the ALTER channel is operated in the cumulative mode, there is no need for an explicit smoothing routine, since the position increment is maintained during a non-updated handshake.

For example, in the cumulative mode, if the robot was to move smoothly between locations 2 and 10 mm from the nominal origin, then the ALTER data could be:-

2 2 2 2

Even if the update rate was slower than the handshake rate, the robot would move smoothly without requiring interpolation.

3.5.4. Comparison of Cumulative and Non-Cumulative Modes

Fundamentally, there is very little difference between the two approaches at representing ALTER data, and both were implemented successfully.

However, the software was more straightforward and more elegant when the data was expressed in the cumulative mode, and the code was marginally more efficient. The communication overhead was greater in the non-cumulative approach, since it required the smoothing routine to be called during a time-critical part of the handshake cycle, i.e. between receiving and transmitting messages.

CHAPTER 4

CLOTH TENSION CONTROL SYSTEM

4.1. Introduction

The previous two chapters described the main components of the FIGARO development system, and its real time path control capability. FIGARO was given an adaptive capability by integrating sensory-based servo control into the path control system.

4.1.1. Robotic Sewing of a Straight Seam

The first FIGARO sewing function developed was to sew a straight seam. The technique that was implemented was imitative of one of the common techniques used by human operators. Once the end of the cloth had been correctly placed under the sewing head, the robot was required to hold the far end of the cloth against a smooth table and to guide the cloth while it was being sewn up.

The sensory servo control system had to ensure that the robot tracked the forward motion of the cloth, caused by the feed mechanism of the sewing machine, and maintained a small tension on the cloth during the sewing operation. The development of this control system is described in this chapter.

4.1.2. Requirements of Cloth Feed Tracking Servo Control

The major problem in applying a robot to control cloth during a sewing operation, is the limp nature of the cloth. Cloth can buckle under small shear forces, in a manner which is usually impossible to predict. Consequently, it is essential to ensure that buckling of the cloth is kept to a minimum, and that it does not occur at all in critical areas of the cloth panel during the operation. Once buckling of the cloth has been eliminated, the cloth panel can be assumed to behave like a rigid lamina.

As described in section 2.8.1 the table had a smooth polished stainless steel surface, in order to minimize buckling. However, between the robot's fingers and the sewing head, buckling could easily occur due to forces applied to the cloth via the feed mechanism or via the fingers. This buckling could be prevented by maintaining a small cloth tension between the fingers and the sewing head during sewing, to ensure that the cloth panel stays rigid.

If there was no cloth tension, then the robot would buckle the cloth when it moved forward or when it rotated the cloth about the needle (in the edge seaming operation). If the tension was too high then the asymmetry of the tension loading on the fabric would cause the cloth end to bend upwards near the presser foot, and this would affect the accuracy of the seam width measurement. In addition, high cloth tension would lead to seam puckering.

4.2. Open Loop Control

Initially, an open loop control system was developed in which robot motion data was calculated from sewing machine speed measurements, so that the robot could track the speed variations of the sewing machine. This arrangement provided open loop control only, since the system had no feedback on the cloth tension, which was the "desired output" of the control system.

The sewing machine speed was measured from the shaft encoder signal, and the desired robot motion was calculated assuming a fixed stitch length, (i.e. the cloth moved a fixed distance per sewing machine revolution).

4.2.1. Shaft Encoder

The sewing machine control unit monitored the sewing speed and the position of the needle using an optical shaft encoder. The incremental encoder, with had an output signal of 36 CMOS square waves per revolution, had a resolution of ±5°. The encoder did not provide any directional information (the shaft is only rotated in one direction even when backtacking), but two additional signals are provided which indicate the "needle up" and "needle down" positions.

4.2.2. Shaft Encoder Interface with IBM AT

Although the Mitsubishi LE-MF control box (section 2.7) did not provide a direct interface with the shaft encoder signal, the signal was accessed by tapping it at entry into the LE-MF control box. The signal was transmitted to the IBM AT and fed into a 16-bit uni-directional counter installed on a prototype card.

A false triggering problem was traced to noise picked up by the cable, due to capacitive signal coupling [47]. The problem was solved by improving the cable shielding and by buffering the signal before transmission down the cable. Wiring and circuit diagrams are given in Appendix H.

This interface permitted the IBM AT to obtain an instantaneous reading of the number of sewing machine revolutions since the counter was reset. The shaft encoder resolution was 36 counts per revolution, and the maximum number of revolutions that could be counted before the counter overflowed was 2^{16} / 36 = 1820. The distance that the cloth is fed past the needle is related to the sewing revolutions by the stitch length setting, e.g. for a stitch length of 1 mm, the counter would overflow after a seam of 1820 mm. Since no continuous seam could be so long, a 16-bit counter was sufficient for this application. For debugging purposes, an error message was generated if the software detected counter overflow.

4.2.3. Software Implementation

4.2.3.1. SEW Task

As described in section 2.3.3.2, the sensory servo control calculations were implemented in the SEW task. This task generated ALTER data in real time on the basis of sensory inputs, to perform a contoured seam. The SEW task assumed that the front end of the cloth had been accurately placed under the needle and that the robot fingers were in place at the far end of the cloth. The basic SEW algorithm was as follows :-

1. Perform initializations
2. Start sewing
3. Calculate ALTER data for correcting in X direction
5. Install ALTER data in new message for COMM to transmit
6. Check if end of seam length has been reached
7. If not yet, then repeat steps 3 to 7
8. Stop sewing machine

4.2.3.2. Implementing Open Loop Control

The 16-bit counter, which counted the square wave signal of the shaft encoder, was reset to zero during SEW'S initialization phase. Consequently, the value of the counter during sewing always indicated the number of sewing machine revolutions since the start of that sewing operation. The length of cloth fed into the sewing machine since the start of the sewing operation could be estimated using the following relationship :-

$$L = \frac{C\ S}{f} \qquad (4.1)$$

where : L is the length of cloth fed so far

C is the count so far

S is the average stitch length

f is the frequency of counts per rev (viz. 36)

The ALTER facility was used in the cumulative mode (section 3.5.4); in this mode the ALTER data is required in terms of position increments (i.e. a velocity demand). The shaft encoder counter was sampled at the update rate, which was usually slower than the handshake rate (section 5.4.3). Consequently, the ALTER data value, x_{ALTER}, was set equal to the cloth feed speed in mm/hs, as follows :-

$$x_{ALTER} = \delta L \; u = \frac{\delta C \; S \; u}{f} \qquad (4.2)$$

where : δL is the increase in L since last update

δC is the increase in C since last update

u is the average update rate

(i.e. no. of updates/no. of handshakes)

4.2.4. Open Loop Control Performance

The stitch length can be manually adjusted on the sewing machine by rotating a knob which alters the stroke of the feed dogs. When the stitch length was set to a nominal value in the software, the robot speed and the cloth feed speed could be synchronized manually using the knob. If the stitch length was too large, the robot lagged behind the cloth feed and the cloth tension was too high. Conversely, when the stitch length was too small, the robot preceded the cloth feed and the cloth went slack and buckled.

When sewing a straight seam, an optimum knob position could be found for that particular fabric type at a particular speed, which gave a stable cloth tension. The optimum knob position varied for different fabric types and for different speeds. Consequently, the open loop control required manual adjustment when changing the fabric material.

When sewing an edge seam, the robot was required to rotate the cloth about the needle (under the seam tracking servo control), and the behaviour of the cloth panel within the feed mechanism was unpredictable. The open loop control system failed to maintain a constant cloth tension during an edge seam operation.

4.2.5. Limitations of Open Loop Control

The unpredictable behaviour of cloth tension during sewing was caused by slipping between the feed dogs and the cloth. During the feed part of the sewing cycle, the cloth is clamped between the presser foot and the dogs. The dogs grip the cloth with their serrated faces, but some slipping still occurs at the beginning and end of the feeding phase, and when there is a rotating moment on the cloth about the dogs.

Different fabrics required different stitch length settings, since some were more prone to slipping than others. By adjusting the stitch length manually, it compensated for the average rate of slipping during the sewing operation.

A suggestion for improving the open loop control was considered, that would involve inserting a constant force spring between the finger and the robot hand; this would accommodate small errors between the robot speed and the cloth feed speed. However, this modification was rejected on the basis that even a small tracking error would require the spring to absorb tension errors cumulatively, and the spring would then soon use up its total displacement length.

The open loop control of the cloth tension during sewing was unsatisfactory, since the cloth tension variations were unpredictable and they could not be compensated for adequately. Evidently, it was necessary to measure the cloth tension during sewing, and to close the loop by feeding back this measurement into the control system.

4.3. Cloth Tension Sensor

4.3.1. Measuring Cloth Tension

In order to measure the cloth tension, a sensor was required which measured the force acting on the robot finger pad from the cloth tension. If the finger held the cloth against a table, the actual tension in the cloth would not be the same as the tension sensed by the finger pad, due to the friction between the table and the cloth. The friction problem could be avoided by holding the cloth end in the air between clamped finger pads.

Human operators sometimes hold the cloth end in the air during long seam sewing operations, but they use this technique because it ensures that both plies will be the same length after sewing. However, since the operator must hold the cloth at its end, this technique is limited to sewing gently curved seams only. For the majority of operations, the human operator holds the cloth down on the table, since this permits greater manipulative flexibility.

Both techniques are useful in different circumstances, but the cloth-held-against-table technique has wider applicability and does not require the end of the cloth to be picked up first. It was decided to attempt the development of a cloth tension servo with the cloth held against the table. If the table friction problem could be solved then the control could be readily adapted for use with the cloth-in-the-air technique, which avoids the friction problem, altogether.

4.3.2. Sensor Specification

The cloth tension sensor and its signal processing circuitry was designed to meet the following specifications :-

High Sensitivity - the optimum cloth tension during sewing is between 0.25 to 1.0 N/cm. For a 2 cm wide finger pad with a spring loading of 4 N, the friction acting between the table surface and the cloth is approximately 0.5 N. Therefore, a sensor, based on a 2 cm finger pad, should have good resolution in the 0 to 1.5 N range.

Measurement Range - a full scale deflection of 4 N would be sufficient.

Low Hysteresis - although the table friction had already introduced significant hysteresis, the finger/sensor arrangement should not add to the problem.

Accuracy - the linearity and repeatability requirements are not very stringent in this application, since there is a range in which the cloth tension is permitted to vary.

Cross-Sensitivity - the sensor should be mechanically decoupled, i.e. it should be sensitive to force in the desired direction and insensitive to any other forces or moments. If the sensor was not mechanically decoupled then the output signal would be dependent on factors other than the cloth tension, such as the finger spring loading.

Bandwidth - As described later, the cloth tension was found to fluctuate smoothly in synchronization with the sewing speed, and the maximum sewing speed is 5500 rpm (92 Hz). Therefore the sensor's bandwidth should be at least 1 kHz.

Drift - since the tension control is only active during short sewing operations, drift and other offset effects can be nulled in the software before each operation, and therefore long-term drift is not a significant problem.

Natural Frequency - the sensor's natural frequency of vibration should be considerably higher than the servo bandwidth (which has a maximum of 35 Hz), to prevent instability. A high natural frequency and stiffness are desirable in order to minimize noise from sympathetic oscillations.

Dimensions and Robustness - since the sensor is to be fitted on the end of a robot finger, it should be small, light and sturdy with a high overload capacity.

4.3.3. Choice of Transducer

Usually, a force sensor consists of an elastic body which deforms under the applied force. Measurement of the elastic deformation, in one or more directions, by an appropriate transducer yields electrical signals from which the force vector can be derived. Several measuring principles are suitable such as, displacement transducers (LVDT, inductive, capacitive), piezo-electric crystals, magneto-elastic devices, conductive rubber, strain gauges, etc.

A wide variety of transducers have been developed for robotic tactile sensing, i.e. the measurement of the variation of contact forces over an area [46,47]. However, strain gauges are by far the most popular transducer for robot force and torque sensors, since they are small, easy to use, cheap and reliable [48].

4.3.4. Mechanical Design

4.3.4.1. Mechanically Decoupled Force Sensors

Several instrumented wrists and fingers have been developed for robots, that measure the three forces and three torques that describe the interaction of the robot gripper with the environment [48,49,50,51,52]. All of these sensor designs were intended to be mechanically decoupled, so that each force or torque could be obtained directly from one or two strain gauge signals.

Feldmann [51] found that his design had poor decoupling, and he had to apply a decoupling matrix to the strain gauge signal measurements in order to extract the required force and torque components. A comparison of Feldmann's design with other wrist sensor designs [48,49,50,52], which exhibited good mechanical decoupling, indicated the probable reason for his sensor's poor decoupling performance. In his design one cantilevered beam was used to measure each force component, whereas the other designs all used two beams per force component.

4.3.4.2. Force Measurement Considerations

When a cantilevered beam is loaded at its free end, the top surface of the beam will be under tension and the bottom surface will be under compression. The bending moment and surface stress acting on the beam at a particular distance from the free end, is given by the following equations :-

$$M = F x \tag{4.3}$$

$$\sigma = \frac{M c}{I} \tag{4.4}$$

where M - bending moment at x
F - load on beam's free end
x - distance from the free end
σ - surface stress
c - distance of surface from neutral axis
I - moment of inertia

For a simple beam, the neutral axis is in the centre of the beam, and therefore the surface stress due to pure bending will be equal and opposite on the top and bottom surfaces. The maximum bending moment (and therefore the maximum surface stress) is at the fixed end.

Consequently, maximum sensor sensitivity is obtained by bonding a strain gauge (SG) on both sides of the beam, close to the fixed end. When the two gauges are installed in the Wheatstone bridge arrangement shown in fig. 4-1, the output signal, V_o, is proportional to the applied load, F.

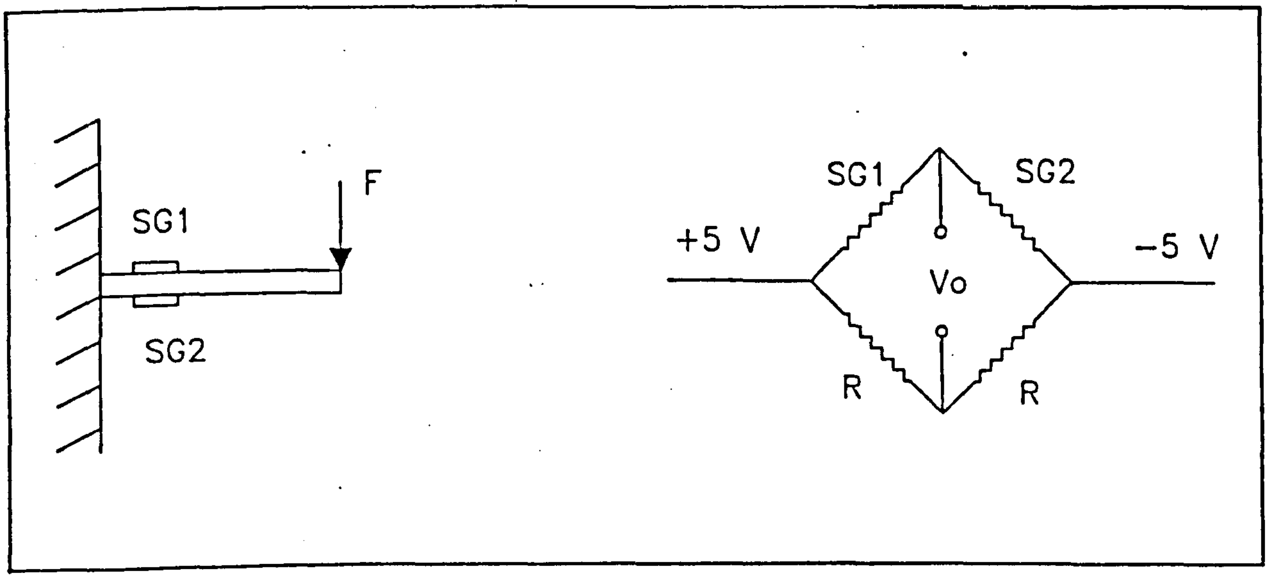


Fig. 4-1: Single Cantilever Sensor Design

If a pure compressive or tensile load is applied to the beam longitudinally, then both gauges will sense equal strains of the same sign, and the bridge arrangement will cancel out these strains. Thus, the sensor in fig. 4-1 is sensitive to lateral loads which produce pure bending, and is insensitive to longitudinal loads which produce pure tension or compression. This arrangement also provides automatic temperature compensation.

However, a compressive longitudinal load on the beam's free end may cause the beam to buckle and then the sensor would measure an apparent bending load. The double cantilever design (fig. 4-2), increases the stiffness of the sensor in the longitudinal direction, effectively decoupling the sensor. The sensitivity of the output signal is unaffected since a full bridge of strain gauges has been used in this sensor. The double cantilever design also exhibits a much higher natural frequency than the single beam design.

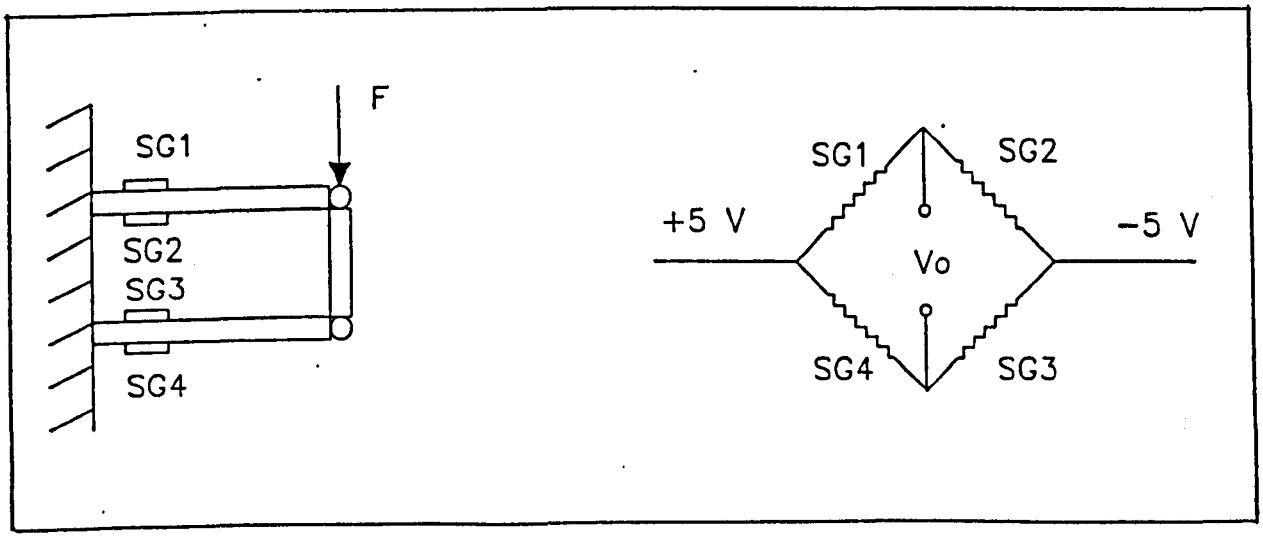


Fig. 4-2: Double Cantilever Sensor Design

4.3.4.3. Choice of Material

In order to make a sturdy sensor with high sensitivity, the sensor's material had to exhibit high tensile and yield strengths, and a low modulus of elasticity (i.e. high strains for small stresses).

High strength aluminium alloys, such as Al 2014 which was developed for aerospace applications, are usually chosen for robotic instrumented fingers and wrists [48,49,50,52]. They exhibit low modulus of elasticity and high tensile and yield strengths. High carbon spring steel exhibits greater strength, however it is more difficult to machine and also requires heat treatment after machining. Furthermore, since steel has a larger modulus, the beams would have to be thinner to provide the same output signal.

The FIGARO tension sensor was made from a square bar of Al 2014 (BS L168.T6511).

4.3.4.4. General Design

The design concept is shown in fig. 4-3 and a photograph of the actual sensor is shown in fig. 4-4.

The sensor consisted of two slender parallel beams which were machined out of a monolithic block of high strength aluminium alloy. One end of each beam was notched, so that the beam was effectively pivoted at that end.

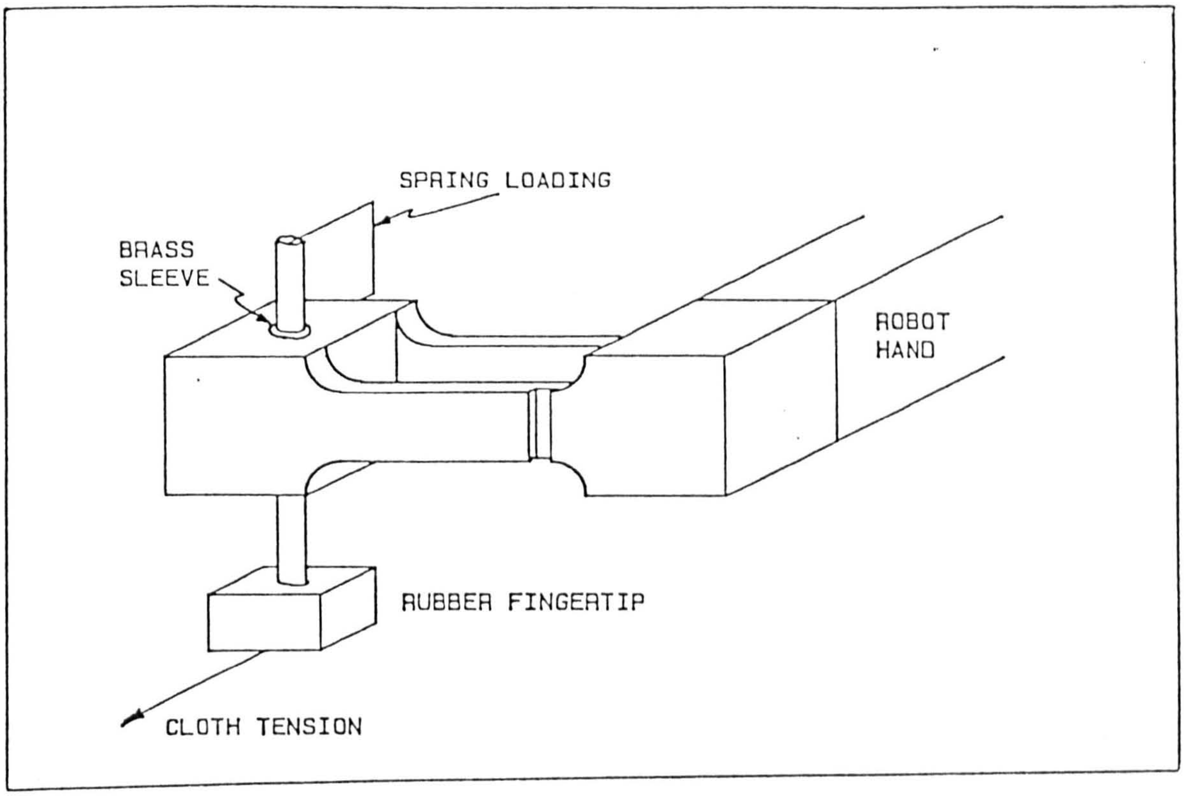


Fig. 4-3: Cloth Tension Sensor - Design Concept

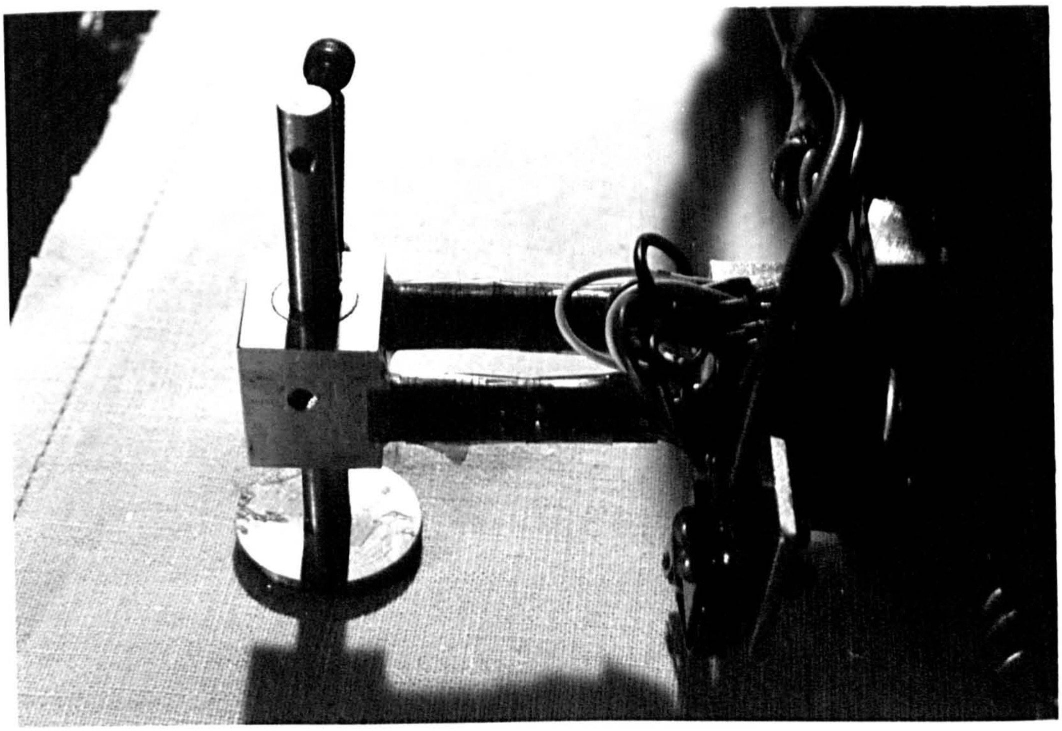


Fig. 4-4: Cloth Tension Sensor - Realization

4.3.4.5. Design Calculations

Using equation 4.4, the strain *on the top surface of a* beam, is given by the following relationship:-

$$\epsilon = \frac{\sigma}{E} = \frac{F c x}{I E} = \frac{F d x}{2 I E} \qquad (4.5)$$

where E - modulus of elasticity
d - beam thickness
ϵ - surface strain at x

The total elongation of the top surface is the integral of the strain over the total length :-

$$e = \int \epsilon \ dx = \int \frac{F d x}{2 I E} dx \qquad (4.6)$$

For a beam with rectangular cross-section,

$$I = \frac{b \ d^3}{12} \qquad (4.7)$$

where b - beam width

Substituting into (4.6),

$$e = \frac{6 F}{E b d^2} \int x \ dx \qquad (4.8)$$

The strain gauge measures the strain over its effective length only, and therefore the measured strain is the integral over the length of beam covered by the strain gauge.

Thus,

$$e_g = \frac{3\,F}{E\,b\,d^2}\,(\,x_2^{\,2} - x_1^{\,2}\,) \qquad (4.9)$$

where x_1 - distance of near edge of gauge to free end

x_2 - distance of far edge of gauge to free end

e_g - extension of gauge (strain measured by gauge)

The output signal of the strain gauge is dependent on a strain gauge factor, k, which is defined as,

$$k = \frac{\delta R}{R\ \epsilon} = \frac{\delta R\ l_g}{R\ e_g} \qquad (4.10)$$

where R - gauge resistance

δR - change in resistance

l_g - gauge effective length

The output voltage signal, v, due to each strain gauge is,

$$v = \frac{\delta R\ V}{R} = \frac{k\ V\ e_g}{l_g} \qquad (4.11)$$

where V - voltage applied to each strain gauge

The full-bridge arrangement of four strain gauges in the sensor produces an output signal four times that of an individual gauge. Thus, the output signal, v_o, is given by,

$$v_o = \frac{4\ k\ V\ e_a}{l_a}$$

$$= \frac{12\ k\ V\ F}{l_a\ E\ b\ d^2}\ (\ x_2{}^2 - x_1{}^2\ ) \qquad (4.12)$$

4.3.4.6. Detailed Design

Although, MacCarthy [56] presents an optimization design procedure for strain gauge transducers, a simpler direct calculation was sufficient in this case. The design of the tension sensor was based on equation (4.12), and on the dimensions of a suitable foil strain gauge.

A single element, constantan on polyimide, foil strain gauge (BLH SR-4 FAE-25-35 S13) was selected, which had the following specifications :

* gauge length	l_a	6.35	mm
* resistance	R	350 ± 0.5	Ω
* gauge factor	k	2.04 ± 1%	
* overall length		13.92	mm
* overall width		6.35	mm

The gauge was bonded using BLH EPY-150 strain gauge adhesive, and a 12-hour curing cycle at 35 °C. The gauge dimensions permit measurement of the surface strain over 6.35 mm of the total length, starting 3 mm from the fixed end. Thus, in equation 4.12, the following substitutions can be made,

$$x_2 = L - 3.00 \quad (mm)$$
$$x_1 = L - 9.35 \quad (mm)$$

where L - beam length

A voltage of 10 VDC was applied to a strain gauge pair, which provided a large output signal without causing any local heating effects, (the current in each strain gauge is 14 mA).

The choice of the length, width and thickness of the beams was made on the basis of equation (4.12), in order to ensure an adequate output signal level within the expected load range.

The cloth tension sensor was manufactured to the following dimensions,

$$L = 25 \text{ mm}$$
$$b = 7 \text{ mm}$$
$$d = 1 \text{ mm}$$

When the above figures were substituted into equation (4.12), the nominal signal output for a 1 N load was calculated to be 19 mV. This is a typical output level for sensors based on foil strain gauges [54].

4.3.4.7. Mechanical Overload Protection

Although the sensors performed satisfactorily throughout the FIGARO development project, the mechanical design was lacking in one respect; the sensor was very fragile, and even a slight knock could break it. When programming a robot to move in a crowded environment, it is very easy to mistakenly direct the end-effector into objects. By nature, sensitive force sensors are delicate, but industrial designs should include mechanical end-stops to prevent mechanical overload.

Although hard end-stops were not incorporated into the FIGARO sensor, two other precautionary measures were taken; micro-switches were installed which switched off the power to the robot arm when it approached too close too close to an object, and an electrical overload circuit was installed which switched off the robot when the sensor output rose beyond a certain level (see section 4.3.5.2).

4.3.5. Electrical Design

A Wheatstone bridge of strain gauges provides a low output signal with a low source impedance. The signal requires high amplification and is highly susceptible to noise and interference.

The circuit diagrams of the amplifier unit and power supplies are given in Appendix H.

4.3.5.1. Noise Prevention

In accordance with recommended practice [55], the following measures were implemented to ensure minimal noise in the amplified signal :-

a) The bridge was supplied with a regulated split-supply

(±5 VDC), with a high CMRR (common mode rejection ratio).

b) The AD524 instrumentation IC amplifier was selected, which provides a gain of 1000 with high CMRR, low drift and high accuracy.

c) The amplifier and associated components were installed on a card in a grounded metal case, mounted on the base of the robot. This location was the closest possible to the sensor, without being mounted on the robot itself. The amplifier unit was not mounted on the robot, since the robot vibrations might have affected the potentiometer settings.

d) High frequency pickup was reduced by connecting decoupling capacitors to the supply lines close to the sensor. All cables shields were grounded at one end.

e) The sense and reference terminals provided by the AD524 were used to prevent signal losses in the wiring.

f) The regulated power supplies were situated in a separate box adjacent to the amplifier unit.

4.3.5.2. Electrical Overload Protection

A safety measure was included that sent an "Emergency Stop" signal to the robot whenever the tension sensor was overloaded. This measure reduced the possibility of the robot damaging the sensor when programmed incorrectly. The tension sensor signal was passed through a window comparator, which raised the "Emergency Stop" line when the signal moved out of the window.

This overload protection circuit was originally located in the amplifier unit. However, the proximity of the comparators to a pre-amplifier bridge-balancing potentiometer gave rise to noise and oscillation problems. These problems were solved by relocating the overload circuitry on a prototype board in the IBM AT. The circuit diagram is included in Appendix H.

4.3.6. Sensor Performance

4.3.6.1. Sensitivity

The instrumented finger was calibrated in all directions by placing small weights on the free end of the sensor, and the results are shown in fig. 4-5. In the major direction (X), the sensor was found to have a sensitivity of 1.27 mV/N and a repeatability of ± 0.7 % or ± 0.003 mV; hysteresis was negligible.

4.3.6.2. Cross-sensitivity

The y and z cross-sensitivities were 0.027 mV/N, or 2 % of the normal sensitivity (fig. 4-5). Van Brussel reported a 3 % cross-sensitivity error for his 6-component force-torque wrist sensor [48].

When the finger pressed against the table, it had a maximum spring loading of 500 gf (i.e. in the z direction), and this gave rise to an error of 0.10 mV (or 8 gf). Since this was a small and fairly constant error during sewing, it was not considered a significant error.

However, if the finger was not accurately orientated perpendicular to the table, then as the robot pushed the finger against the table, it exerted a load on the finger in the x direction, and the sensor registered an apparent tension. Consequently, care was taken to assure that the finger was orientated perpendicular to the table during sewing to minimize this error.

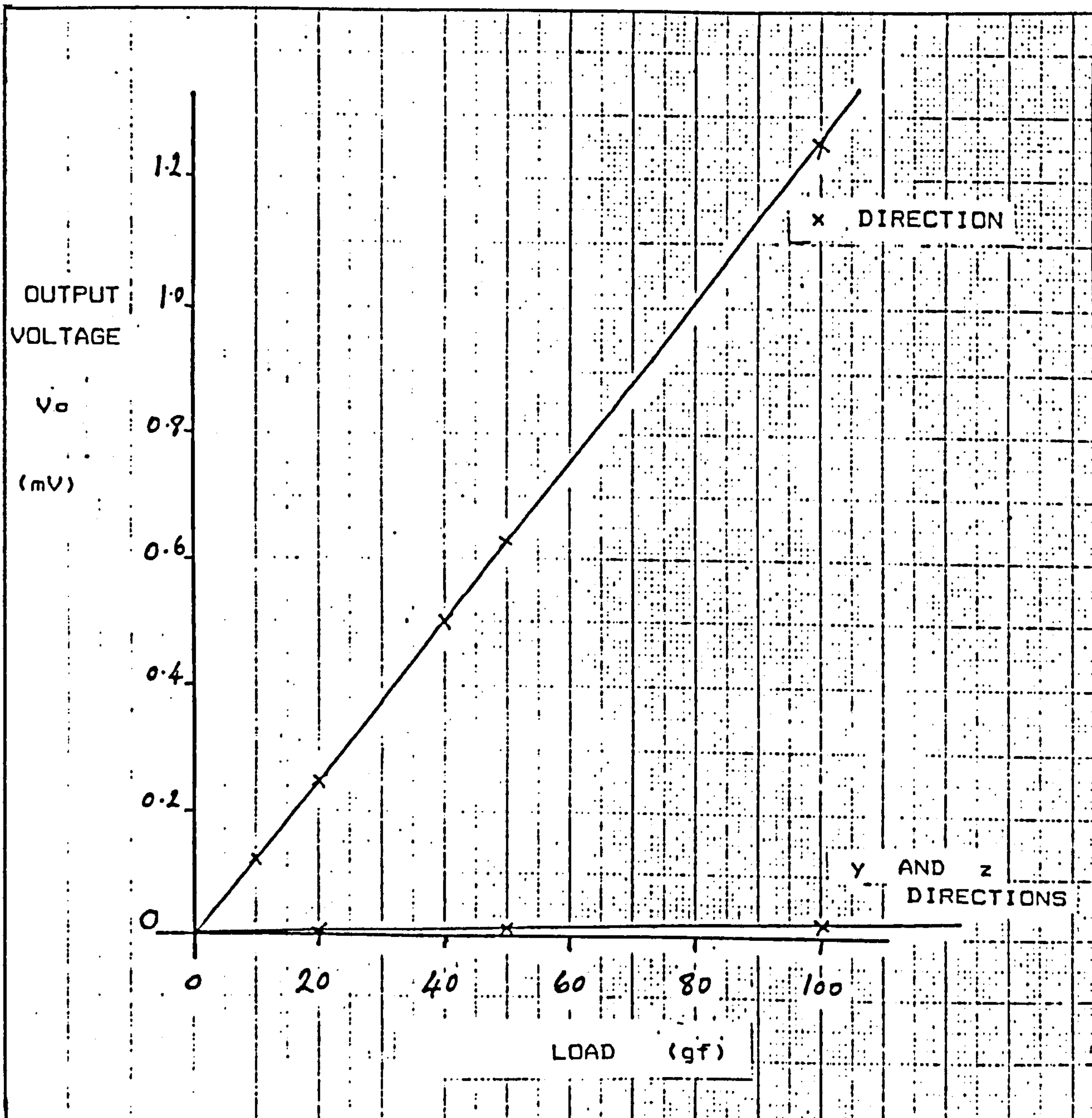


Fig. 4-5: Measured Sensitivity of Tension Sensor

4.3.6.3. Natural Frequency

The sensor's natural frequency, which was measured by "flicking" the finger and recording the signal trace, was found to be approximately 200 Hz. This fairly low value is inevitable in designs in which a lumped mass is attached to the main body by slender elastic beams. Van Brussel reported a natural frequency of 296 Hz for his sensor [48].

When the end-effector was not in contact with the table, oscillations of up to 0.2 mV were observed in the tension sensor signal, which were due to the sympathetic vibration of the sensor with the robot motion. When the finger was stationary and pressed against the table, it would pick up the table vibrations due to the sewing machine, and at high speed the amplitude of this noise signal was considerable (up to 1.2 mV). This high noise level was caused by vibration of the polished stainless steel cover which was loosely placed on the table top.

However, when the finger moved with the cloth as it was being sewn, the sensor signal was smooth and noise-free, since the cloth tension damped out the influence of the table vibrations.

4.3.7. Signal Conditioning

4.3.7.1. Signal Conditioning Requirements

The sensor's raw signal was viewed on an oscilloscope, whilst the robot was holding the cloth, and tracking the feed speed using open loop control. The signal had a smooth sinusoidal form and its frequency was proportional to the sewing speed. It was obvious that the intermittent nature of the dog feed mechanism was giving rise to this periodic variation in the cloth tension.

Since the signal had a smooth wave form, no filtering of the signal was required. However, the tension control could not use the raw tension signal directly, since digital control systems operate only on intermittent samples of the inputs, and the sampling rate is independent of the oscillation of the tension signal.

Consequently, a peak detector and an Analog to Digital Converter (ADC) were required to interface between the IBM AT and the tension signal, so that the IBM AT could read the maximum tension signal that had occurred since the previous sample.

4.3.7.2. Peak Detector

A purely analog peak detector circuit could be designed for the sensor signal, based on 2 op-amps, a FET switch and a diode. These analog circuits require a compromise between accuracy and bandwidth [55], and they are therefore optimized for a specific frequency range. However, since the sewing machine could be operated for a wide range of speeds, a digital peak detector was implemented because there would be no drift of the peak reading even for very slow sampling rates.

The digital peak detector was incorporated within the ADC circuit, and the detailed design is described in section 4.3.7.4.

4.3.7.3. Analog to Digital Converter

An 8-bit resolution was considered sufficient for the ADC, because the ADC's sensitivity could be easily adjusted, and the measurement range could be centred on the desired tension. If the control system is well behaved then it should suffice with a fairly narrow measurement range about the reference level.

The ADC was sensitive only to positive signals, so that any negative sensor signal would read as zero. Any signal above the full scale setting would read as 255 tension units. Thus the cloth tension could only be measured within a range of 0 to 255 tension units.

For convenience, tension units are abbreviated to tu throughout the remainder of this thesis.

4.3.7.4. Detailed Design

The circuit diagram of the ADC and peak detector is shown in fig. 4-6.

The tension sensor's signal is fed into a comparator, IC1, which compares it with the output of an 8-bit DAC (digital to analog converter), IC2. The DAC's output is determined by a binary ripple counter, IC3, which is clocked at 0.89 MHz. The counter counts clock pulses until the comparator detects that the DAC's output is greater than the tension signal; the comparator then switches off the clock via a NAND gate, IC4. This arrangement of a counter, a DAC, a clock and a comparator is based on the "single-slope integration" technique of analog to digital conversion [55].

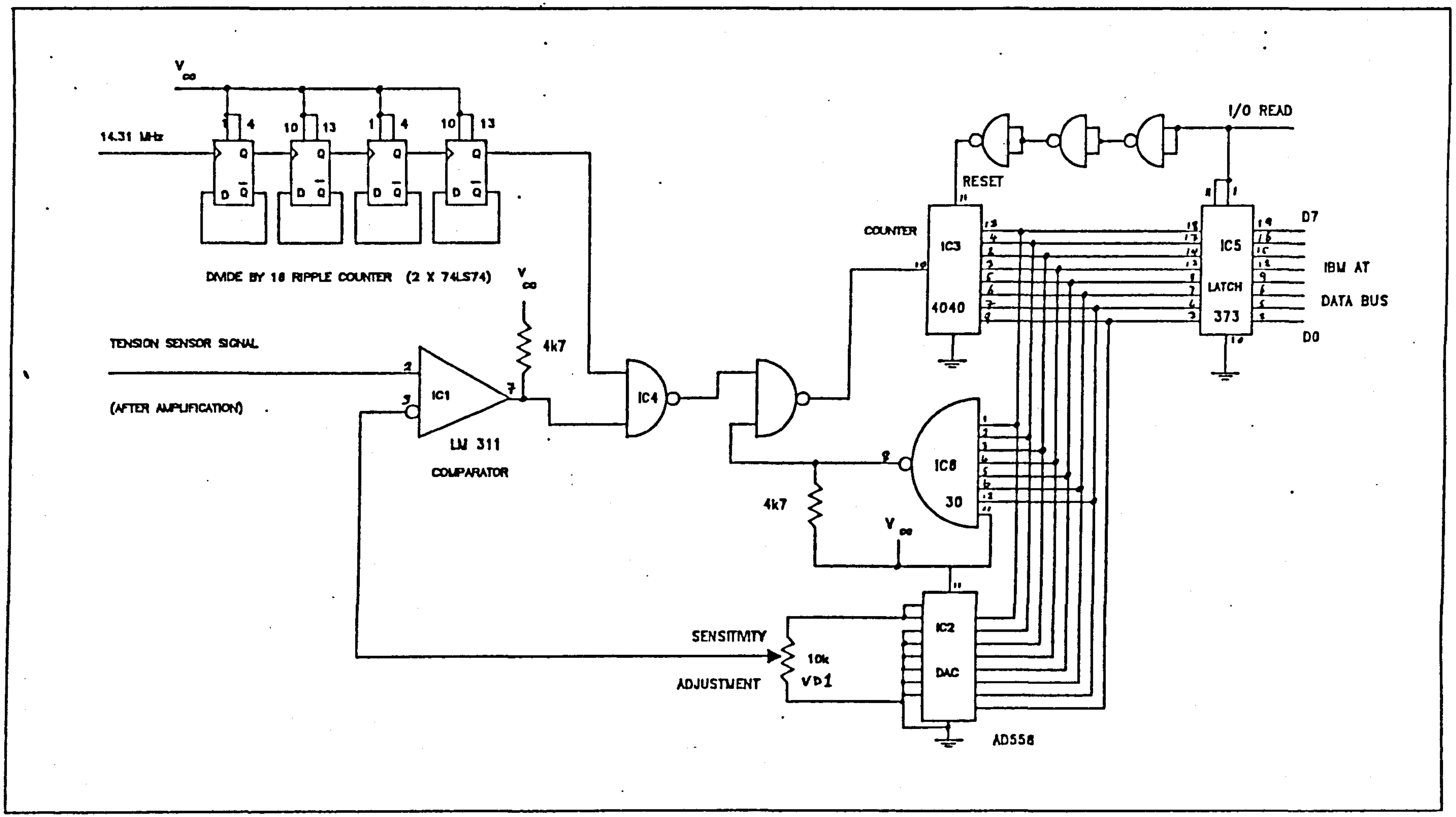


Fig. 4-6: Peak Detector/ADC Circuit

Conversion begins when the latch IC5 is read. The I/O READ line, after a small propagation delay, resets the counter to zero, the DAC's output reverts to zero and the comparator releases the clock signal to the counter. The counting is stopped either by the comparator, when the tension signal has been equalled, or by IC6 which detects counter overflow. The counter's output is frozen until either the tension signal goes higher, or the counter is reset.

The latch IC5 tracks the output of the counter, so that it will contain a digital value proportional to the maximum tension since the last time it was read. The small propagation delay ensures that the conversion cycle begins only after the previous peak tension measurement has been read into the IBM AT. The 0.89 MHz clock signal is obtained from the IBM AT 14.31 MHz system clock, via a "divide by 16" circuit constructed from four flip flops arranged in a ripple counter configuration.

With a 0.89 MHz clock and an 8-bit counter, the maximum conversion cycle time for the peak detector/ADC described above is 255 / 0.89 = 286.5 µs. Since the tension signal was observed to be a smooth signal which oscillated at the sewing machine's frequency, this conversion rate was satisfactory for tracking the peak tension, (the maximum sewing frequency was about 80 Hz).

4.3.7.5. Sensitivity

The voltage divider VD1 provided a sensitivity control so that the full scale of the ADC could be set. When the voltage divider restricted the DAC's output to a range of 0 to 5 V, the ADC would register a full scale reading (255 tu) for any tension signal above 5 V. Consequently, the 8-bit resolution would be spread over a smaller voltage range, and the ADC's resolution would be ±0.01 V. If the maximum DAC output was increased to 10 V then the ADC's resolution would be only ±0.02 V.

The sensitivity was adjusted so that the mid-point of the measurement range (i.e. 127 tu), corresponded approximately with the desired cloth tension. The sensitivity was set with the robot finger lying horizontal. A 100 g weight was placed on the free end of the sensor, and the sensitivity was adjusted until a reading of 156 tu was obtained. Thus 1 tu was equivalent to 0.64 gf.

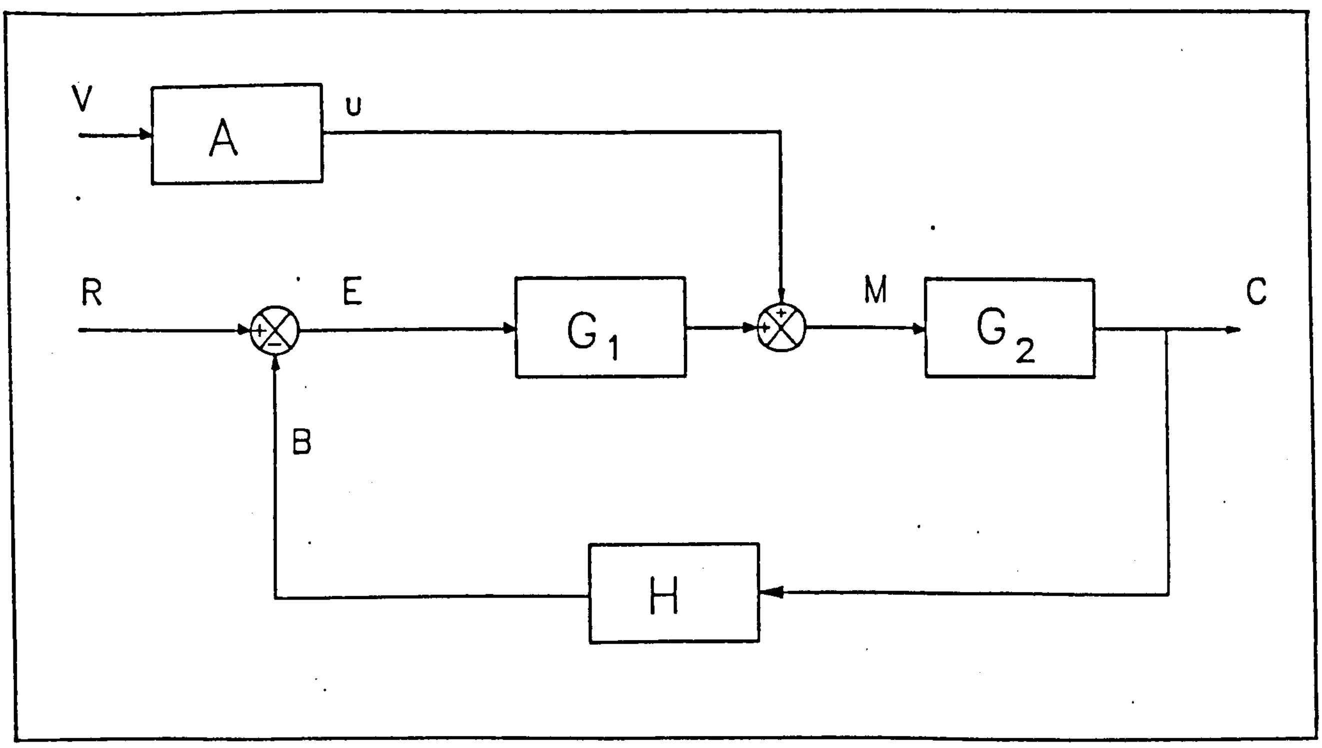


Fig. 4-7: Closed Loop Tension Control System

4.4. Closed Loop Control System Design

4.4.1. Control System Approach

4.4.1.1. Block Diagram

The block diagram for the closed loop control system is shown in fig. 4-7, and the symbols are defined in table 4-1.

	ANSI Std Nomenclature	Description
V	Tracking Signal	shaft encoder count
R	Reference Input	desired cloth tension (tu)
E	Actuating Signal	cloth tension error (tu)
U	Unmodified Variable	cloth feed speed (mm/hs)
C	Controlled Variable	actual cloth tension
B	Feedback Signal	measured cloth tension (tu)
M	Manipulated Variable	ALTER data for X direction
A	Input Element	relationship between V and U (equation (4.2))
G_1	Control Elements	transfer function
G_2	System Elements	controlled system (Plant)
H	Feedback Elements	tension sensor and signal conditioning circuitry

Table 4-1: Tension Control System Terminology

The "Plant", G_2, refers to the combination of the following elements :

* ALTER communications
* VAL II control system
* PUMA 560 robot
* cloth
* sewing machine

In the closed loop system, the open loop system for tracking the sewing machine speed (section 4.2.3.2), was retained, but the open loop robot speed demand was modified by negative feedback of the cloth tension, in the following manner;

- If the robot is lagging behind the cloth feed, the cloth tension will rise and produce a negative tension error, which will lead to an increase in the robot speed demand.

- If the robot is moving too fast, the cloth will go slack, and the positive tension error will lead to a reduction in the robot speed demand.

4.4.1.2. Software Implementation

The closed loop control system was implemented in the SEW Task, using the following algorithm :-

```
while not end_of_seam do
begin
    read V                          ( shaft encoder count)
    calculate U                     ( sewing speed )
    read B                          ( cloth tension )
    calculate E                     ( tension error )
    calculate M = (G1 E) + U
    send M to VAL II via ALTER
end
```

4.4.2. Preliminary Investigation into Closed Loop Control

A series of experiments were carried out to explore the control problem and to investigate the effect of different transfer functions. Although satisfactory control was not achieved by these trial-and-error attempts, various control problems were highlighted.

4.4.2.1. Start-up Acceleration

When the sewing machine started sewing, the cloth experienced a large initial tension due to the time delay between start-up of the sewing machine and the robot. Such a large tension peak caused havoc in the closed loop tension control system.

The problem was effectively solved by slowly accelerating the sewing machine at start-up. The start-up acceleration was controlled by a function called speed_control (see Appendix D).

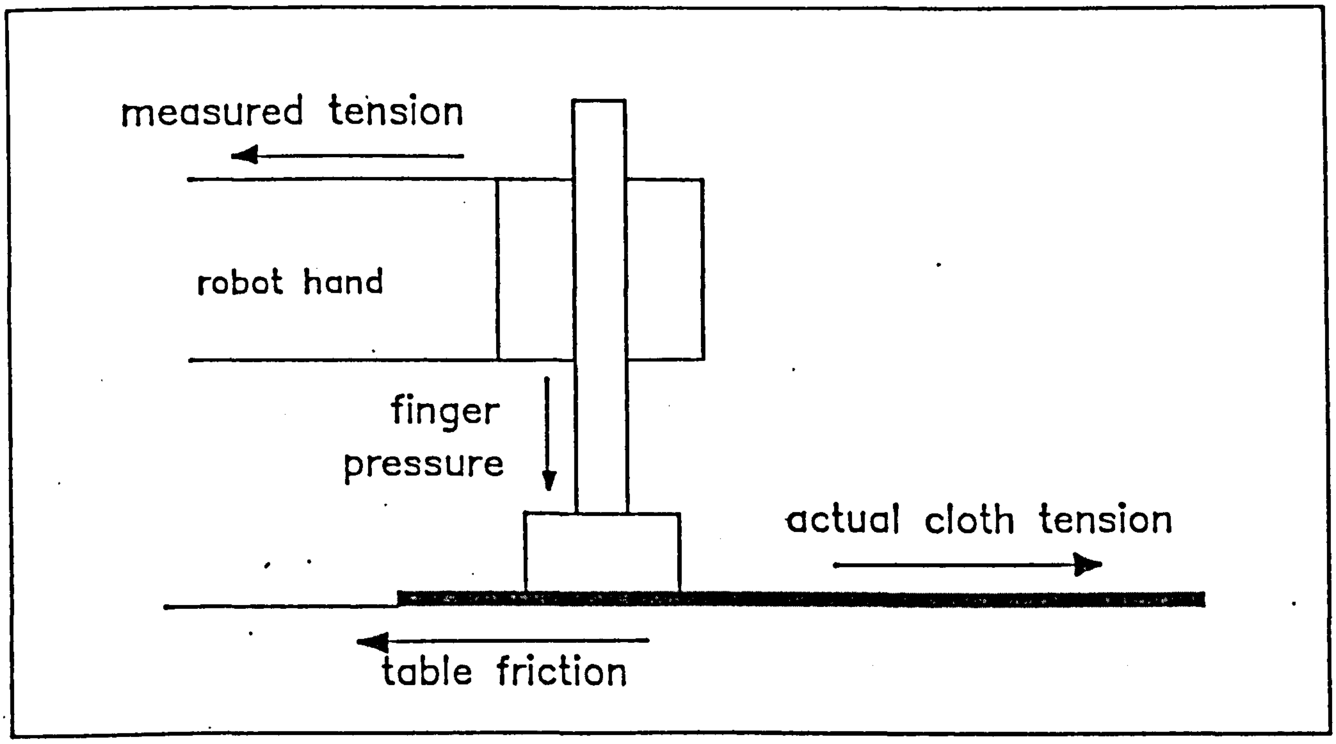


Fig. 4-8: Effect of Table Friction on Tension Measurement

4.4.2.2. Effect of Table Friction

When a large proportional gain was applied to the tension error, the robot "vibrated" about a stationary point. This behaviour was traced to the effect of the table friction on the tension measurement.

When the robot moves forward (fig. 4-7), towards the sewing machine, the force sensor measurement is :-

measured tension = cloth_tension - table_friction

However, when the robot moves away from the sewing machine, the table friction changes direction (since friction always opposes motion), and the force sensor measures the following :-

measured tension = cloth_tension + table_friction

Consequently, when the robot attempted to move backwards in order to tension the slack cloth, it immediately sensed an apparent cloth tension, even though the cloth was still slack. The solution to this problem was to limit the robot motion, in the x direction, to forwards only.

4.4.2.3. System Instability

When, under closed loop control, a small proportional gain was applied to the tension error, the system became unstable and the cloth tension oscillated between very high and zero tension. When the gain was reduced to a value close to zero, the system was effectively under open loop control, and the cloth tension tended to drift off towards either very high or zero tension.

The difficulty in obtaining stable closed loop control was due to the characteristics of the "Plant". Since a small extension of the cloth results in a large increase in cloth tension, the Plant has a high inherent proportional gain. A system with a high proportional gain has a greater tendency to go unstable, due to a reduced stability margin [57]. The stability margin can be increased by introducing compensation into the transfer function.

4.4.2.4. System Compensation

In classical control systems, there are two main forms of compensation that can be introduced into the controller transfer function G_1, viz. derivative and integral control. Derivative control can increase system damping and improve system stability, but it has no affect on steady-state errors and it accentuates any noise or disturbances in the system. Integral control reduces steady-state errors to zero, but they increase the order and type of the system, and therefore it may make the system even more unstable.

Although the raw tension signal had a smooth waveform when viewed on an oscilloscope, the variation in the values of peak tension, which are used in the control algorithm, was noisy. Furthermore, the slow sampling rate of the peak tension would lead to large errors when calculating its time derivative. Consequently, derivative control was unsuitable for this system.

However, integral control could be beneficial to long-term steady-state tension control, provided that the combination of proportional and integral gain values give sufficient system stability [57,68].

4.4.2.5. Implementation of Integral Control

The tension integral was calculated by maintaining a variable which contained the sum of all previous peak tension readings.

A consequence of a slow start-up (section 4.4.2.1.) was a significant build-up in the tension integral of the start-up tension errors. This problem caused a distorting effect on the integral control, and it was effectively solved by resetting the integral to zero on the first occasion that the tension passed the desired tension.

4.4.2.6. Effect of Speed on Closed Loop Control

As explained in section 4.4.1.2. and in fig. 4-7, the ALTER data in the X direction was calculated as follows :-

$$M = U + E G_1 \qquad (4.13)$$

When the closed loop control was attempted for different sewing speeds, it was obvious that this control equation was inadequate. Although U is proportional to sewing speed, EG_1 is not and therefore the modifying action of EG_1 on M will be effectively reduced with increased sewing speed. The control equation was modified to make the controller, G_1, independent of the sewing speed, as follows :-

$$M = U (1 + E G_1) \qquad (4.14)$$

In other words, the ALTER data, M, is modified proportionately by the tension feedback.

4.4.2.7. Final Block Diagram

The final block diagram for the closed loop control system is shown in fig. 4-9. The modified control equation is represented by the multiplication junction, and the controller transfer function, G_1, has been expanded to show the proportional and integral components, K_1 and K_2 respectively.

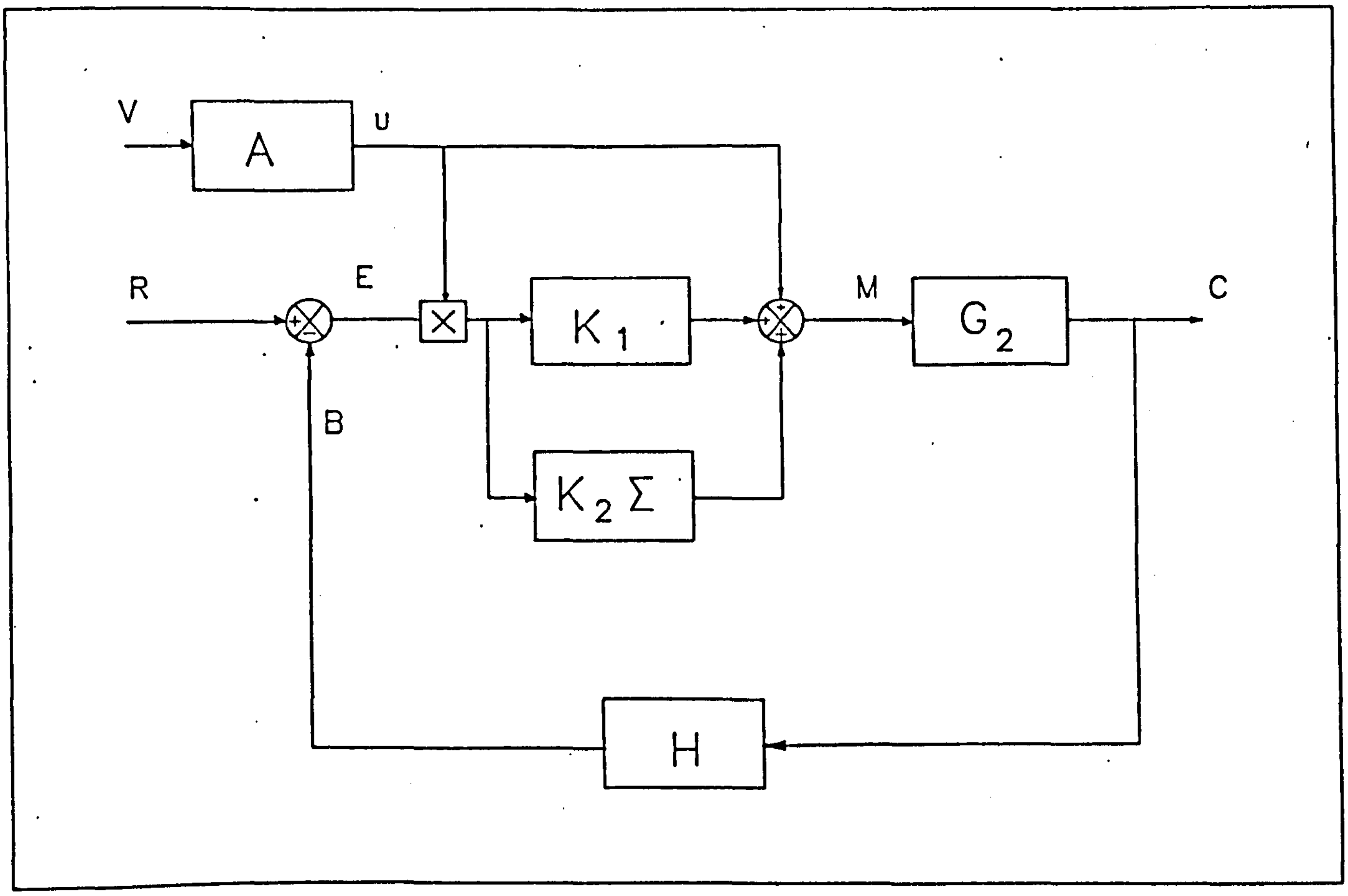


Fig. 4-9: Modified Block Diagram of Tension Control System

4.4.3. Bode Design of Control System

Although the preliminary experiments had provided much valuable information concerning the control problem, the Plant's characteristics were still largely unknown, and the trial-and-error attempts at selecting suitable integral and proportional gain values had been unsuccessful. Clearly, a formal control system design procedure, based on more precise knowledge of the Plant, was necessary.

The system has several significant non-linearities, which are listed and discussed below section 4.6.1. A design procedure which accounted for these non-linearities would require a complete analysis of each one and of their interactions, which would be very difficult to achieve satisfactorily. "Linearization" techniques, in which the system is approximated to a linear system in the region of interest, are applied to non-linear systems, whenever applicable, so that classical linear control design procedures can be used [57].

Since a mathematical description of the cloth tension system would have been difficult to derive, an experimentally based design procedure was more suitable. The Bode design method requires the open loop frequency response, which can be measured experimentally. The Bode technique is based on the assumption of a linear system, and although a linearization approximation was not strictly applicable to this system, the Bode design procedure was carried out in order to obtain an approximation of the Plant's dynamic behaviour, and to assist in identifying the "ball park" in which the correct gain values lie.

4.4.3.1. Bode Design Procedure

The theory on which the Bode analysis and design procedures are based, is explained in many textbooks [57,68,69]. The Bode design procedure has the following stages :-

a) The open loop frequency response of the system (i.e $G_e H(jw)$), is obtained either by measuring the steady-state response in amplitude and phase to a sinusoidal input function, or by analysis.

b) The frequency response function is plotted on a Bode diagram.

c) Control system stability performance is selected in terms of gain margin and phase margin.

d) A compensation function is chosen so that it will "reshape" the $G_e H(jw)$ plots and provide the required stability performance. This stage may be iterative.

4.4.4. Measurement of Open Loop Frequency Response

4.4.4.1. Experimental Technique

The open loop frequency response was measured as follows :-

a) The sewing speed was fixed to 2000 stitches per minute. The stitch length knob was adjusted so that the cloth tension was constant, under open loop control.

b) The test fabric that was selected is described in section 4.4.4.2. The dimensions of the test panel was 710 mm by 280 mm.

c) The first 170 mm of the test panel were sewn up under pure open loop control, to ensure steady state conditions.

d) For the remainder of the length, a sinusoidal function was superimposed on the ALTER data, and the resultant tension variations were recorded every handshake.

e) The amplitude of the forcing function was fixed at either 1 mm or 2 mm and the period of the forcing function was varied between 4 and 24 handshakes.

f) The amplitude and phase angle of the tension variations were extracted using the auto-correlation statistical technique.

g) The maximum tension amplitude that could be measured with an 8-bit ADC and a sensitivity of 0.64 gf/tu was ± 80 gf. The tension sensor sensitivity was halved to 1.28 gf/tu, so that a greater range of tensions could be measured. The tension measurements taken during these tests were then doubled so that the sensor's effective sensitivity was still 0.64 gf/tu.

4.4.4.2. Test Fabric

A light, tightly woven cotton plain weave fabric was selected for the experimental measurement of the open loop frequency response. This fabric, which was also used in the majority of the final performance tests (sections 4.5.1. and 5.5.1.), was chosen because it was relatively sensitive to to pucker, compared with suiting fabrics. Excessive tension variations during sewing produced puckered seams in the test fabric. The test fabric weighed 0.0143 g/m^2, with 54 ends per inch and 46 picks per inch.

4.4.4.3. Results

Fig. 4-10 shows examples of the tension variations obtained, after reduction by auto-correlation. The full set of experimental results is given in table 4-2; the Bode plot diagram for these results is shown in fig. 4-11.

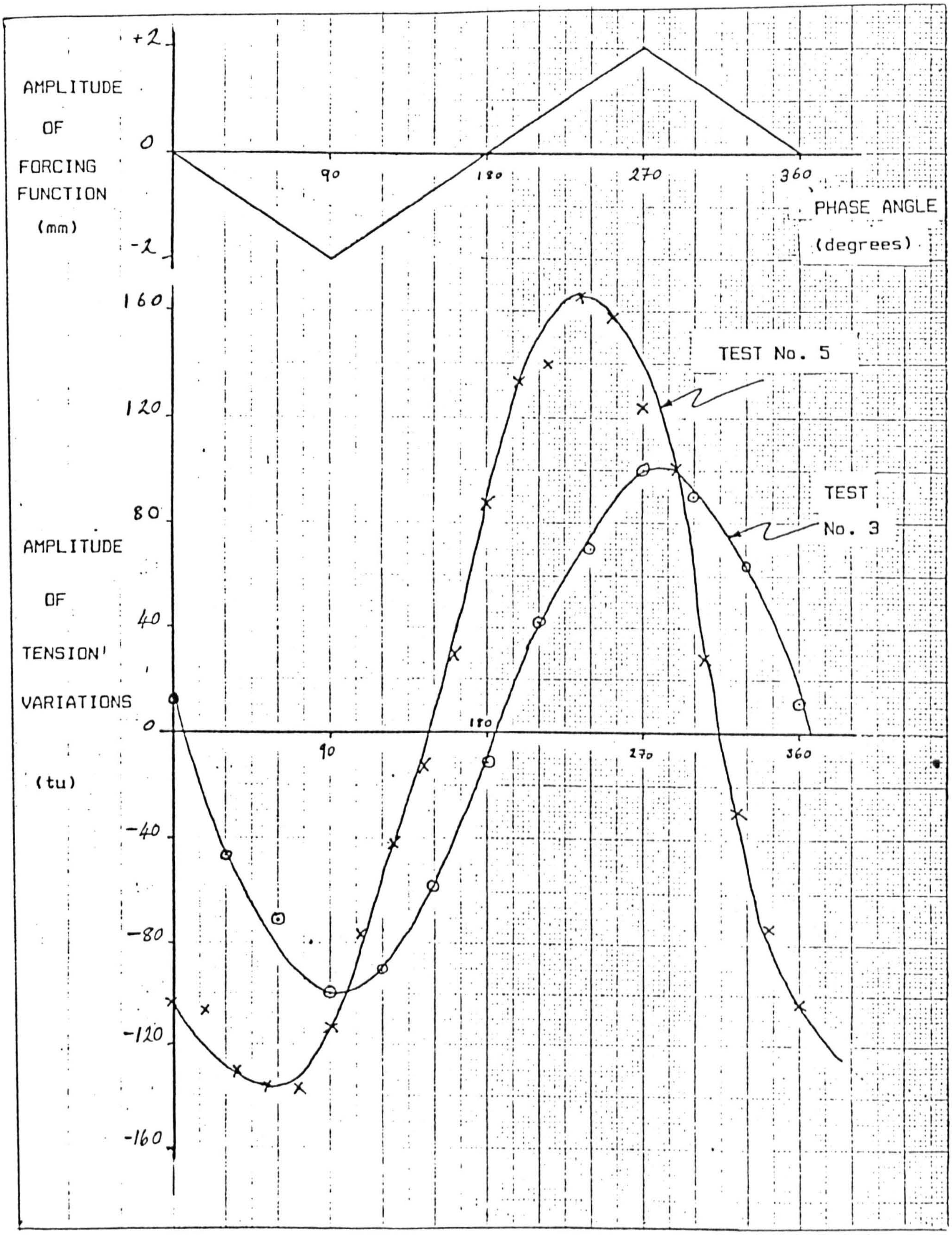


Fig. 4-10: Cloth Tension Variations Due to Sinusoidal Forcing

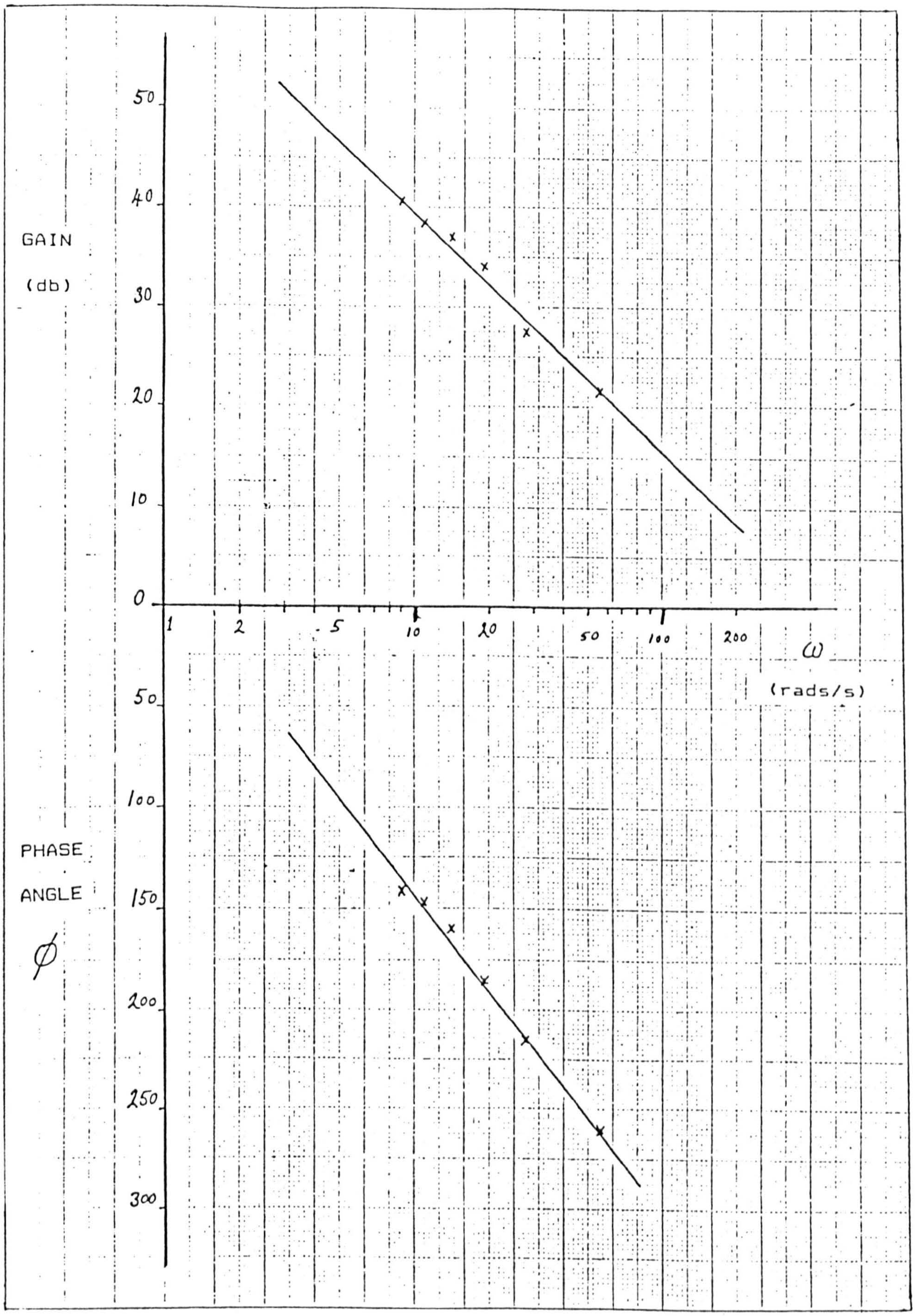


Fig. 4-11: Bode Plot Diagram for Cloth Tension Control System

Test No	Forcing Function			Tension Variation	
	Amplitude mm	Period hs	Frequency rad/s	Amplitude tu	Phase Shift degrees
1	1.0	4	56	12	260
2	2.0	8	28	48	215
3	2.0	12	19	100	186
4	2.0	16	14	138	160
5	2.0	20	11	166	148
6	2.0	24	9	212	142

Table 4-2: Experimental Results for Open Loop Frequency Response

4.4.5. Compensator Characteristics

The controller transfer function is given by :-

$$G_1 E = M - U = U (1 + K_1 E + K_2 \int E) \qquad (4.15)$$

For simplification, a constant sewing speed, U, can be assumed, and then the second summing junction can be included in the controller transfer function, as follows;

$$G_1 E = M = U (K_1 E + K_2 \int E) \qquad (4.16)$$

Taking the Laplace Transform yields,

$$M(s) = U \left(K_1 E(s) + \frac{K_2 E(s)}{s} \right) \tag{4.17}$$

The transfer function of the controller is then given by,

$$P(s) = \frac{M(s)}{E(s)} = \frac{s U K_1 + U K_2}{s} \tag{4.18}$$

The transfer function can be reduced to Bode form by replacing s with jw, as follows,

$$P(jw) = \frac{M(jw)}{E(jw)} = \frac{jw U K_1 + U K_2}{jw} \tag{4.19}$$

Rewriting (4.19) gives,

$$P(jw) = U K_2 \cdot \frac{1}{jw} \cdot \left(1 + jw \frac{K_1}{K_2}\right) \tag{4.20}$$

$$= \text{constant} \cdot \text{integrator} \cdot \text{single zero}$$

The magnitude and phase angle of the compensator are given by,

$$\text{mag}(P(jw)) = U \sqrt{(K_1^2 + (K_2^2 / w^2))}$$

$$\text{ang}(P(jw)) = \tan^{-1} \left(\frac{K_2}{w K_1} \right) \tag{4.21}$$

Figure 4-12 shows the Bode plot of the compensator function, P(jw), which can be sketched directly from equations (4.20) and (4.21).

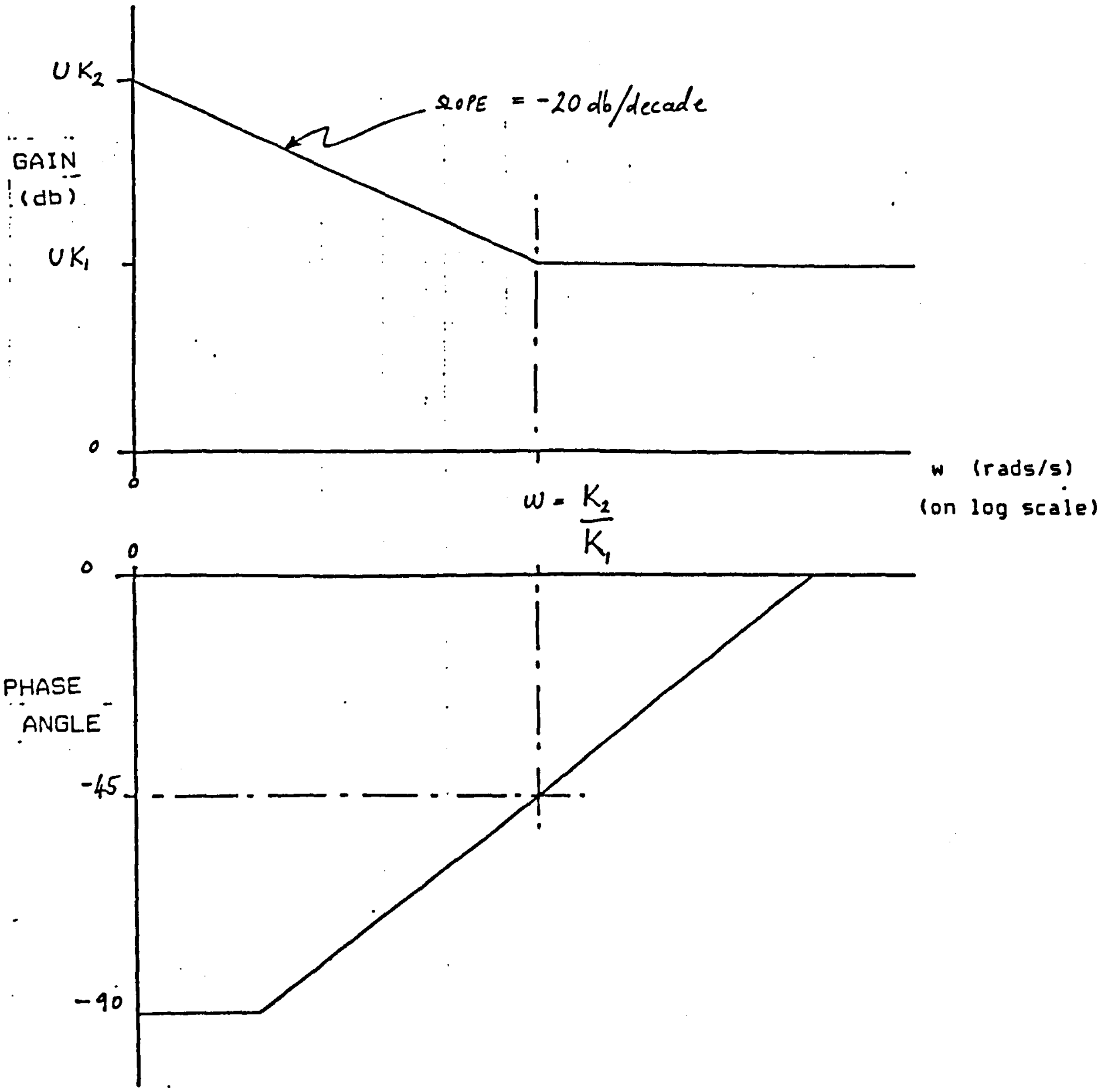


Fig. 4-12: Bode Plot Diagram of Compensator, P(jw)

4.4.6. Determination of Compensator Parameters

4.4.6.1. Calculation Method

The control system was designed to meet the following stability criteria,

$$\text{gain margin} = 8 \text{ db} \qquad (4.22)$$
$$\text{phase margin} = 30°.$$

The K_1 and K_2 factors were calculated to give maximum system performance within the above stability criteria, using an iterative graphical procedure, as follows :-

a) Find the maximum K_1 that meets both the phase and gain margin requirements, assuming K_2 is zero.

b) Calculate K_2, so that w_0, the centre of the compensator frequency range, is positioned 2π rads/s below the −180° crossover frequency.

c) Recheck that the stability criteria are still satisfied for this value of K_2.

4.4.6.2. Compensator Calculation

a) Apply phase margin criterion, assuming K_2 is zero.

The phase margin criterion states that at a phase change of 150°, the system gain should be less than 1 (or 0 db).

From fig. 4-11, 150° corresponds to a gain of 75 (or 37.5 db) for the uncompensated system, and therefore the maximum value for UK_1 is -37.5 db (or 0.0133).

b) Apply gain margin criterion, assuming K_2 is zero.

The gain margin criterion states that the gain should be less than -8 db at the 180° crossover frequency. From fig. 4-11, the uncompensated system has a gain of 42 (or 32.5 db) at the crossover frequency. Therefore the largest value for UK_1 is -40.5 db (or 0.0094).

All the frequency response tests were carried out at 2264 stitches per minute, with a stitch length of 3 mm, which resulted in an ALTER demand of 3.17 mm per handshake. Therefore, the maximum value for K_1 is

$$K_1 = 0.0094 / 3.17 = 0.003 \ \ tu^{-1} \qquad (4.23)$$

c) Calculate K_2 by graphically positioning w_s 2π rads/s below the crossover frequency, on the Bode diagram.

From fig. 4-11, the crossover frequency for the uncompensated system is 17.7 rads/s. From fig. 4-12,

$$w_s = \frac{K_2}{K_1} = \frac{17.7}{2\pi} = 2.82 \ \ \text{rads/s} \qquad (4.24)$$

Hence,

$$K_2 = 2.82\ K_1 = 0.0085\ tu^{-1}\, s^{-1} \qquad (4.25)$$

However, K_2 is required in terms of handshakes, not seconds.

$$K_2 = 0.0085 / 28 = 0.0003 \ \ tu^{-1}\, hs^{-1} \qquad (4.26)$$

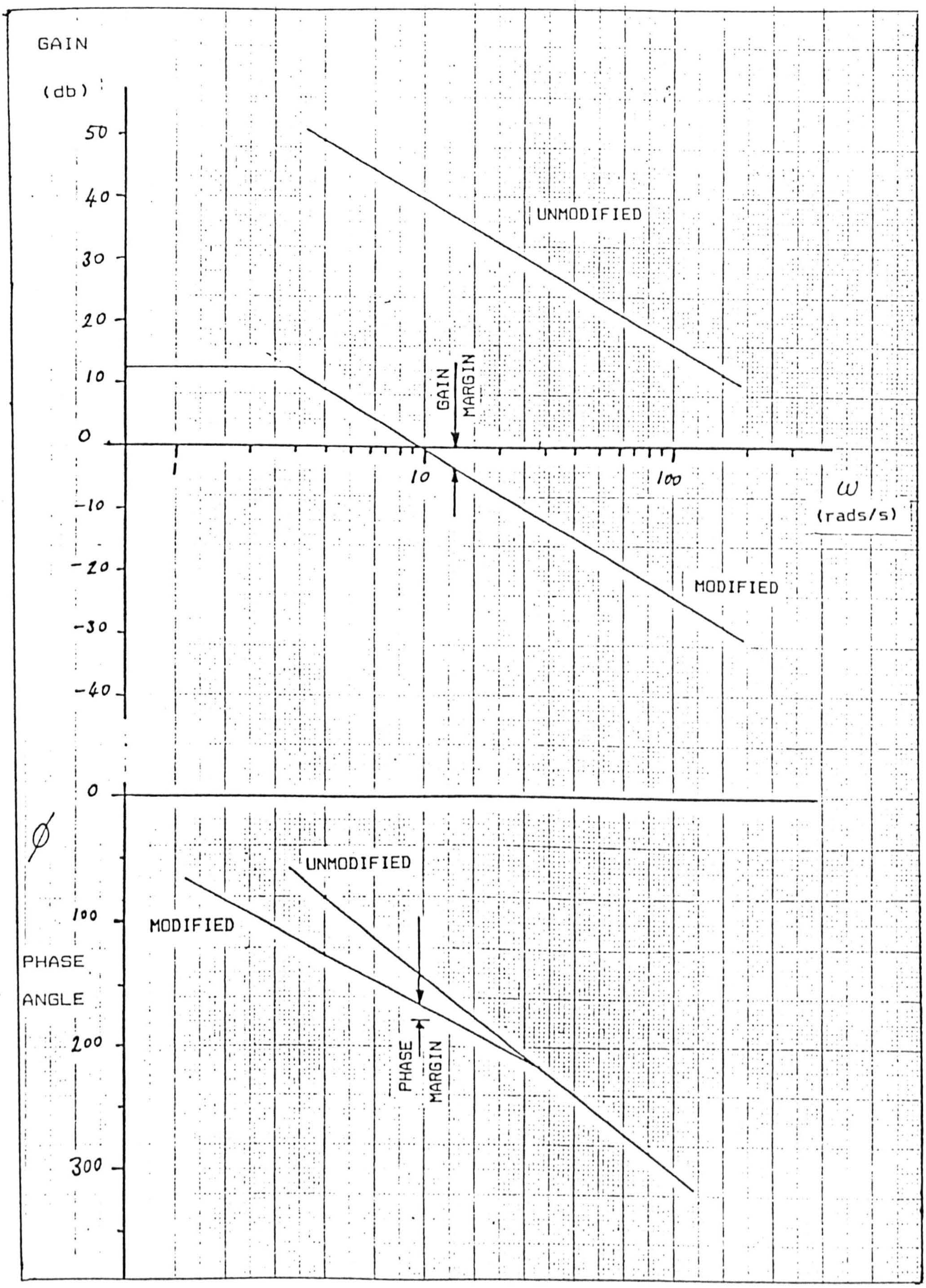


Fig. 4-13: Modified Bode Plot Diagram

d) The modified open loop Bode plot is plotted in fig. 4-13. The crossover frequency is now 15 rad/s, the gain margin is 4 db, and the phase margin is only 20°. Consequently, K_1 and K_2 have to be reduced further until adequate stability margins are obtained.

Additional calculation iterations were not attempted since the calculation procedure is only approximate for this system. The Bode design procedure assumes a linear system, and the cloth tension system is particularly non-linear. Therefore, the controller transfer function was "fine-tuned" experimentally.

4.5. Control System Performance

4.5.1. Performance Criterion

When considering the performance of the tension control system, two different criteria could be used; the standard deviation or the average of the tension error. The standard deviation gives a measure of the tension fluctuations, and the average error indicates the tension offset during the sewing operation.

Although tension fluctuations are detrimental to seam quality, a small constant offset to the demand tension will not cause puckering. Since the ultimate objective of the tension control system was to produce pucker free seams, the standard deviation was used as the performance criterion for comparing the system's performance under different conditions.

Initial performance tests were performed using the same test fabric used in the Frequency Response Measurement Experiment (section 4.4.4.), and the cloth panel had approximately the same dimensions. The reference tension, R, was set at 70 tu (or 45 g) for all performance tests.

Although, the seam quality is inversely proportional to the standard deviation of the tension error, the sensitivity of seam quality to tension variations varies enormously for different fabrics [71]. The test fabric was particularly sensitive to pucker due to its light weight and tight weaving, such that a standard deviation of tension error of 30 tu or more resulted in an unsatisfactory puckered seam. When the tension variation was controlled to 20 tu or less, the resultant seam was of excellent quality.

When two plies of the test fabric were sewn up, the extra weight reduced the pucker sensitivity to tension variations, such that a standard deviation of 80 tu resulted in an acceptable seam. When a heavier suit fabric was tested, a similar reduction in pucker sensitivity was observed.

4.5.2. Experimental Fine-Tuning

The Bode design procedure indicated that the integral and proportional gain parameters should be less than 0.0003 and 0.003, respectively. During the preliminary experiments, such low values had been considered insignificant, and therefore satisfactory control had been elusive. Once the correct range of values was known, the optimum gain values were easily determined experimentally.

The performance results for a sample of the fine tuning experiments are given in table 4-3. The following gain values were finally selected as the optimum values for providing stable and adequate tension control for a single ply of the test fabric over a range of speeds :-

$$K_1 = 0.0015, \quad K_2 = 0.00003 \qquad (4.27)$$

The results in table 4-3 demonstrate that system performance was particularly sensitive to excessive proportional gain, K_1. A sample printout of the robotic sewing program, showing details of the performance of the cloth tension control, is shown in fig. 5-21.

K_1 tu^{-1}	K_2 $tu^{-1}\ hs^{-1}$	Update Rate hs^{-1}	Sewing Speed rpm	Std. Dev of tensn error tu
0.0015	0.00003	1	2270	24.4
0.0015	0.00010	1	2270	31.1
0.0015	0.00001	1	2270	27.1
0.0045	0.00003	1	2270	61.4
0.0005	0.00003	1	2270	29.1

Table 4-3: Sample of Fine-Tuning Experimental Results

4.5.3. Performance Versus Speed

Once the optimum gain values had been determined, the control system's performance was measured for a range of sewing speeds. The results are shown in fig. 4-14 and the performance curves are identified in accordance with table 4-4.

For an update rate of 1 hs^{-1}, the tension control was satisfactory up to about 2000 rpm. A transition was observed at approximately 2750 rpm, such that higher speeds produced much poorer seams.

The processing overhead for the tension control calculations was not significant, so that the maximum update rate was easily achieved. However, the seam width control system overheads were significant, so that when both systems were running simultaneously in the edge seaming operation, the update rate was reduced to at least 0.5 hs^{-1}. The performance of the tension control system was measured for a reduced update rate of 0.5 hs^{-1} (see fig. 4-14).

The tension control was unaffected by the slower sampling rate at slow speeds, since at slow speeds a digital control system is effectively continuous. The sampling interval was 2 hs (or 56 ms) and at 1000 rpm, each stitch takes 60 ms. Since the control is based on peak tension measurements, the cloth tension cannot be sampled more than once per stitch. Consequently, the control system's effective sampling rate is limited by the sewing speed for speeds below 1070 rpm, and at higher speeds, it is limited by the ALTER update rate, viz. 0.5 hs^{-1}.

For speeds above 2000 rpm, the tension control was markedly worse for the slower update rate, as the sampling interval started to influence the control system performance. The transition in the performance curve occurred at a lower speed, 2250 rpm, for the slower update rate.

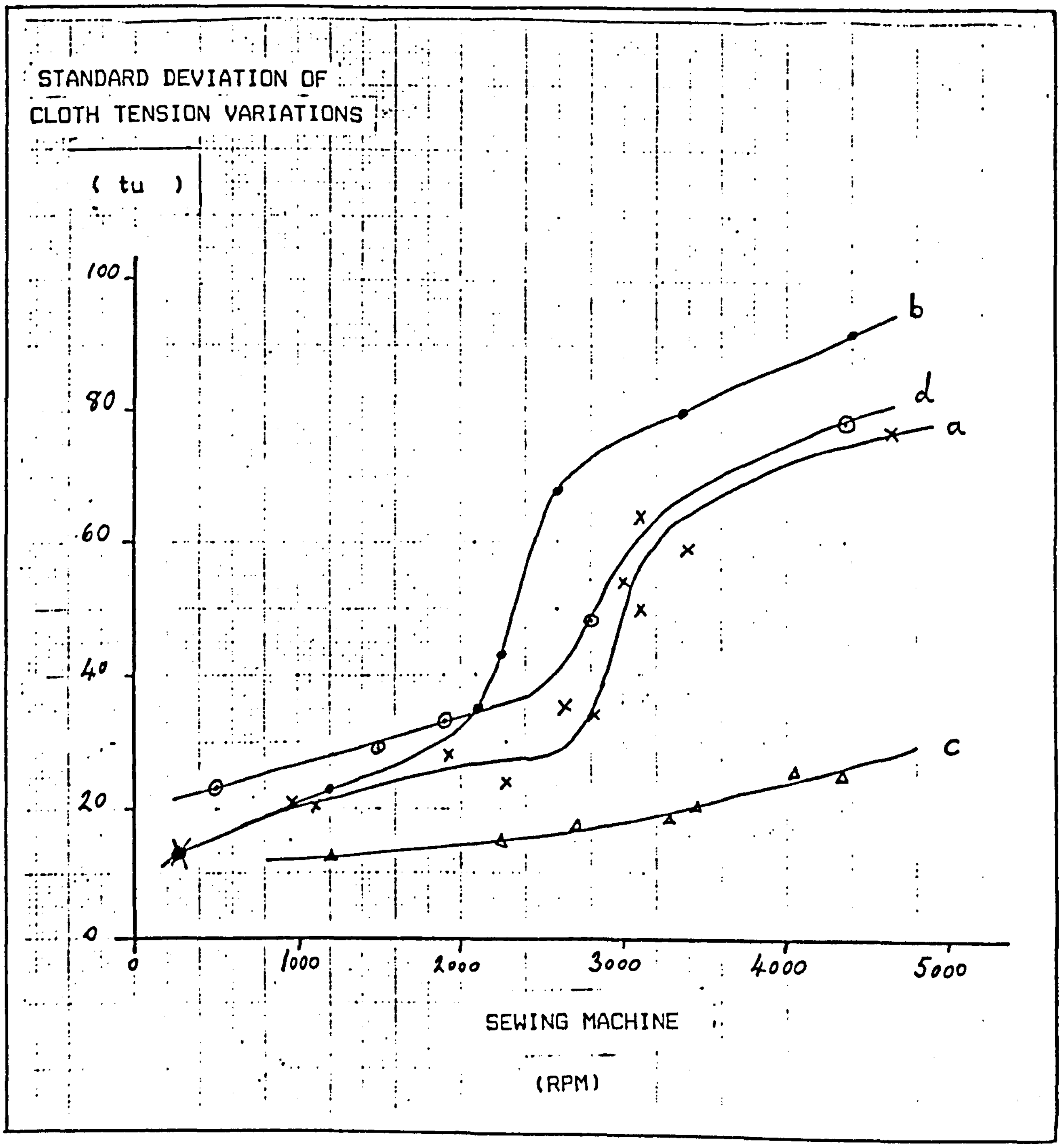


Fig. 4-14: Tension Control System Performance

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

Curve	Update Frequncy	No of Plies	Sewing Directn	K_1	K_2
a	1.0	1	normal	0.00150	0.000030
b	0.5	1	normal	0.00150	0.000030
c	1.0	1	bias	0.00150	0.000030
d	1.0	2	normal	0.00075	0.000015

Table 4-4: Key to Fig. 4-14

4.5.4. Performance Versus Fabric Properties

4.5.4.1. Sewing a Two-Ply Panel

Two plies are approximately twice as stiff as one ply, therefore the gain of the open loop transfer function of the Plant, G_2, will be doubled by adding a second ply. Consequently, the values of the compensator parameters, K_1 and K_2, must be halved, in order to maintain the equivalent closed loop performance that was developed for a single ply.

Fig. 4-14 shows the tension control system performance for sewing two-ply panels of the test fabric, when K_1 and K_2 were reduced to 0.00075 and 0.000015 respectively. Tension control was slightly worse for two-ply sewing; the performance curve closely follows the curve for single-ply sewing.

4.5.4.2. Sewing along the Bias

All the tests so far had been performed with the sewing direction approximately aligned with the warp or weft of the cloth panel. A test panel was prepared which was equivalent to the previous test panels, except that it was cut across the "bias", i.e. the direction of sewing was now at 45° to the warp and weft directions.

When the control system performance was measured using this test panel, good tension control was obtained at all speeds, (fig. 4-14). This was due to the much lower stiffness of the fabric in the bias direction, which effectively reduced the gain of the system and improved the stability margin. However, the fabric buckled badly during sewing, because of the high deformation of the structure of the fabric.

The buckling could have been reduced by either placing many finger pads all over the cloth surface, to minimize the fabric deformation, or by reducing the demand cloth tension to the level of a few grams force. However, the demand cloth tension could not be reduced to the low level required, because of the table friction and the hysteresis in the sensor design.

4.5.4.3. Different Fabrics

When other woven fabrics were tested, each fabric was found to require different values for K_1 and K_2. For example, the gain values had to be reduced by at least 60 % before equivalent tension control was obtained on a heavy trouser material. However, the heavier fabric was much less sensitive to tension variations.

When a single jersey knitted fabric was tested, the tension variations were small, but the panel buckled badly. The fabric behaved in a similar fashion to the original test fabric when it was sewn along the bias direction.

4.5.4.4. Spring Loading

Initially, all single-ply tests were performed with lightly sprung fingers (spring rate of 7 g/mm). When two-ply panels were tested, it was observed that when the top ply was pushed forward by the finger, it separated from the bottom ply which was held taut by the table friction. This problem was corrected by installing stronger springs with a spring rate of 70 g/mm.

The single-ply tests were repeated with the stronger springs, and no significant difference in the tension control or in the seam quality was observed.

4.6. Discussion

Maintaining a small tension on a cloth panel during sewing using an adaptively controlled robot was found to be a complex problem. The system's complexity is due to the combination of non-linear elements, which must be identified and understood individually. The most serious and troublesome non-linearities are those associated with the mechanical properties of the fabric.

4.6.1. System Non-Linearities

The major potential sources of non-linear behaviour in the tension control system are as follows :-

a) Time delay between measuring tension and the robot's corrective action.

b) The mechanical properties of the fabric panel.

c) The cloth tension can only be zero or positive since cloth buckles under compressive loading.

d) The table friction causes a dead zone, i.e. small tensions are measured as 0 tu.

e) The robot motion was limited to forward motion only, due to the effect of the table friction.

Other non-linearities, such as the velocity and acceleration limitations on the robot motion and the 8-bit resolution of the tension sensor, were not significant.

When the cloth tension control was satisfactory, the robot motion was smooth and continuous and the tension reading seldom dropped to 0 tu, i.e. items c), d) and e) did not affect the control system since the saturation levels were avoided. However, if the tension control was attempted at higher speeds or if a lower reference cloth tension, R, was specified, then these non-linearities would soon affect the control directly.

The first two items are discussed further below.

4.6.2. System Time Delay

Time delays have a destabilizing effect on control systems, and in particular, the stability of digital control systems is dependent on the sampling time delay [57,68]. At slow sewinq speeds, the system time delay is insignificant and the control is effectively continuous. However, as demonstrated in section 4.5.3., system performance can be improved at high speeds by reducing the time delay.

In the tension control system developed above, the affect of the time delay on the system dynamics has been ignored. In fact, different gain values are optimum for different sewing speeds. The system overall performance could possibly be improved by adjusting the values of K_1 and K_2 for different sewing speeds.

4.6.3. Mechanical Properties of Cloth

Fabrics have highly non-linear mechanical properties. Under tensile loading, they exhibit anisotrophy, a strain-dependent modulus and hysteresis. Under compressive loads they buckle and their behaviour under shear loading is also complex [60].

4.6.3.1. Tensile Loading along Warp or Weft Directions

Woven fabrics have non-linear load-extension curves, and a typical curve for the warp or weft directions is shown in fig. 4-15 [59,60].

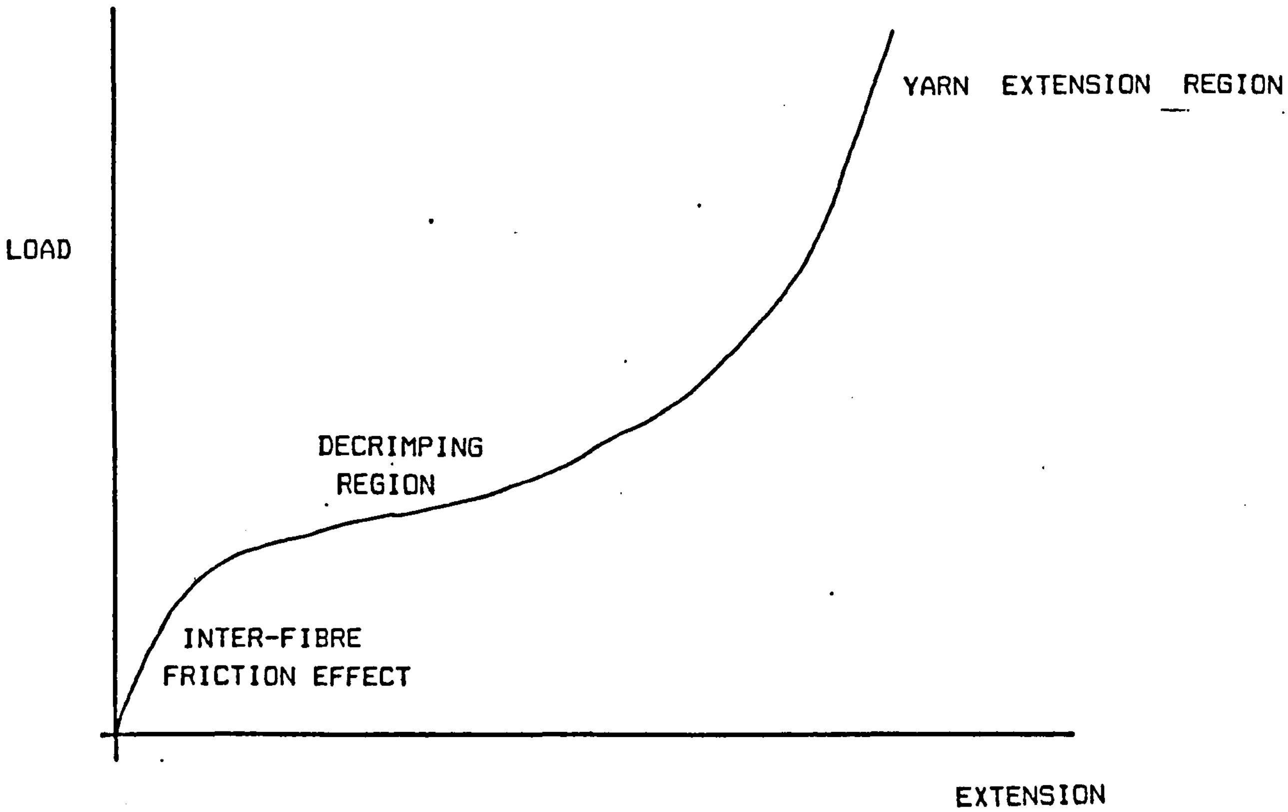


Fig. 4-15: Typical Load Extension Curve for Woven Fabrics

Three regions or phases can be identified on the curve below the yield point :-

a) The initial high modulus of the fabric is usually due to frictional resistance to bending of the thread.

b) Once the frictional restraint is overcome, a low modulus region is entered during which the threads in the direction of the force become taut (i.e. "decrimping").

c) Once the slack in the fibres has been taken up, a high modulus region is reached in which the fibres themselves are stretched.

In addition to the non-linear load-extension curve, there is considerable hysteresis between the extension curve and the recovery curve.

These non-linear characteristics were clearly responsible for much of the difficulty encountered in developing the cloth tension control.

4.6.3.2. Tensile Loading Along Bias Direction

The modulus of elasticity is slightly different in the warp and weft directions. However, in the bias direction (at 45° to the warp and weft), the modulus is very much lower than in either of the other two directions, since the cloth has a totally different deformation mechanism. When loaded along the bias, the cloth structure deforms by shear, i.e. the lattice framework is sheared as the fibres align themselves along the bias direction. This mode of deformation is shown diagrammatically in fig. 4-16.

Although the lower modulus of elasticity improved the performance of the tension control along the bias, the shear deformation of the fabric structure resulted in unacceptable buckling on either side of the high tension zone, which lay between the fingers and the presser foot. Although reducing the cloth tension to a few grams force may prove beneficial, this form of buckling can only be prevented satisfactorily by clamping the cloth against the table over as much of the panel as possible, during sewing.

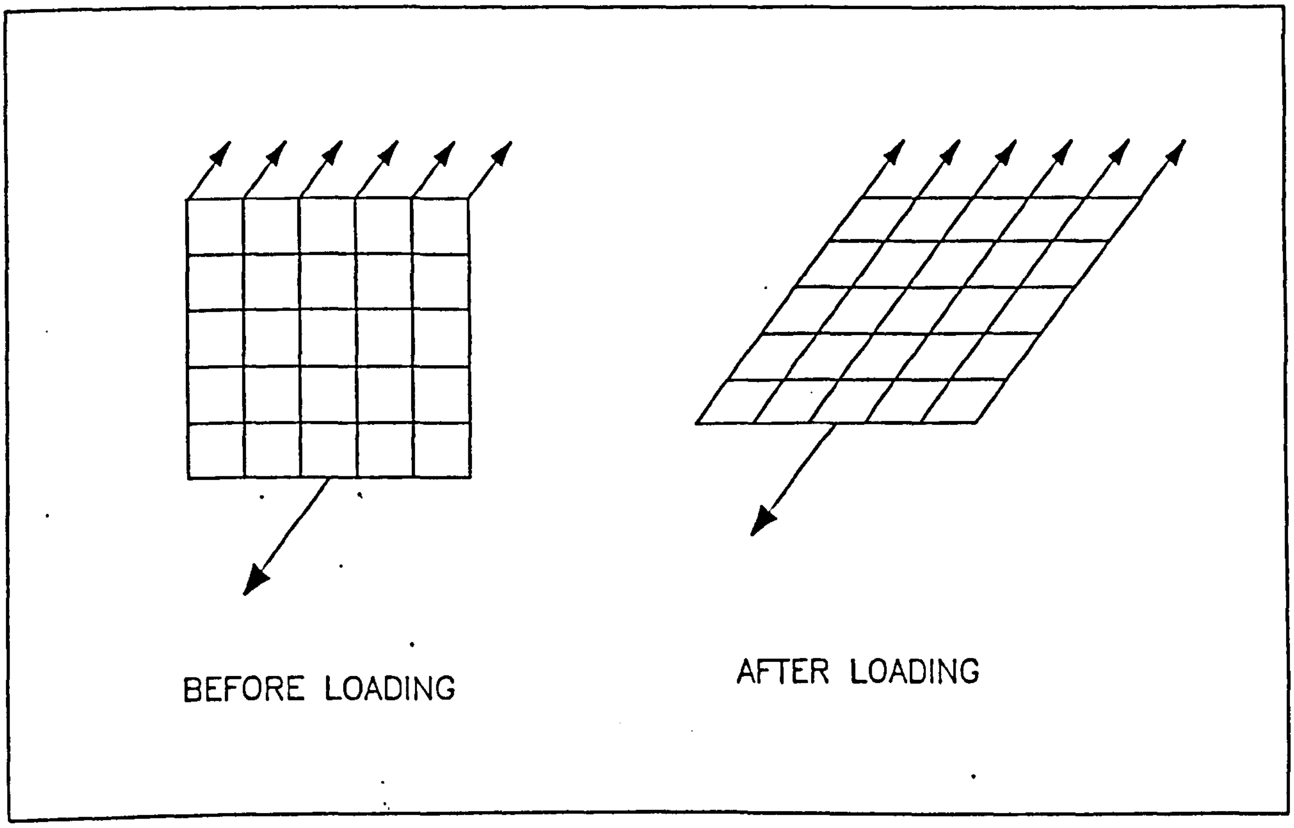


Fig. 4-16: Deformation of Woven Fabric, Loaded in the Bias Direction

4.6.3.3. Knitted Fabrics

Under tensile loading, knitted fabrics exhibit very high extensibility relative to woven fabrics, due to elongation of their looped structure. This high extension was limited to the high tension zone between the fingers and the presser foot, and the shear forces between the high and low tension zones generated severe buckling. Consequently, knitted fabrics are even more difficult to handle than woven fabrics cut along the bias.

4.6.4. Conclusions

a) A tension control system was successfully developed in which an adaptive robot holds the end of a cloth panel against a table during sewing.

b) The system is unsuitable for sewing along the bias direction of a woven fabric, or for knitted fabrics. Under such conditions, the fabric must be supported over a much greater proportion of its surface to prevent buckling, e.g. using a jig system or using a belt arrangement (section 1.3.2.2.). Alternatively, the tension measurement system could be redesigned to be more sensitive, to measure cloth tensions of only a few grams force.

c) Pucker free seams can only be produced at relatively slow sewing speeds in fabrics which are pucker-sensitive. Good quality seams can be produced in less sensitive fabrics at any sewing speed up to 5000 rpm.

d) The system's gain parameters require modification for different fabrics. However, values for K_1 and K_2 can be selected that will give good performance for a range of fabric types, especially if the fabrics have low pucker sensitivity.

e) The system can accommodate single or multi-ply cloth panels, as long as the number of plies is known in advance.

f) The system high speed performance can be improved by reducing the system time delay, e.g. increasing the update rate or reducing the handshake cycle time.

CHAPTER 5

SEAM WIDTH CONTROL SYSTEM

5.1. Introduction

5.1.1. Description of the Problem

In order to adaptively sew a seam parallel to the cloth edge, the robotic system must include a sensor that measures the position of the cloth edge relative to the needle in real time. This seam width measurement must then be used to compute a robot motion that will correct the orientation of the cloth panel about the sewing needle and eliminate the seam width error.

The first edqe seaming technique that was developed was the FAR technique, in which the robot fingers held the cloth at the far end of the cloth. The cloth tension control was developed for the same arrangement, which is shown in fig. 5-1.

When the robot holds the far end of the cloth, it can only correct the position of the cloth by rotating it about the sewing needle. Simultaneously, the robot must track the cloth feed by moving forwards to maintain a small cloth tension, using the tension control system described earlier.

Thus the robot cannot directly correct the seam width error, it may only alter the incident angle of the cloth axis. This corrective action depends on the forward motion of the cloth to help eliminate the seam error.

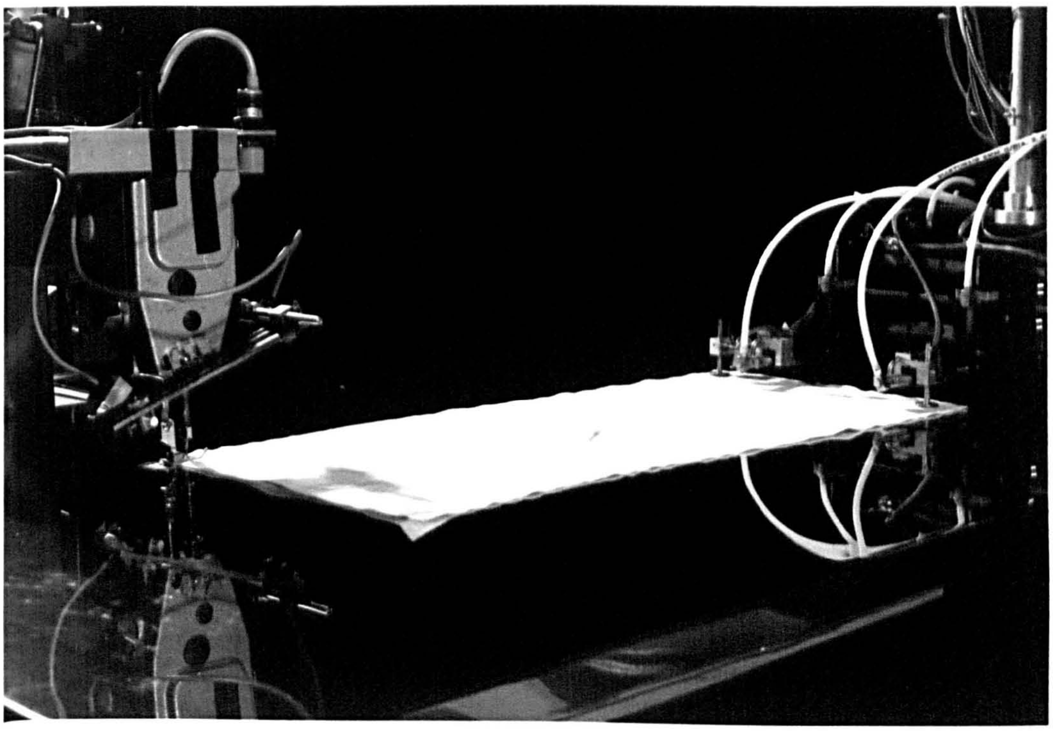


Fig. 5-1: Initial Finger Position for FAR Sewing Technique

5.1.2. Block Diagram

The control system is shown in schematic outline in fig. 5-2, and the symbols are defined in table 5-1.

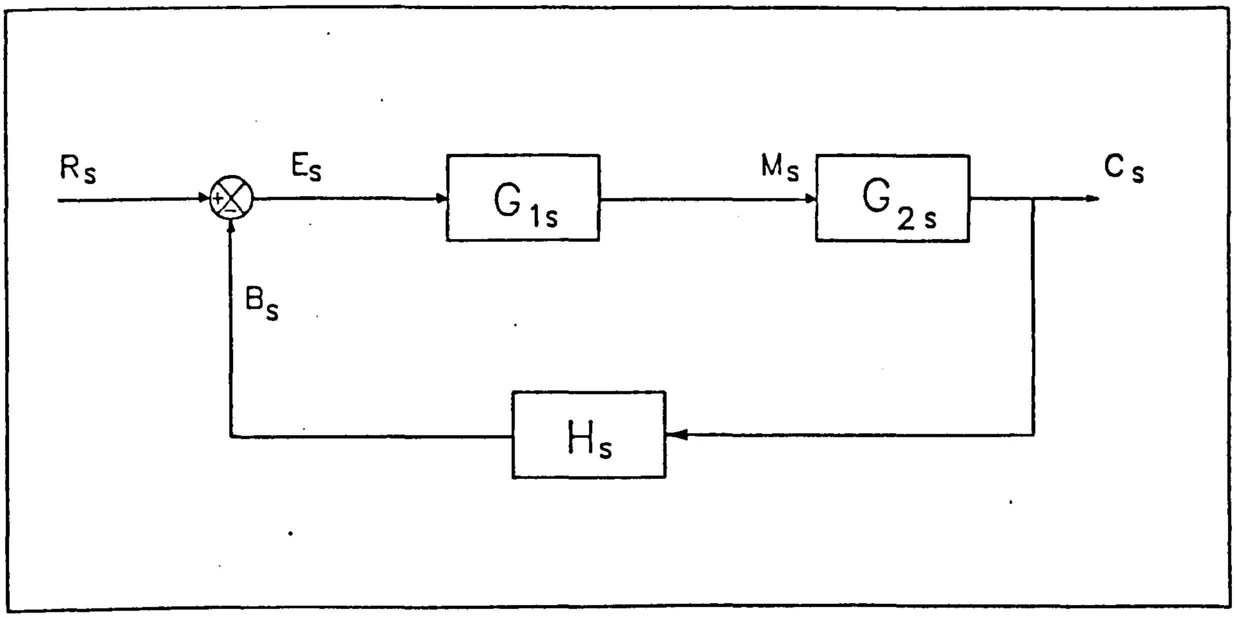


Fig. 5-2: Seam Width Servo Control System

	ANSI Std Nomenclature	Description
R_s	Reference Input	desired seam width (mm)
E_s	Actuating Signal	seam width error (mm)
B_s	Feedback Signal	measured seam width (mm)
C_s	Controlled Variable	actual seam width (mm)
M_s	Manipulated Variable	ALTER data
G_{s1}	Control Elements	transfer function
G_{s2}	System Elements	controlled system (Plant)
H_s	Feedback Elements	vision system

Table 5-1: Tension Control System Terminology

5.1.3. Design Options

The design of the tension control system was based on the experimental measurement of the frequency response of the open loop system, i.e. $G_2H(jw)$. This design method, which assumed a linear and continuous system, was necessary since the system could not be readily analyzed or simulated.

The seam width control system also involves a complex interaction of non-linearities due to fabric properties, table friction, motion limitations, etc. Attempts were made to analyse a model of the system, but they were aborted when it was realized that too many simplifying assumptions were necessary.

A simulation technique was developed for the seam width control problem which accounted for many system non-linearities. The simulation was based on two reasonable assumptions, that the cloth panel was stiff, and that the robot could accurately manipulate the cloth panel. The geometry of the system, robot motion limitations, vision system limitations and system time delays were incorporated into the simulation model.

5.2. Simulation Program

5.2.1. Development of the Algorithm

The simulation program, which was written in Turbo Pascal, was developed in 3 phases. First the basic control problem was simulated in which an ideal robot rotates the cloth by a computed correction angle based on accurate sensory measurements. The actual limitations of the PUMA 560 robot and the measuring accuracy of the proposed vision system were then introduced into the program. Finally a graphic display routine was added which permitted interactive use of the program during the simulation experiments.

5.2.1.1. Basic Algorithm

Fig. 5-3 describes the basic control problem and defines the main parameters which were used in the algorithm. The symbols used in fig. 5-3, together with other parameters used in the algorithm, are defined in table 5-2. The problem is viewed from within the coordinate frame of the cloth panel, as if the cloth remains stationary and the sewing needle rotates and translates across the cloth.

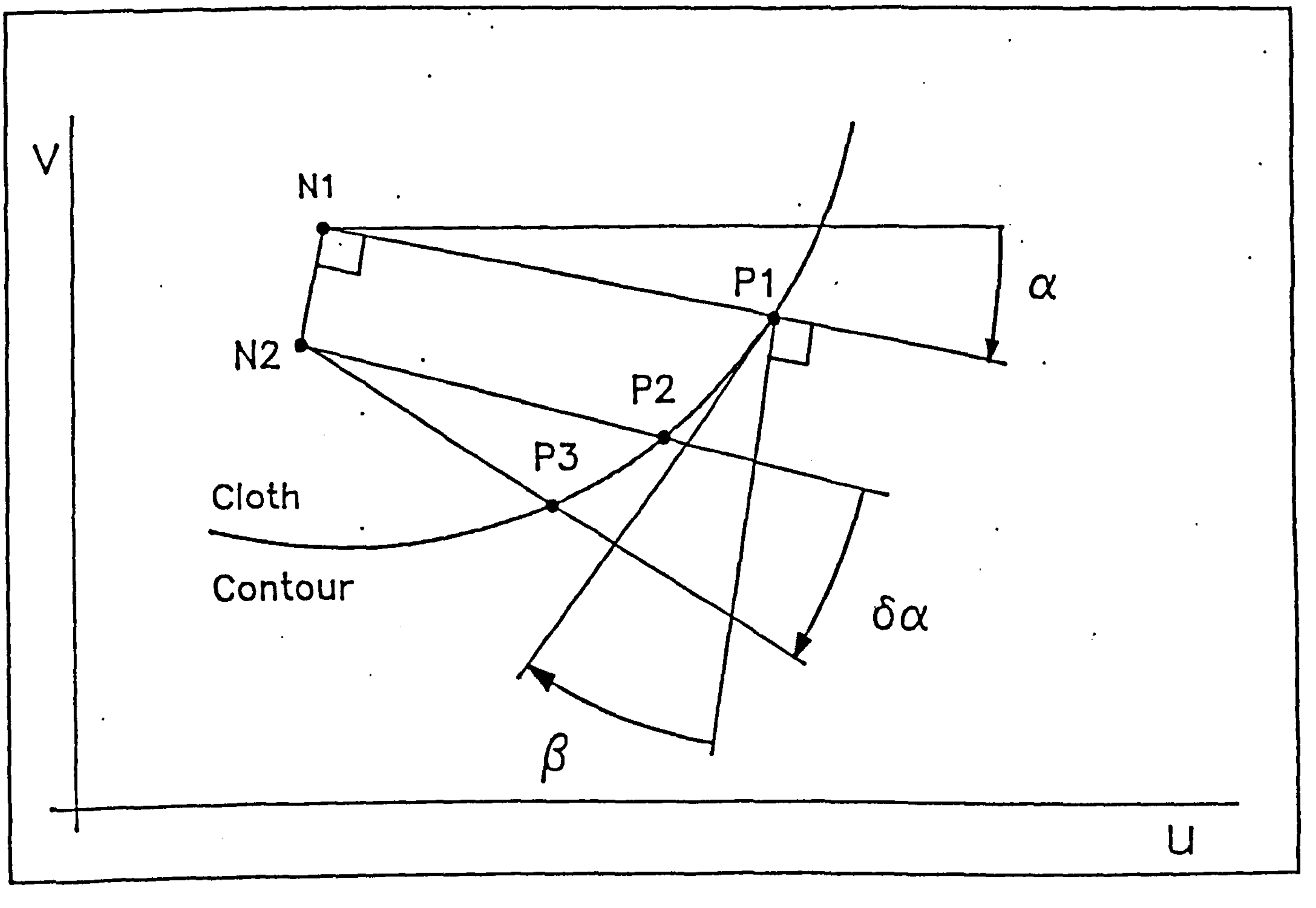


Fig. 5-3: Seam Width Control Problem

Item	Definition
x, y	coordinates of a point w.r.t. axes of sewing m/c
u, v	coordinates of a point w.r.t. axes of cloth panel
α	angle between y axis and u axis
β	angle of cloth contour tangent at $y = 0$ to x axis
$\delta\alpha$	corrective rotation angle to reduce seam error
δt	system time delay
N_1	needle position at time t_1
N_2	needle position at time t_2
$N_1 P_1$	measured seam width at time t_1
$N_2 P_2$	measured seam width at time $t_2 = t_1 + \delta t$
$N_2 P_3$	measured seam width after cloth rotated by $\delta\alpha$
$f(u)$	contour of cloth edge
V_c	cloth feed velocity
δs	distance sewn during δt

Table 5-2: Definitions of Simulation Parameters

The system time delay, δt, which is the delay between measurement and actuation, is a lumped parameter which comprises delays due to the vision system, processor delays, ALTER communication delays and actuation delays.

The origin of the x and y axes is the needle of the sewing machine as defined in section 2.8.4. In fig. 5-3, at time t_1, the x axis lies along the line $N_1 N_2$, and the y axis lies along $N_1 P_1$.

A parabolic function was chosen to define the contour of the cloth edge, for the simulation program, because of its gradually increasing curvature. The contour function was :-

$$f(u) = u^2 / 200 \quad (5.1)$$

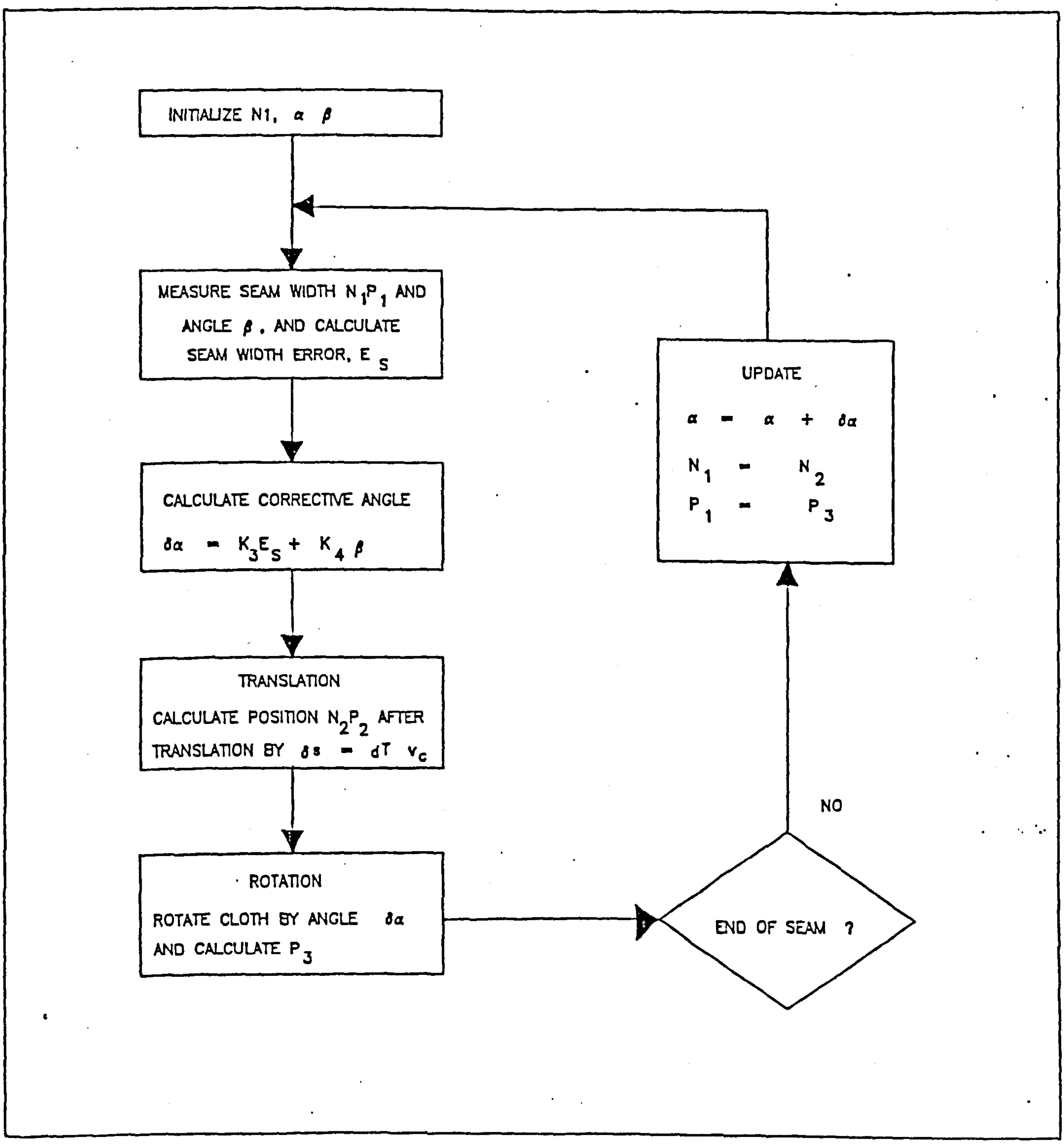


Fig. 5-4: Flowchart of Simulation Algorithm

The basic control algorithm, which is depicted in a flowchart in fig. 5-4, is based on the discretization of the measurement and actuation processes, i.e. the motion of the cloth due to the cloth feed mechanism and its rotation by the robot are treated as separate short motion segments which occur alternately.

Starting from a known initial needle position, N_1, the cloth is first translated along the x axis due to the cloth feed during the system time delay ($t_2 - t_1$). At time t_2 the robot rotates the cloth by the corrective angle $\delta\alpha$ which was computed using measurements taken at time t_1.

The cloth translation phase is depicted on fig. 5-3, by the needle moving from N_1 to N_2, relative to the cloth contour. The cloth rotation phase is depicted on fig. 5-3 as the sewing machine rotating by $\delta\alpha$ relative to the cloth contour.

The algorithm progresses along the seam length using the "time-marching" technique. At the end of each step the parameters α, N_1 and P_1 are updated and the calculations are repeated until a termination condition has been met.

5.2.1.2. Calculation of Seam Width Error, E_s

The line joining the needle and the cloth edge on the sewing machine y axis, N_1P_1, which can be measured directly by a vision system, is only an apparent seam width. Fig. 5-4 compares the actual and apparent seam widths.

Since the apparent seam width changes with the rotation angle of the cloth, α, initial simulation runs confirmed that the control system required a more accurate value for the seam width. The actual seam width cannot be measured directly but a satisfactory approximation can be obtained from the apparent seam width and cloth incident angle, ß, as follows :-

$$\text{seam\_width} = N_1 P_1 \cos \beta \tag{5.2}$$

Hence, the seam width error is given by :-

$$E_s = N_1 P_1 \cos \beta - R_s \tag{5.3}$$

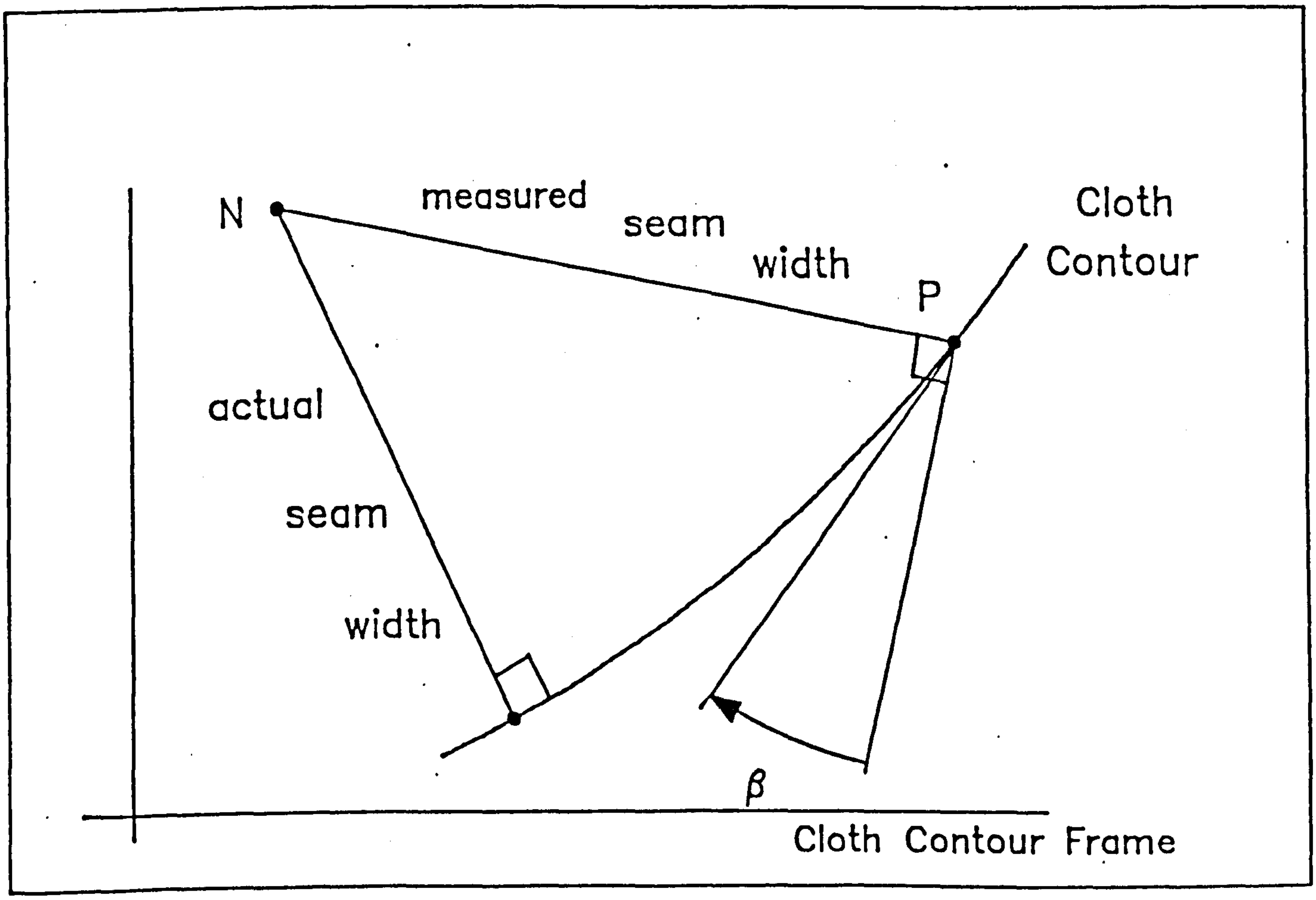


Fig. 5-5: Apparent and Actual Seam Width

Equation (5.3) is accurate for a straight line cloth edge and its accuracy is only dependent on the cloth curvature, and independent of the cloth angle, α. Consequently, this relationship was found to be suitable for the seam width control.

5.2.1.3. Calculation of Cloth Rotation

The position of the cloth edge (P), as detected by the vision system, for a particular needle position (N) and cloth rotation angle (α), was calculated from equations (5.5) and (5.8), which were derived as follows :-

Problem : given N_u , N_v and α, calculate P_u and P_v where (N_u ,N_v) and (P_u ,P_v) are the coordinates of N and P relative to the cloth contour.

Solution : NP is a straight line with gradient $-\tan\alpha$.

Thus

$$-\tan\alpha = \frac{N_v - P_v}{N_u - P_u} \qquad (5.4)$$

Since P lies on the curve, $200\,v = u^2$ (equation (5.1),

$$200\ P_v = (P_u)^2 \qquad (5.5)$$

Eliminating P_v between equations (5.4) and (5.5) yields

$$\frac{(P_u)^2}{200} + \tan\alpha\, P_u - (N_v + N_u \tan\alpha) = 0 \qquad (5.6)$$

Hence

$$P_u = -100 \tan \alpha \pm 10 \sqrt{(100 \tan^2\alpha + 2 N_v + 2 N_u \tan \alpha)} \qquad (5.7)$$

Since the required solution lies in the first quadrant,

$$P_u = -100 \tan \alpha + 10 \sqrt{(100 \tan^2\alpha + 2 N_v + 2 N_u \tan \alpha)} \qquad (5.8)$$

5.2.1.4. Calculation of Cloth Translation

The translation phase of the simulation cycle simulates the cloth feeding past the needle without any rotation taking place. In terms of the cloth coordinates u and v, the needle moves from location N_1 to N_2. The distance $N_1 N_2$ is determined by the time delay δt, and the cloth speed V_c, as follows :-

$$N_1 N_2 = \delta s = V_c \ \delta t \qquad (5.9)$$

Refering to fig. 5-3, the new needle position, N_2, is given by :-

$$N_{2u} = N_{1u} - (N_1 N_2 \sin \alpha) \qquad (5.10)$$
$$N_{2v} = N_{2v} - (N_1 N_2 \cos \alpha)$$

The new cloth edge location, P_2, can be calculated from the cloth rotation equations (5.5) and (5.8).

5.2.1.5. Control Transfer Function, G_1

The control system's transfer function had the following form :-

$$\delta\alpha = K_3 E_s + K_4 \beta \tag{5.11}$$

Since the incidence angle, β, is in effect the derivative of the seam width error, E_s, the two constants, K_3 and K_4, are analogous to proportional and derivative gains, respectively.

The derivative component was clearly necessary, especially since $\delta\alpha$ directly affects the angle β and only indirectly affects E_s. Thus, the control system must act to minimize both E_s and β.

Initial simulation runs confirmed that an integral control component, which would improve steady state errors at the expense of stability margin, would not be beneficial since the primary control difficulty was stability and the steady-state errors were not critical.

5.2.1.6. Robot Motion Limitations

The preliminary experiments in controlling the PUMA 560 robot via the ALTER channel showed that the tool's velocity and acceleration had to be limited to less than 8 mm/hs and 3 mm/hs/hs respectively (section 3.5.1). In addition the robot's reach was limited to

$$-200 \text{ mm} < y_F < 200 \text{ mm} \tag{5.12}$$

where the main finger has coordinates (x_F, y_F)

For a given correction angle, $\delta\alpha$, the required displacement of the robot in the y direction is proportional to x_F, the finger to needle distance. Thus the limitations of the robot are more detrimental to seam width control for large values of x_F, i.e. when the robot is further away from the needle.

Thus, in the real system, the robot approaches the needle together with the cloth, so that the cloth can be rotated by larger angles towards the end of the seam. In the simulation program, x_F was held artificially constant, so that the effect of the robot's limitations would not vary during the simulation run. This measure facilitated the interpretation of the simulation results since the effect of other variables, such as curvature, could be more easily identified.

5.2.1.7. Simulation of Vision System

The simulation program was modified so that either two or one camera vision systems could be investigated. A camera was modelled as a linear array of pixels so that the pixel resolution and the number of pixels could be specified.

One camera was assumed to lie along the y axis in order to measure N_1P_1 directly. In a two camera system, the second camera was placed at a distance x_{CAM} in front of and parallel to the first camera, in order to measure the incident angle, β.

If a second camera was included, then, y_{CAM}, the y coordinate of the cloth edge at $x = x_{CAM}$, can be measured directly. In the simulation program, y_{CAM} was calculated from equations 5.10,5.5 and 5.8, by substituting x_{CAM} for N_1N_2 and y_{CAM} for N_2P_2. The angle ß was then calculated as follows :-

$$ß = \tan^{-1} \left(\frac{N_1P_1 - y_{CAM}}{x_{CAM}} \right) \qquad (5.13)$$

If only one camera was specified, then ß was estimated from the rate of change of the seam width :-

$$ß = \tan^{-1} \left(\frac{NP_{K-1} - NP_K}{\delta s} \right) \qquad (5.14)$$

where NP_K is the value of N_1P_1 for this time step
NP_{K-1} is the value of N_1P_1 for the previous time step

5.2.1.8. Graphic Output

The simulation program was extended to generate a graphical display of the seam width control in real time. This improved the usefulness of the program since parameters could be changed interactively and the results were displayed graphically within a few seconds.

Two examples of simulation runs are shown in figs. 5-6 and 5-7. Fig. 5-6 shows an excellent simulated seam produced with a two camera vision system, and fig. 5-7 shows an unstable control resulting from a one camera vision system.

The cloth edge and the ideal needle path, which are the outer and inner parabolas respectively, were plotted at the start of the run. At each time step, the line N_1P_1 is plotted. The P_1 end of these short lines always lies on the cloth edge, by definition. The other end represents the position of the needle, relative to the cloth contour, at the beginning of each time step. The variation of $\delta\alpha$ is clearly visible from the gradient and the seam width error, E_s, is shown by the perpendicular distance between the needle position and the ideal needle path.

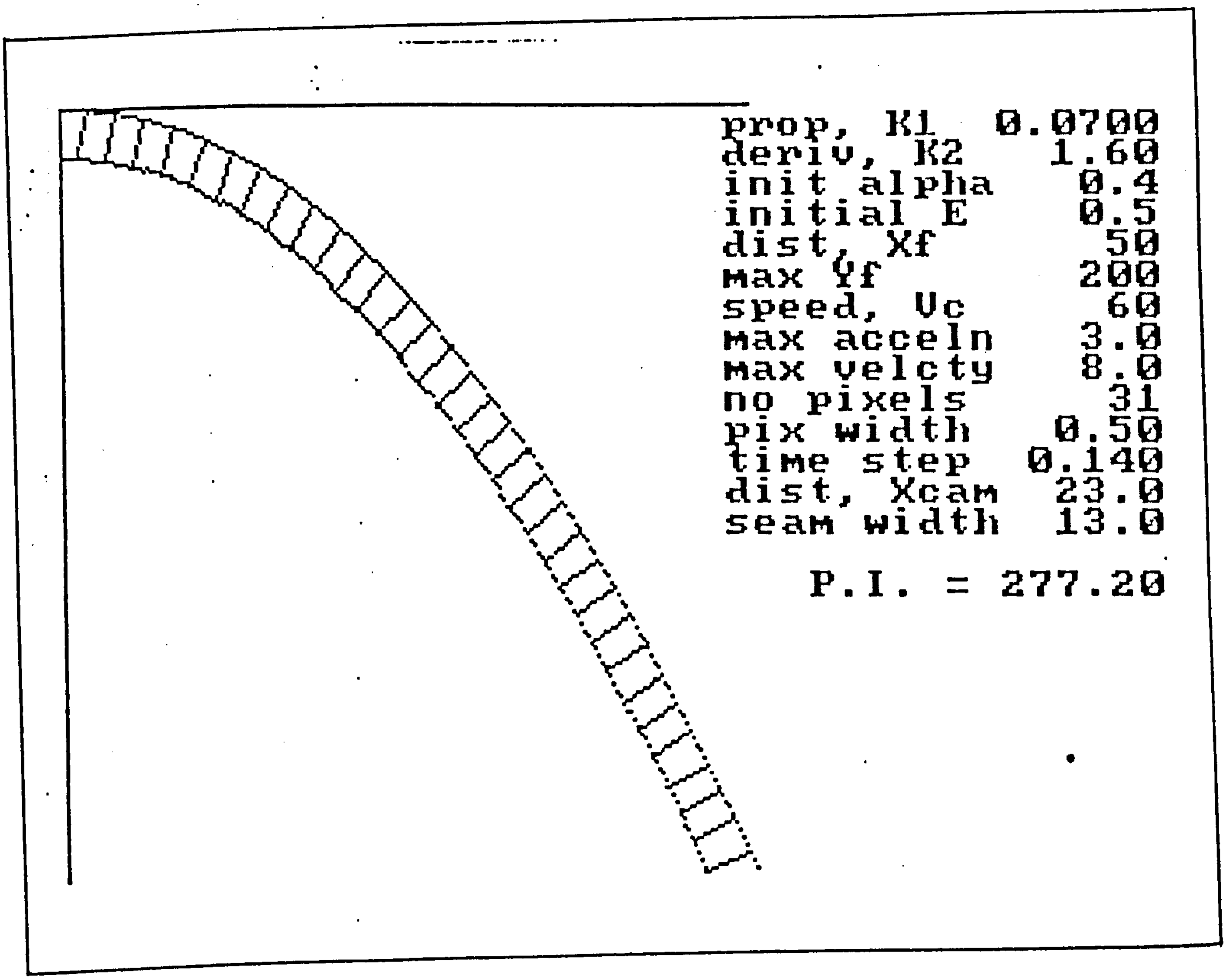


Fig. 5-6: Simulation Plot for Two Camera System

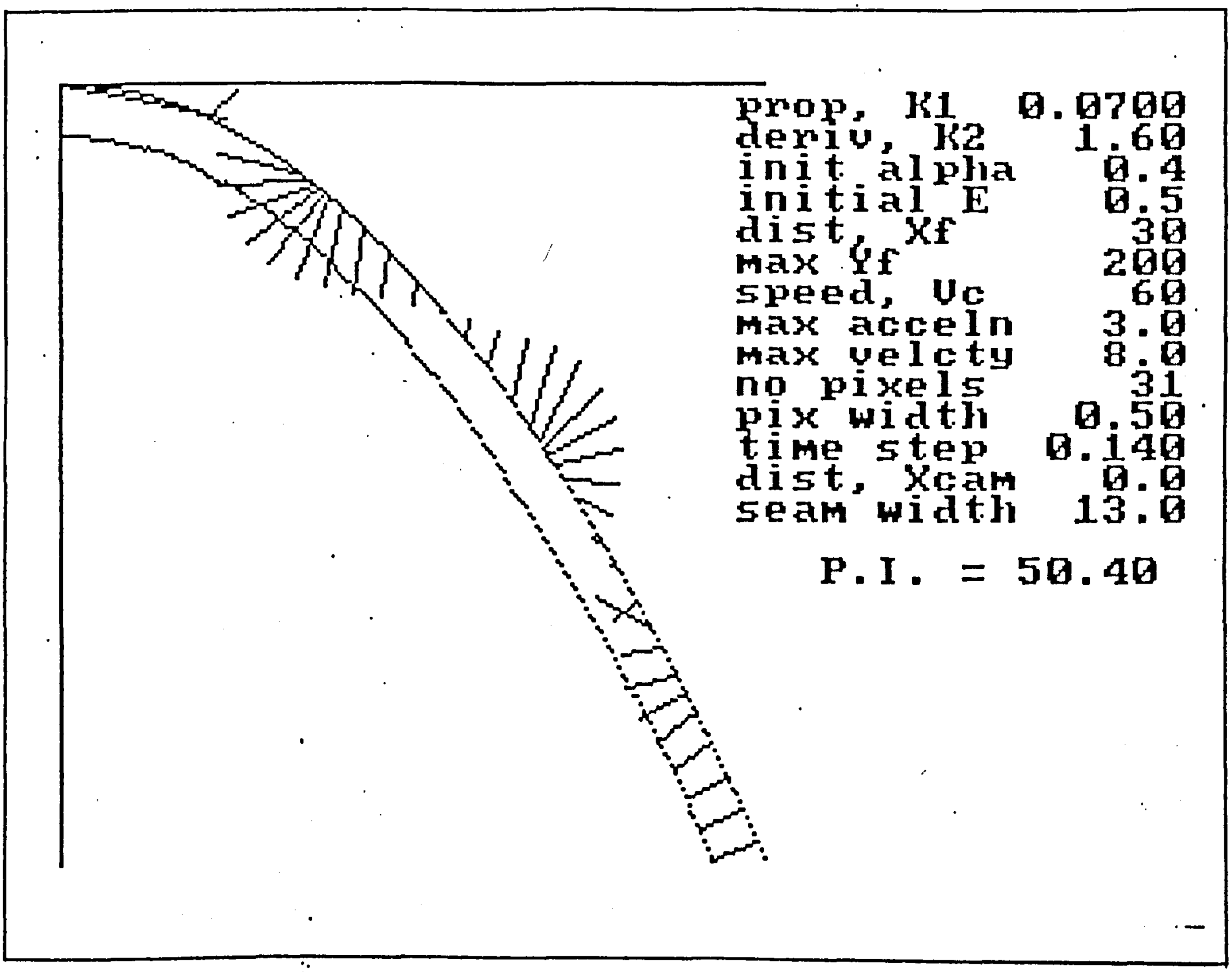


Fig. 5-7: Simulation Plot for One Camera System

5.2.2. Simulation Experiments

5.2.2.1. Performance Index (P.I.)

The seam contour function, $200\ v = u^2$ used in the simulation was chosen because the curvature of the contour gradually increased as the sewing progressed. A convenient measure of control system performance was the distance sewn before the seam error exceeded 1 mm. The initial values of the seam width error and alpha were set at 0.5 mm and 0.4 radians, respectively, for all the simulation runs. The initial v coordinate of the needle was set at 199 mm.

5.2.2.2. Photocell and One Camera Systems

The one camera system was found to be unstable, under all circumstances (fig. 5-7). The use of one or two photocells, in place of the two cameras, was investigated, and was also found to be insufficient.

5.2.2.3. Performance of the Ideal System

Fig. 5-8 shows performance plots for four sewing speeds for the ideal system, i.e without vision system or robot motion limitations. Each plot shows the maximum variation in K_3 and K_4 for a specific value of performance index. The parameter settings for the simulation runs that produced these performance plots are listed in table 5-3.

The system's stability margin is sensitive to the distance sewn during the system time delay, δs, which is dependent on both the sewing speed, V_c, and on the system time delay, δt, (equation 5.9). Thus, if δt is increased then V_c must be decreased before the same performance is obtained, and vice versa.

Variation in the desired seam width, R_s, had only a minor effect on the control system. Large initial values of seam width error or incidence angle, β, gave rise to instability.

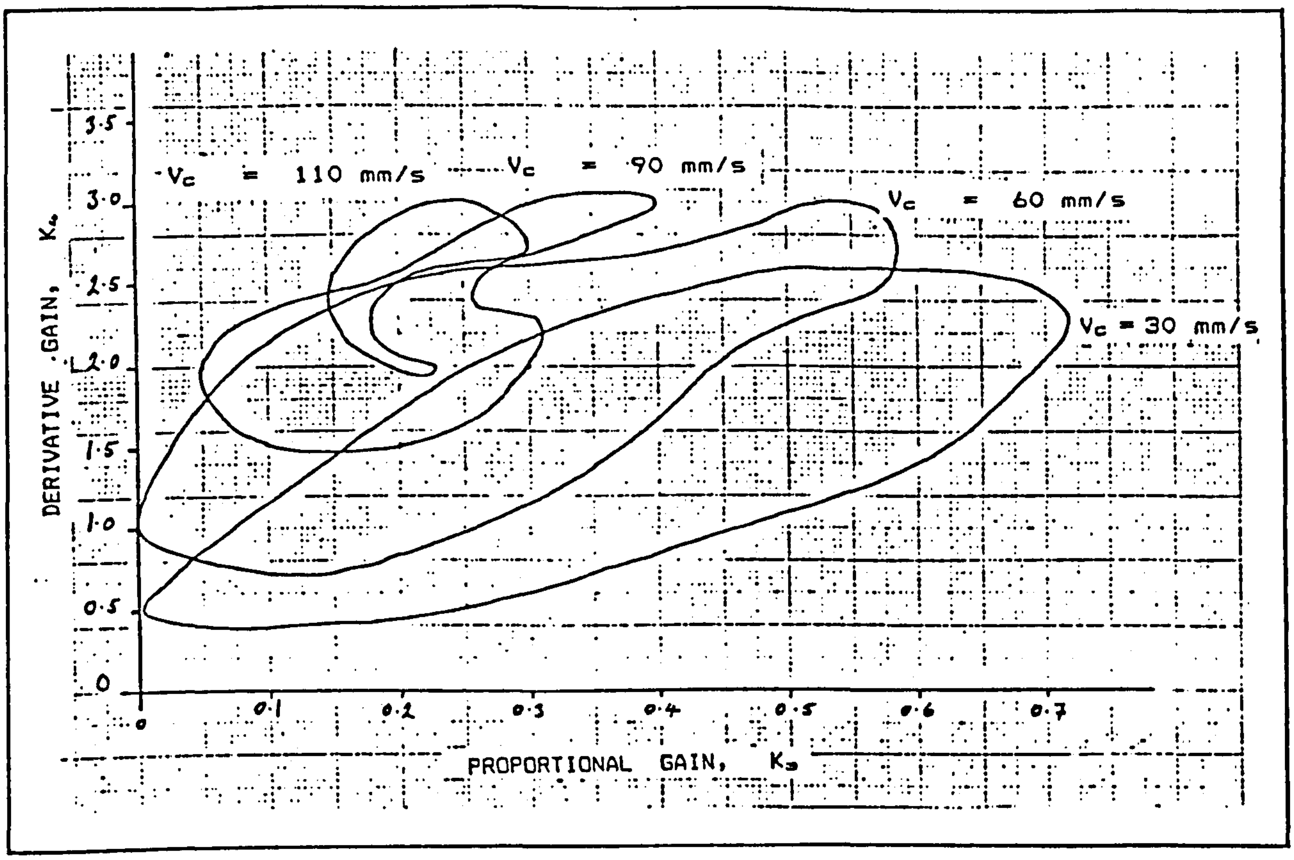


Fig. 5-8: Effect of Speed on Simulated Seam Width Control

Parameter	fig 5-8	fig 5-9	fig 5-10	fig 5-11
No. of pixels	71	31	31	31
Pixel width (mm)	0.5		0.5	0.5
Dist, x_F (mm)	3	3	3	
Delay, δt (s)	0.14	0.14	0.14	0.14
Dist, x_{CAM} (mm)	23	23		23
Speed, V_c (mm/s)		60	60	60
Perf. index, PI	268	268	268	208
Seam width (mm)	13	13	13	13
Max. acceln	3	3	3	3
Max. velocity	8	8	8	8
Max y_F (mm)	200	200	200	200

Table 5-3: Parameter Values for Simulation Tests

5.2.2.4. Vision System Limitations

Fig. 5-9 shows the effect of pixel resolution on seam width control performance. Increasing the pixel resolution (by reducing the pixel width) significantly improved the system's stability for high proportional gains, but slightly reduced the stability margin for high derivative gains.

Increasing the length of the pixel array above 8 mm, had negligible effect on the system's performance.

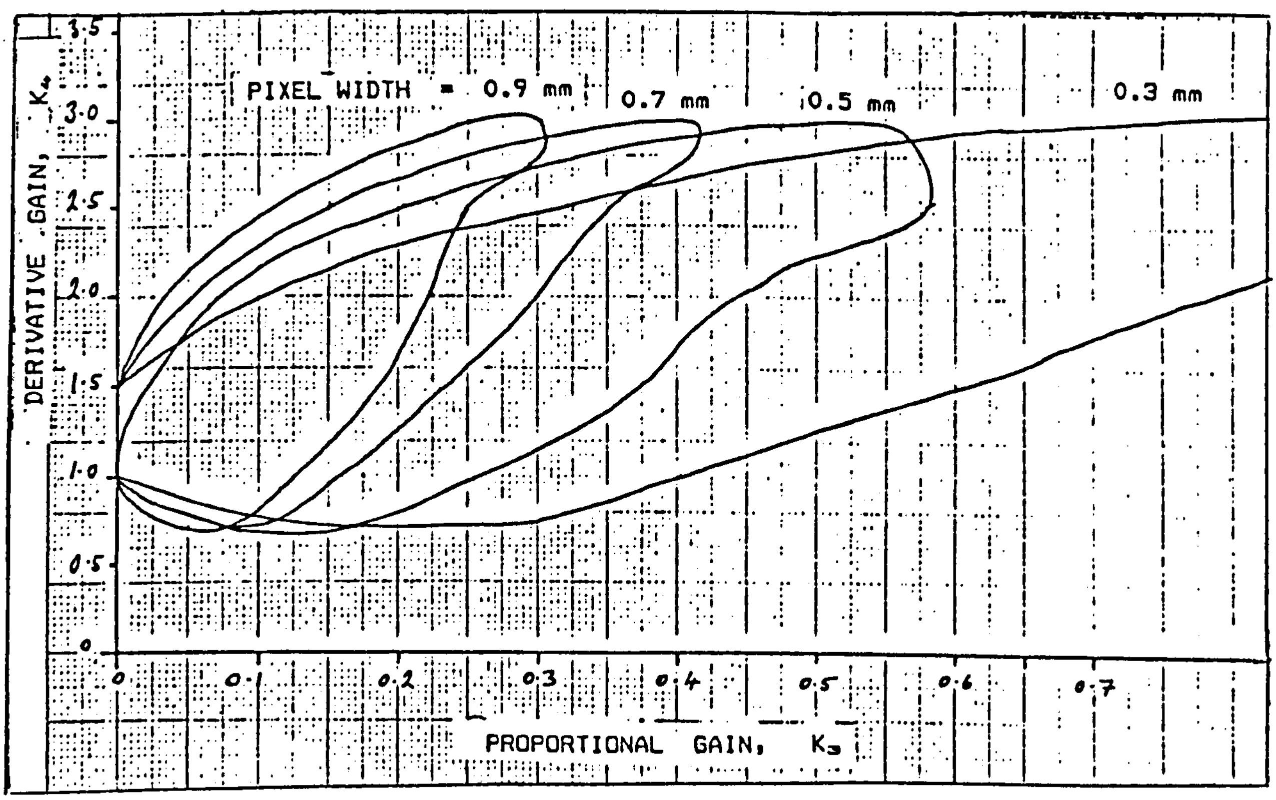


Fig. 5-9: Effect of Pixel Resolution on Simulated Seam Width Control

Fig. 5-10 shows the effect of varying x_{CAM}, the distance between the two cameras, on the system's performance. The optimum distance was found to be between 20 and 30 mm. Performance was impaired for smaller values of x_{CAM} because the accuracy of measuring the angle ß was affected. Larger values of x_{CAM} affected the accuracy of calculating the angle ß from the measurements, because the calculation was based on a straight line assumption (equation 5.13).

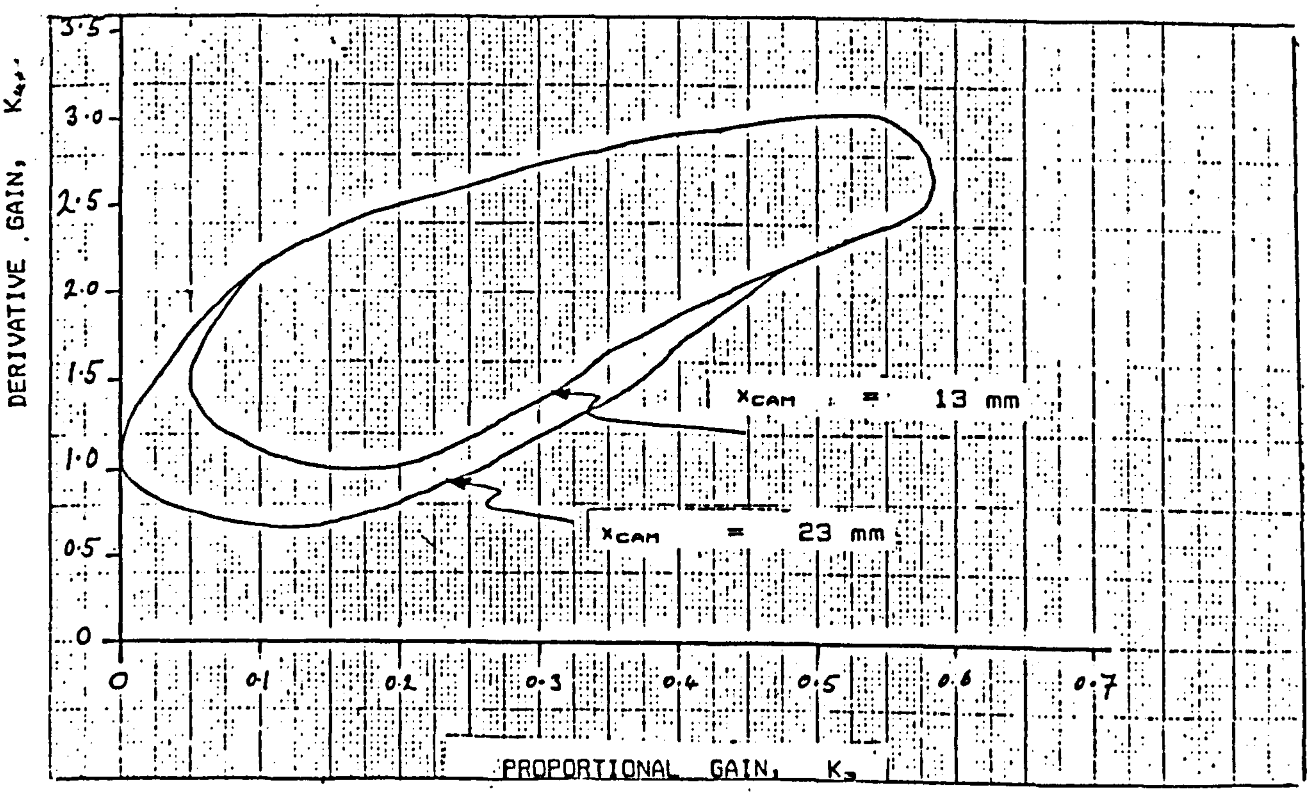


Fig. 5-10: Effect of x_{CAM} Seam Width Control

5.2.2.5. Robot Motion Limitations

All the performance plots shown in figs. 5-8 to 5-10, were based on a performance index of 268 (section 5.2.2.1.), which corresponds to sewing accurately round almost the entire contour up to the origin. However, once the robot's limited reach capability (equation 5.12) was included in the system model, the system could no longer follow the extreme curvature of the contour in the region of the origin.

When the maximum reach limitation was introduced into the simulation program, but without the dynamic robot motion limitations, the maximum performance index obtained decreased as x_r (the robot to needle distance), was increased. Obviously, this effect was due to the limit that the robot can rotate the cloth.

Fig. 5-11 shows the effect of the robot's dynamic motion limitations (i.e. maximum acceleration and velocity) on system performance. The performance plots are based on a performance index of 208, which is more realistic for a real robot with limited reach, since the tangential angle of the edge contour does not exceed the maximum rotation angle of the robot about the needle for the values of x_r considered.

Although the acceleration and velocity limitations were fixed to 3 mm/hs/hs and 8 mm/hs respectively, the performance was plotted against x_r, since the effect of these dynamic limitations on the angular acceleration and velocity of the cloth was dependent on x_r. For small values of x_r, the dynamic limitations have very little affect on the robot motion, but at large values of x_r, they severely damp down the cloth's rotational motion.

As clearly shown in fig. 5-11, the dynamic motion limitations improved the stability margin of the system for large gain values by damping down excessive robot motions. By preventing the high gain values from generating excessive robot motion, these limitations are keeping the effective system gain within a stable region.

The dashed section of the performance curve for x_r = 200 mm denotes an untested region. The curves were plotted using a modified version of the simulation program which automatically found the minimum and maximum values of K_4 for a particular value of K_3. The search for minimum and maximum values of K_4 had been limited to below 7.

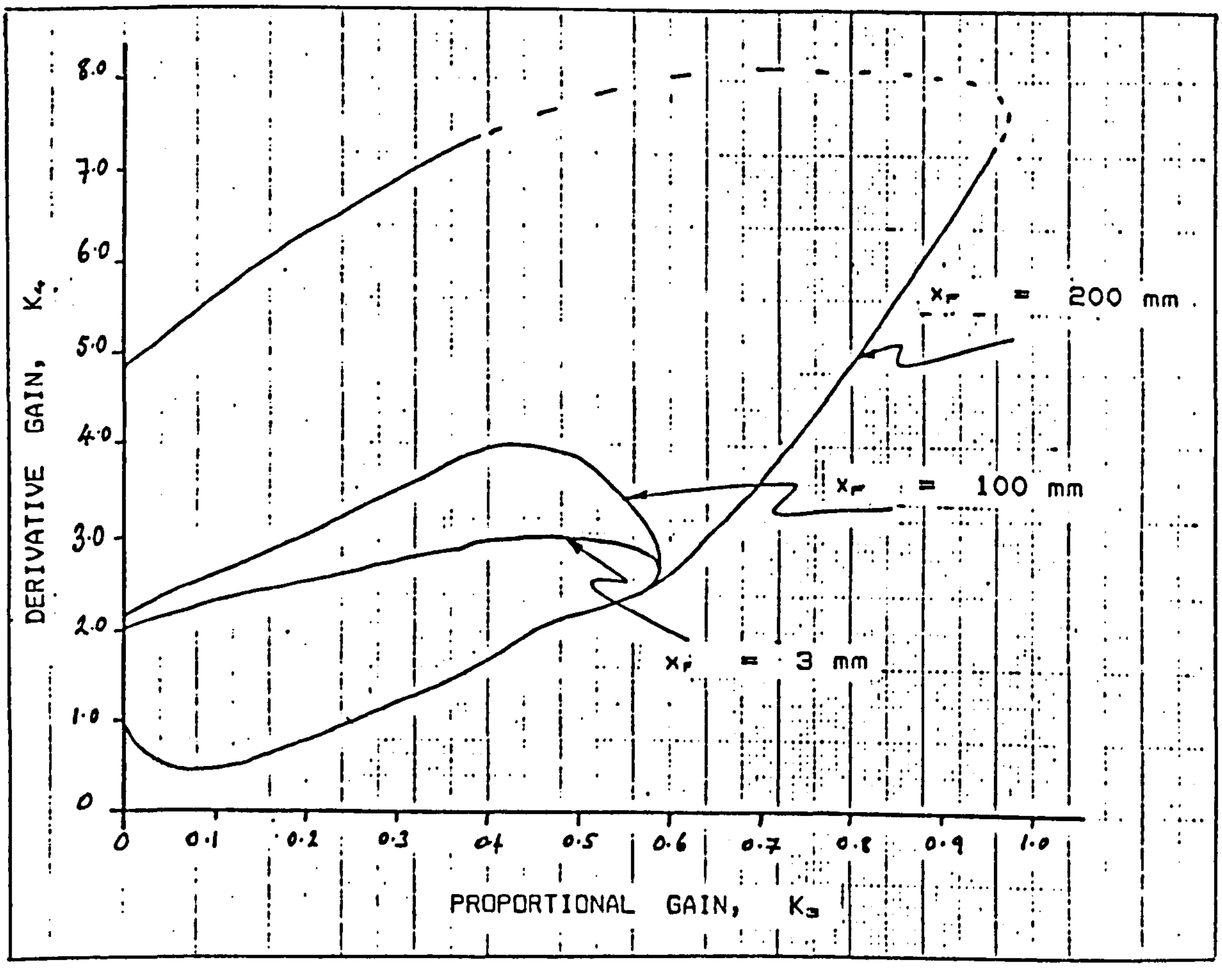


Fig. 5-11: Effect of x_r on Simulated Seam Width Control

5.2.3. Conclusions

The simulation program was a valuable aid in understanding the system's control problems and limitations. The following conclusions were made from the simulation experiments :-

a) Stable control could not be obtained using one or two photocells, or using only one camera.

b) Stable control could be obtained using two linear array cameras. The I-SIGHT cameras, which were proposed for the FIGARO application and are described later, were shown to provide satisfactory control performance under simulation.

c) The performance of the seam width control is very sensitive to system time delay, and the maximum sewing speed is primarily limited by the system time delay.

d) The maximum curvature that could be tracked was dependent on the robot's reach limitations and on x_F, the robot to needle distance. The maximum tangential angle of an edge contour that can be accommodated is given approximately by :-

$$\tan^{-1} ((200 - y_{F1})/x_F) \qquad (5.15)$$

where y_{F1} is the y coordinate of the main finger's initial position, at the start of the seam.

e) The robot's acceleration and velocity limitations reduced the system's sensitivity to high values of K_3 and K_4, by keeping the effective gain values low.

f) The initial seam width error and incidence angle should be kept to a minimum.

g) The two cameras should be placed between 20 and 30 mm apart.

5.3. Vision System

The simulation program confirmed that the vision system had to have the following specification :

* high speed operation (to limit system time delays)
* two cameras
* a pixel resolution of at least 0.5 mm in the object plane
* a pixel array length of at least 8 mm in the object plane

5.3.1. Cameras

Two I-SIGHT cameras were installed on the sewing machine, as shown in fig. 5-12. Each camera has a 32 X 30 pixel array and their proprietary mode of operation is similar to that of CCD cameras. These cameras were chosen because of their small physical dimensions which permitted direct attachment to the sewing machine.

Although there were few pixels per row, this crude camera resolution was compensated by their close proximity to the table surface, so that an object plane resolution of 0.5 mm was easily achieved. Furthermore, since the processing time associated with the vision system is proportional to the number of pixels, the relatively small pixel array size resulted in low system time delay.

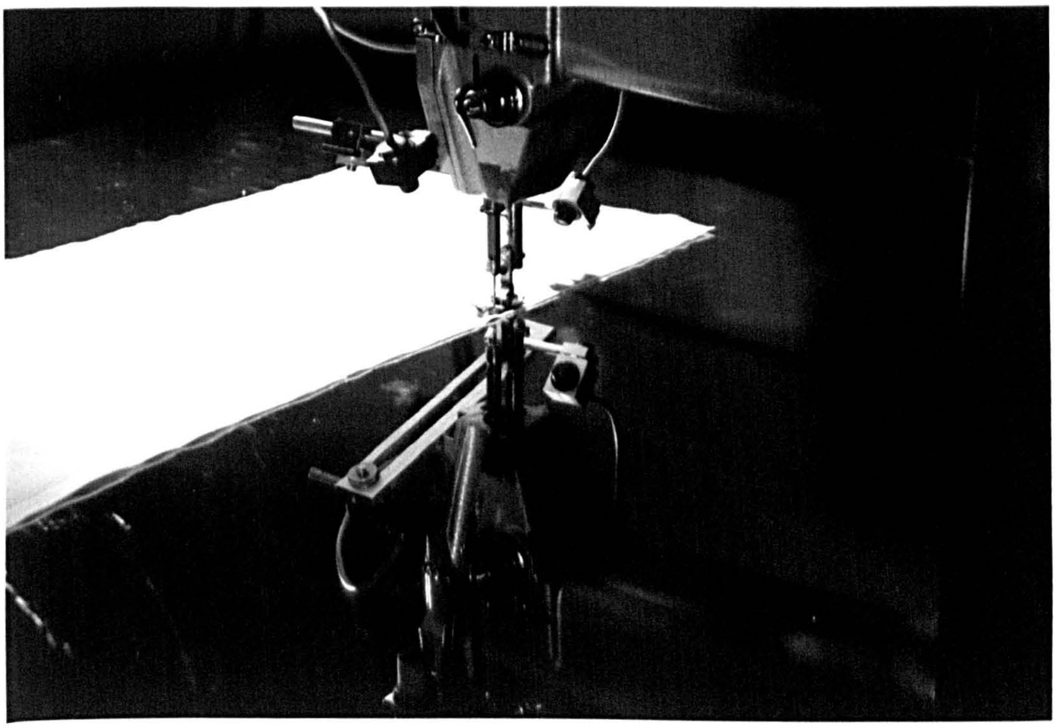


Fig. 5-12: The I-SIGHT Cameras Mounted on the Sewing Machine

The cameras operated in a binary mode only, i.e. a pixel could be only black or white, and gray levels could not be differentiated. The threshold between black and white was determined by specifying an exposure value (between 0 and 127) which controlled the camera's exposure time interval.

The cameras are focused by rotating the lens in the camera body. An advantage of selecting cameras with a crude resolution is that the depth of field is increased and therefore they do not require accurate focusing [62].

5.3.2. Interface to IBM AT

The manufacturer's of the I-SIGHT camera, Electronic Automation Ltd, provided an interface card which linked the cameras to the IBM AT. The card, which was installed directly in the IBM AT bus, contained a Z80 microprocessor, an EPROM and a block of dual ported RAM, in addition to the necessary digitizing hardware for the cameras.

The Z80 performed the frame grabbing and thresholding operations, thus reducing the vision system overheads of the IBM AT. The block of dual ported RAM constituted the frame stores for two cameras.

The IBM AT initiated a frame grabbing cycle by setting a hardware flag to the Z80. Utilizing high speed DMA transfers, the Z80 loaded the frame stores with the digitized and thresholded pixel data. When the Z80 had completed its operation, it signalled the IBM AT through another flag. The IBM AT then requested the Z80 to relinquish the data bus to the frame stores and it then transferred the pixel data to its internal RAM.

More details of the operation of the interface card are to be found in reference [61].

5.3.3. Lighting Arrangement

The lighting arrangement for the cameras, shown diagrammatically in fig. 5-13, comprised a projection lamp and the mirror surface of the table. The cameras were mounted vertically above the cloth edge and the lamp was directed to shine a pool of light on the field of view at an angle of about 45° to the table surface.

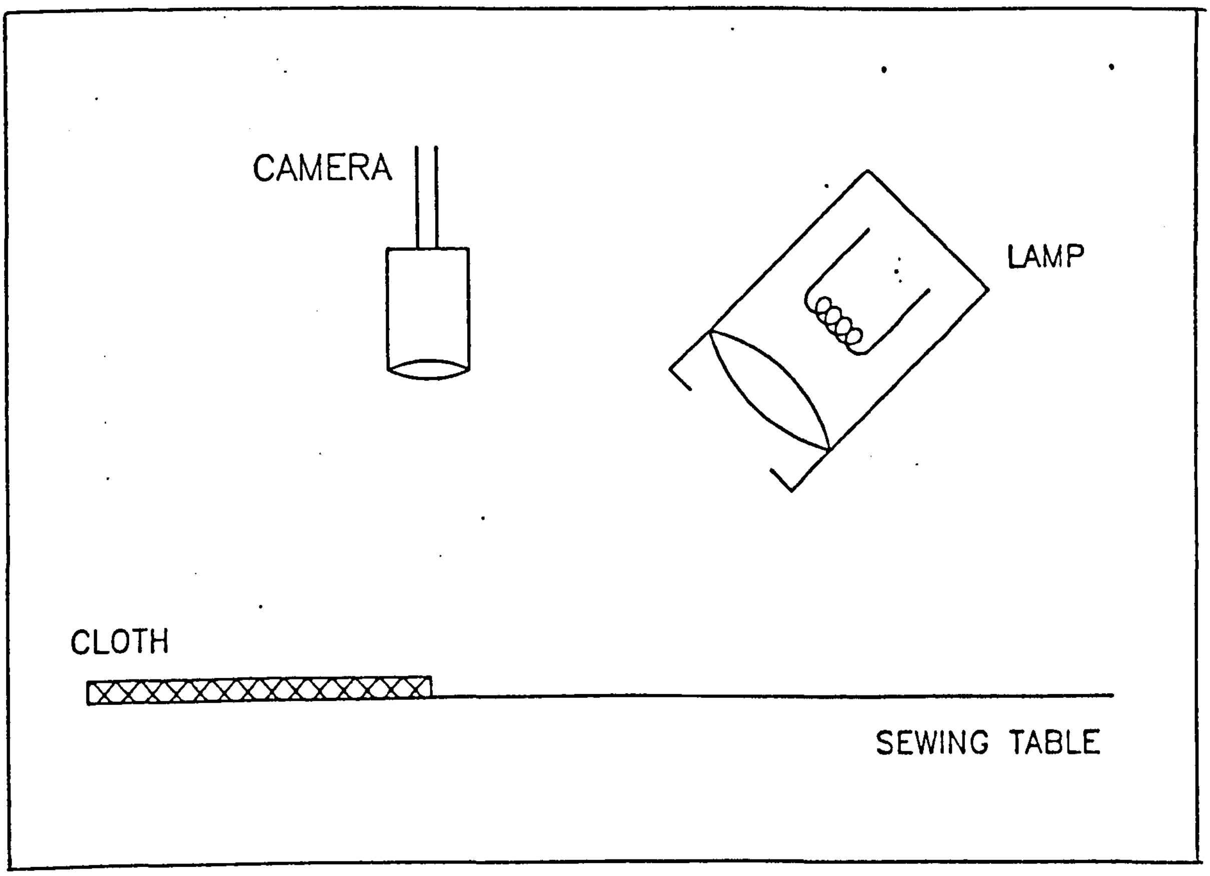


Fig. 5-13: Lighting Arrangement

When there was no cloth in the field of view then the mirror surface reflected the light away from the cameras and the image was black. When the cloth was present, the light was dispersed by the cloth and the camera image was white. Although this lighting arrangement was effective with all kinds and colours of fabric, darker materials required longer exposure times since they absorbed more of the light and dispersed less. Satisfactory images were obtained for white material for an exposure value of 10.

Unwanted reflections, which caused false images, were avoided by careful positioning of the lamp and by painting some of the polished surfaces black, such as the presser foot.

5.3.4. Projection Lamp

The I-SIGHT cameras required a lighting system that provided an intense and uniform pool of light with high infra-red content, that covered both fields of view. A normal filament bulb and reflector system was found to be unsuitable, since the filament created bright spots on the illuminated object. High quality projection lamps include a condenser lens, which ensures that a uniform pool of light is produced. A 48 W high intensity lamp, with an iris diaphragm and focusing condenser assembly, was selected for the FIGARO system.

The I-SIGHT cameras are only sensitive to a narrow band of light (approximately 820 nm wavelength) in the infra-red portion of the spectrum, and they produce clearer and more stable images when the object is illuminated by an 820 nm laser beam. Although, laser illumination was not implemented in the FIGARO prototype, it has been used in some industrial applications of these cameras [63].

5.3.5. Software Implementation

The slave processor architecture of the camera interface card permitted the IBM AT to perform its real time processing of sensory data simultaneously with the frame gabbing operation.

The image of the cloth edge captured by the cameras was quite noisy even when a clean edge was viewed. The image of the cloth edge would fluctuate by one or two pixels. Since the cameras provided a two dimensional array of pixels, the position of the cloth edge at $x = 0$ and at $x = x_{CAM}$ were measured by averaging the edge locations taken at three adjacent pixel rows. This technique provided a more accurate and stable measure of the position of the cloth edge.

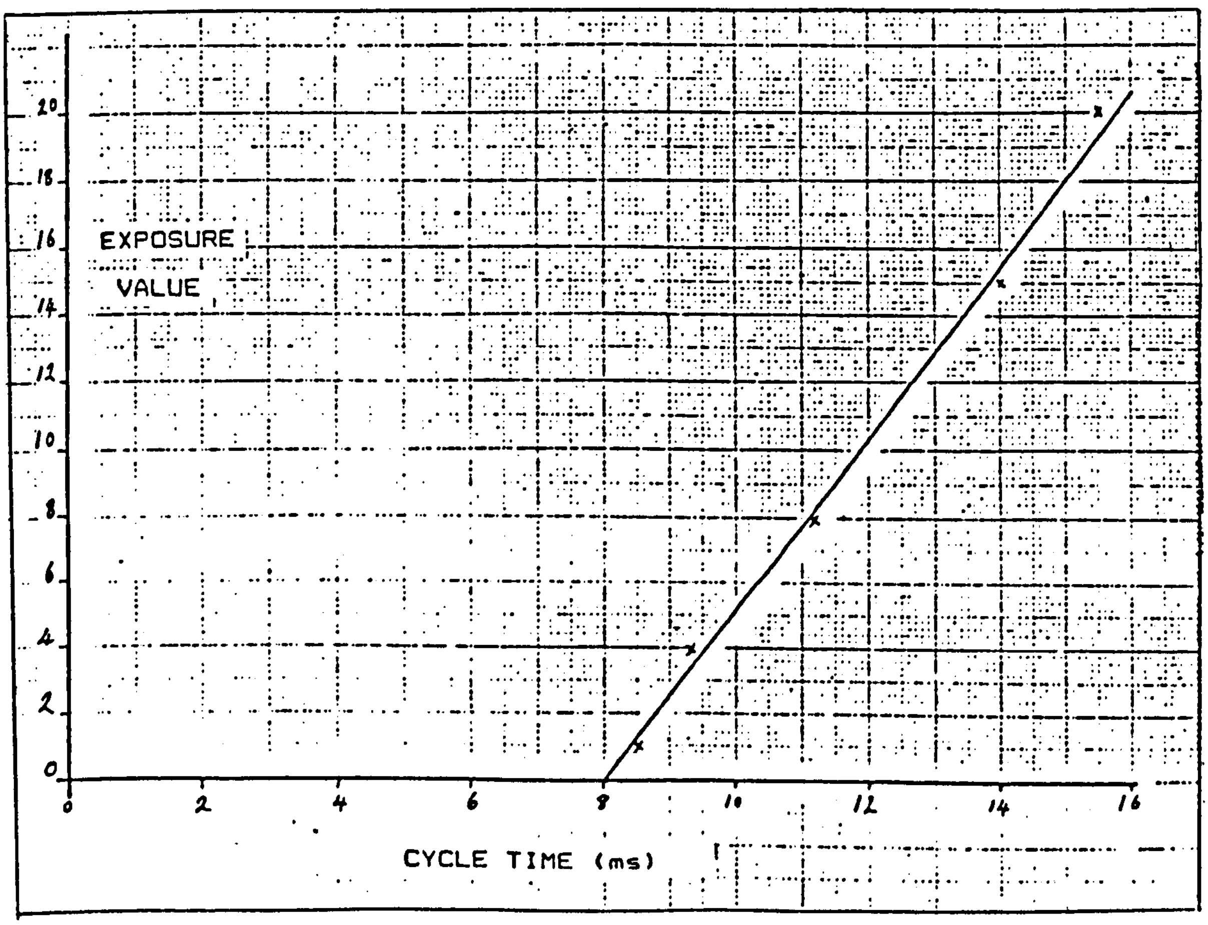


Fig. 5-14: Vision Processing Time vs Camera Exposure Value

The pixel data were transferred from the I-SIGHT card to the IBM RAM using a high speed hardware block move, and the routines for finding the cloth edge and calculating E_s and β were optimized for fast execution.

The time taken by the combined system to grab the two frames, process the pixel data and calculate E_s and β was measured for different camera exposure values and the results are shown in fig. 5-14. The cameras were usually set at an exposure value of 10, for which the vision processing time was approximately 11 ms.

5.3.6. Calibration Technique

The accuracy of measurements based on the camera data depended on careful calibration of the vision system. In particular the seam width control was very sensitive to misalignment of the two cameras. Since accurate alignment and positioning of the cameras' field of view was difficult and time consuming to do manually, a calibration technique was developed in which the true position of the field of view of each camera was accurately measured in terms of pixel offsets from the ideal position. These offsets were then entered as factors into the robotic sewing program which used them to calculate accurate values of E_s and β from the camera data.

The calibration procedure, which involved a calibration program and two calibration overlays shown in fig. 5-15, consisted of the following steps :-

a) Place the large overlay on the table and, using the sewing needle and alignment marks, accurate locate it over the cameras' field of view.

b) View the camera images on the screen using the calibration program. If the cameras' fields of view are grossly in error, make manual adjustments to the position and orientation of the cameras. Fine adjustments are not necessary.

c) From the statistical data displayed on the screen, record the row numbers that correspond to the $x = 0$ and $x = x_{CAM}$ coordinates, and the column numbers that correspond to $y = R$.

d) Place the small overlay in the field of view of each camera and align it using the displayed image. Record the slot width of the image in pixels displayed on the screen, and hence calculate the pixel resolution.

e) Enter the calibration data into the robotic sewing program.

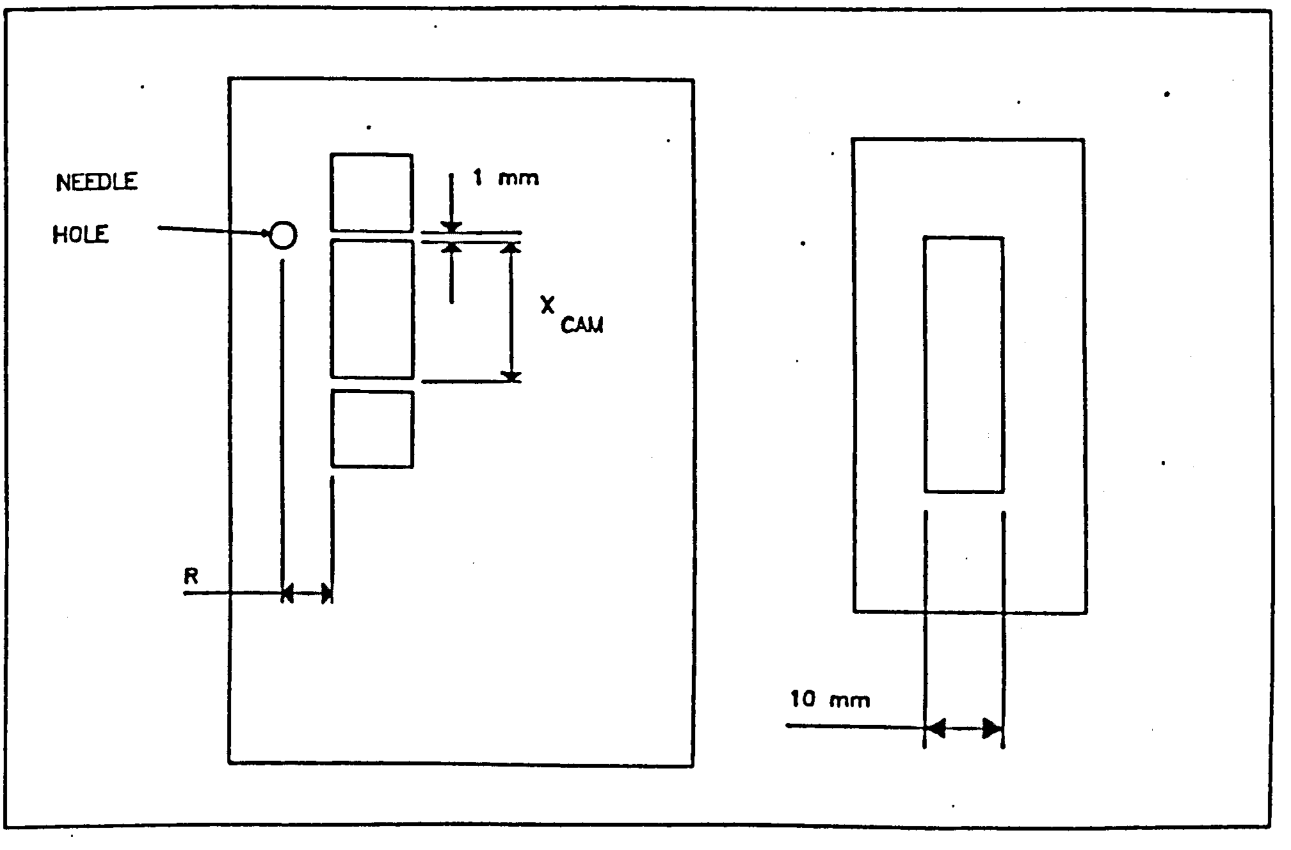


Fig. 5-15: Overlays Used in Vision System Calibration

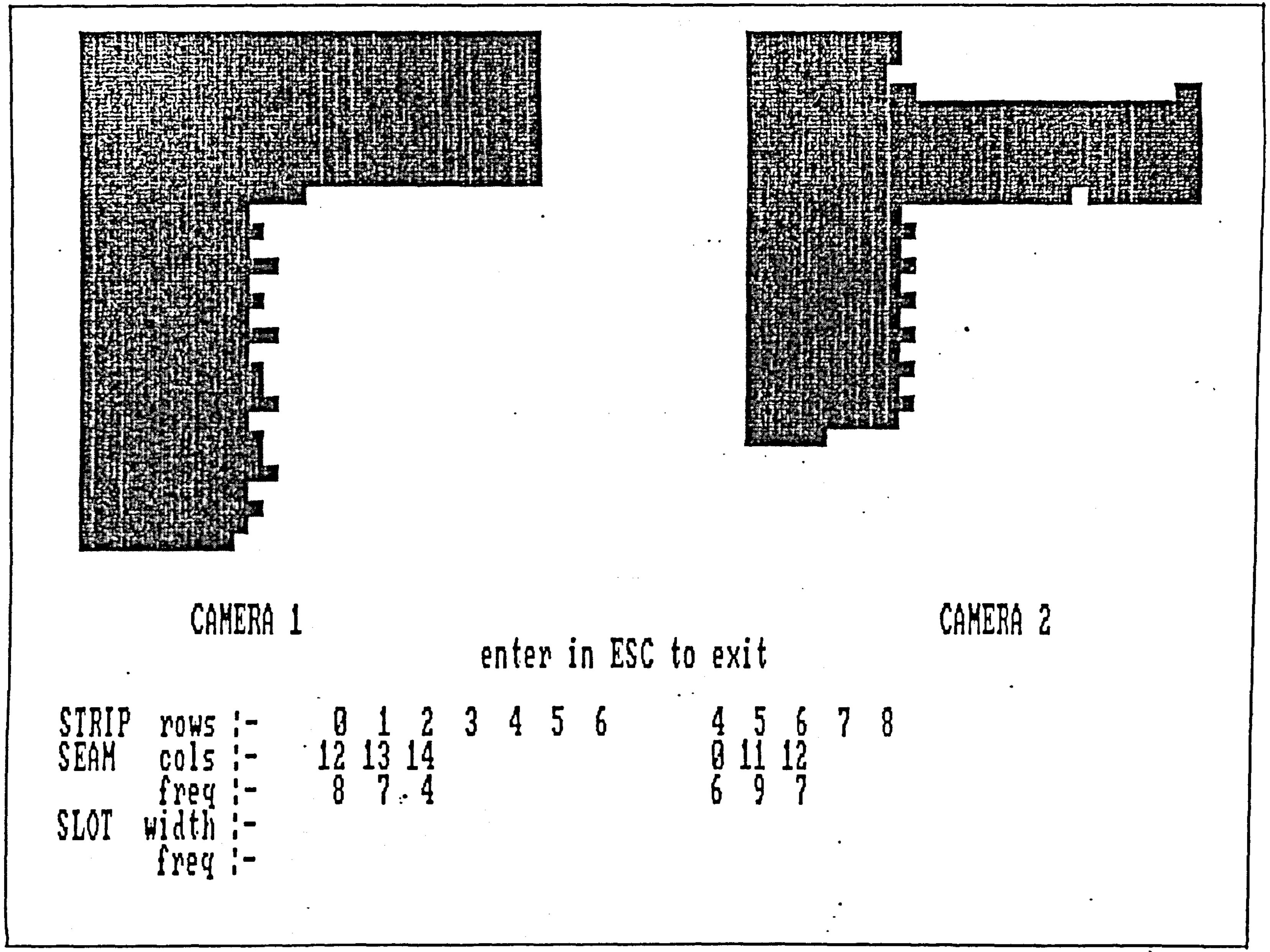


Fig. 5-16: Calibration Program Display - Large Overlay

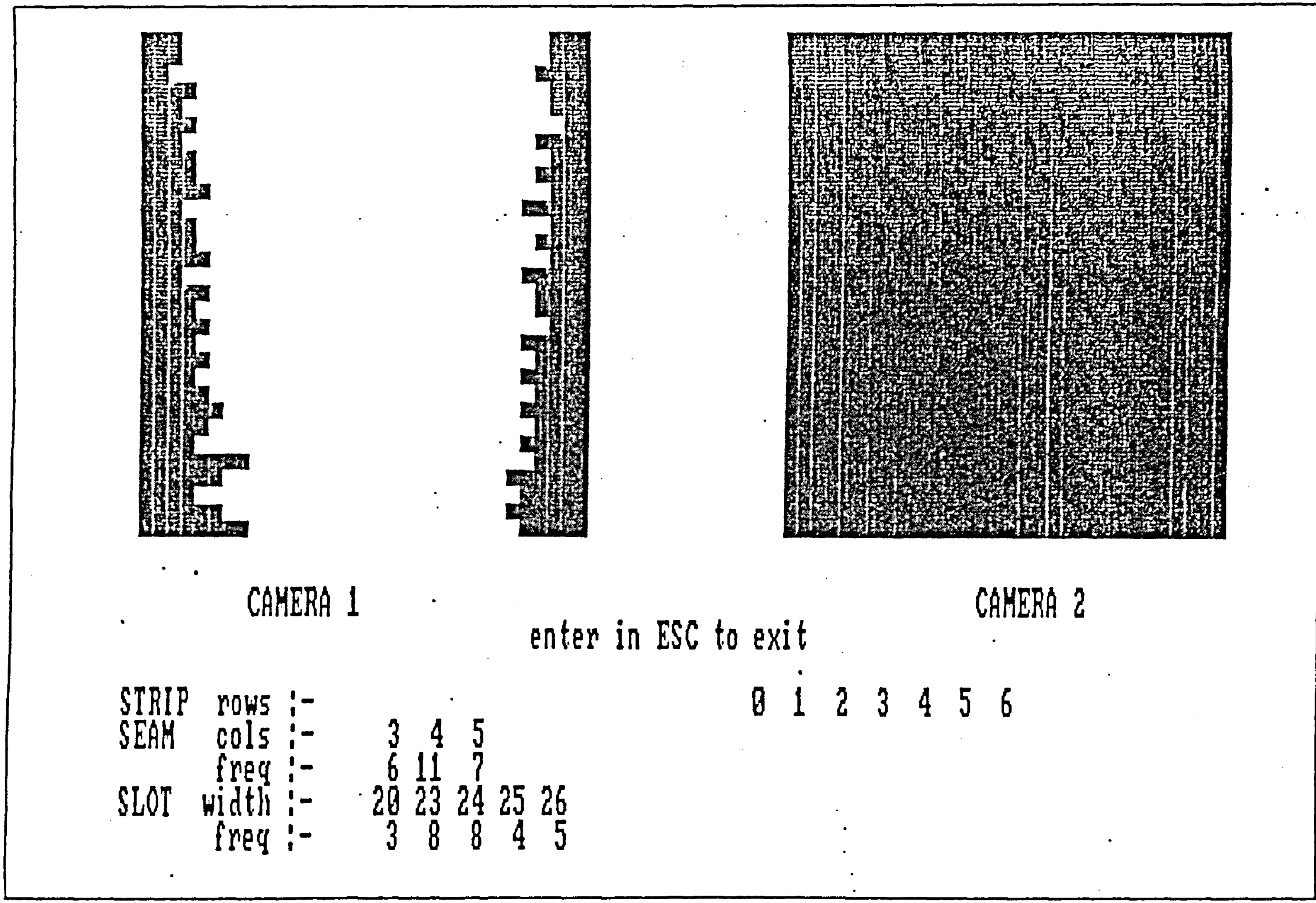


Fig. 5-17: Calibration Program Display - Small Overlay in Camera 1

The calibration program is listed and explained in Appendix F. A typical display of the calibration program for each of the two overlays is shown in figs. 5-16 and 5-17. Using the statistical data shown in figs. 5-16 and 5-17, the vision system parameters would be set as follows :-

irow1	=	3	-	row no. along line x = 0
irow2	=	6	-	row no. along line $x = x_{CAM}$
pix1_offst	=	2	-	offset in pixels, from $y = R_s$
pix2_offst	=	3	-	offset for Camera 2
y1_pixel	=	10/24	-	pixel resolution in y direction
	=	0.42 mm		for Camera 1

5.3.7. Vision System Performance

The I-SIGHT/IBM vision system that was integrated into the FIGARO system was in laboratory prototype form only. Towards the end of the project, the manufacturer admitted continued difficulties in debugging the product, and the delivery of the final production system was delayed indefinitely.

Two problems seriously affected the performance of the prototype vision system :-

1) The hardware that refreshes the CCD chip before a new picture is captured, appeared to be only partially effective. It took between 2 and 4 attempts at taking a new picture before the pixel data reflected changes in the field of view. This delay was observed for both light to dark and dark to light transitions.

2) Camera 2 generated only a partial image, and the extent of the image varied with the amount of light in

the field of view. Thus, in fig. 5-16, the bottom half of the overlay is missing, but in fig. 5-17, the image is complete due to the brightness of the field of view.

The second problem was minimized by aiming the camera so that only the line x_{CAM} passed through the top third of the image. The first problem effectively increased the system time delay (section 5.6.3).

5.4. Implementation of Seam Width Control

The simulation program assumed that the cloth panel remained rigid throughout the sewing operation. However, fabric panels exhibit very low lateral rigidity, and buckling of the cloth was the prime difficulty in implementing the seam width control system.

5.4.1. Calculation of Robot Motion to Rotate Cloth

The seam width control required that the robot corrected the orientation of the cloth panel during sewing. This was achieved by superimposing two motion elements; rotation of the main finger about the sewing needle, and rotation of the auxiliary finger about the main finger.

In fig. 5-18 the cloth is to be rotated about the needle by an angle $\delta\alpha$. The geometry clearly shows that the main finger should be rotated about the needle by $\delta\alpha$, and the auxiliary finger must be rotated about the main finger by the same angle, $\delta\alpha$, (see fig. 5-18).

The ALTER data for rotating the main finger about the needle were calculated using the equations derived below.

Consider rotation of the main finger from F_1 to F_2 about the needle, N, (see fig. 5-18). The coordinates of F_1, x_1 and y_1, are known, and the coordinates of F_2 are calculated as follows :-

$$y_2 = NF_2 \sin(\alpha + \delta\alpha)$$
$$= NF_2 \left(\sin\alpha \cos\delta\alpha + \cos\alpha \sin\delta\alpha \right)$$

Applying small angle approximations for $\delta\alpha$,

$$y_2 = NF_2 \left(\sin\alpha + \delta\alpha \cos\alpha \right)$$
$$= NF_2 \left(\frac{y_1}{NF_1} + \frac{\delta\alpha\, x_1}{NF_1} \right)$$

No buckling condition requires that $NF_1 = NF_2$, therefore

$$y_2 = y_1 + \delta\alpha\, x_1 \qquad (5.16)$$

Similarly,

$$x_2 = NF_2 \cos(\alpha + \delta\alpha)$$
$$= NF_2 \left(\cos\alpha \cos\delta\alpha - \sin\delta\alpha \sin\alpha \right)$$
$$= NF_2 \left(\cos\alpha - \delta\alpha \sin\alpha \right)$$
$$= NF_2 \left(\frac{x_1}{NF_1} - \frac{\delta\alpha\, y_1}{NF_1} \right)$$

which simplifies to

$$x_2 = x_1 - \delta\alpha\, y_1 \qquad (5.17)$$

Thus, three ALTER components were necessary in order to rotate the cloth about the needle :-

$$x \text{ increment} = x_2 - x_1$$

$$y \text{ increment} = y_2 - y_1$$

$$\text{Rotation about } z \text{ increment} = \delta\alpha$$

The x increment was superimposed on top of the x ALTER data due to the cloth tension control (section 4.4.1.2).

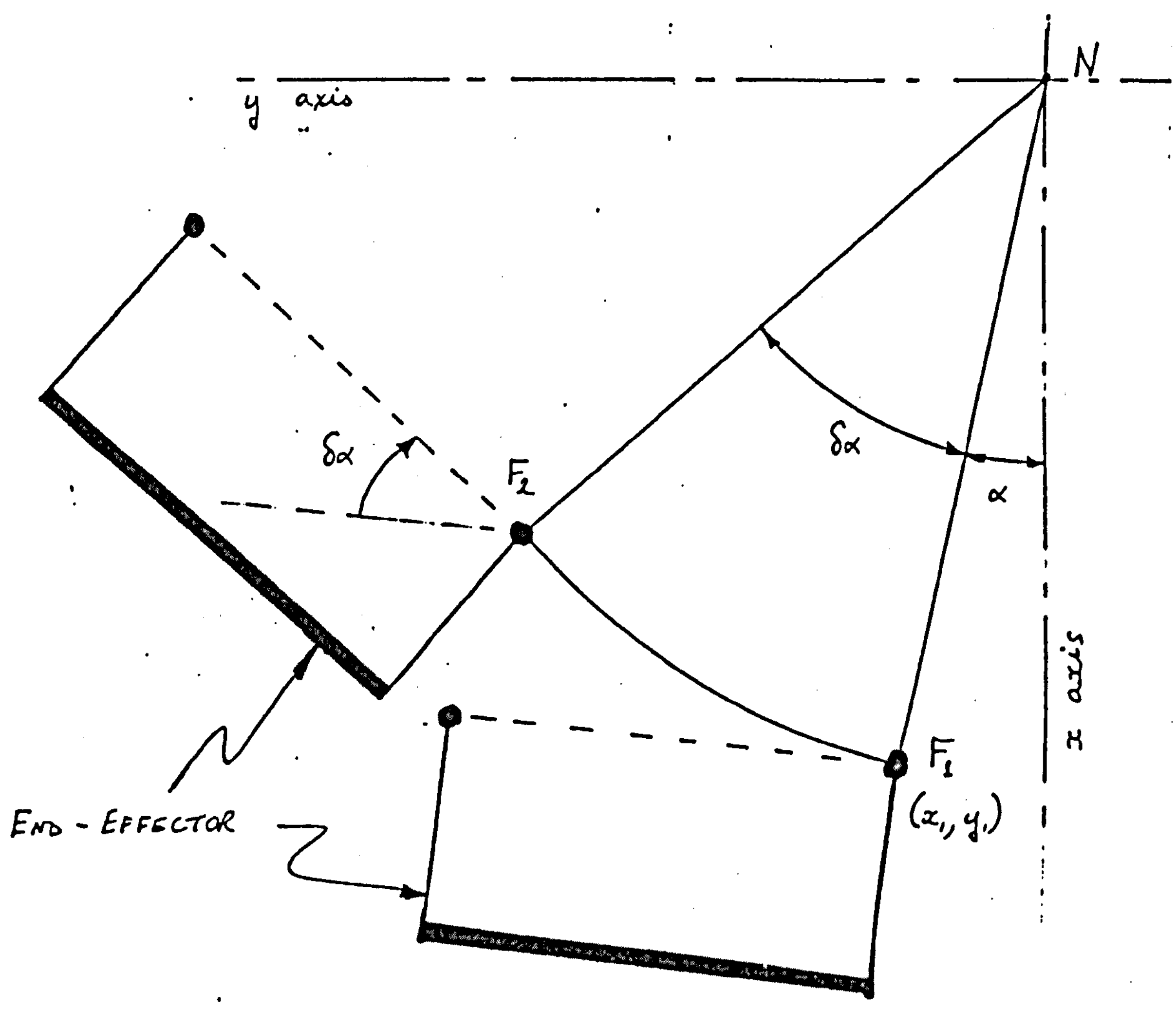


Fig. 5-18: Robot Motion Required to Rotate Cloth About Needle

5.4.2. Robot Reach Limitations

In addition to the acceleration and velocity limitations discussed in section 3.5.1, the ALTER data had to be limited so that the robot was not directed beyond a safe envelope boundary.

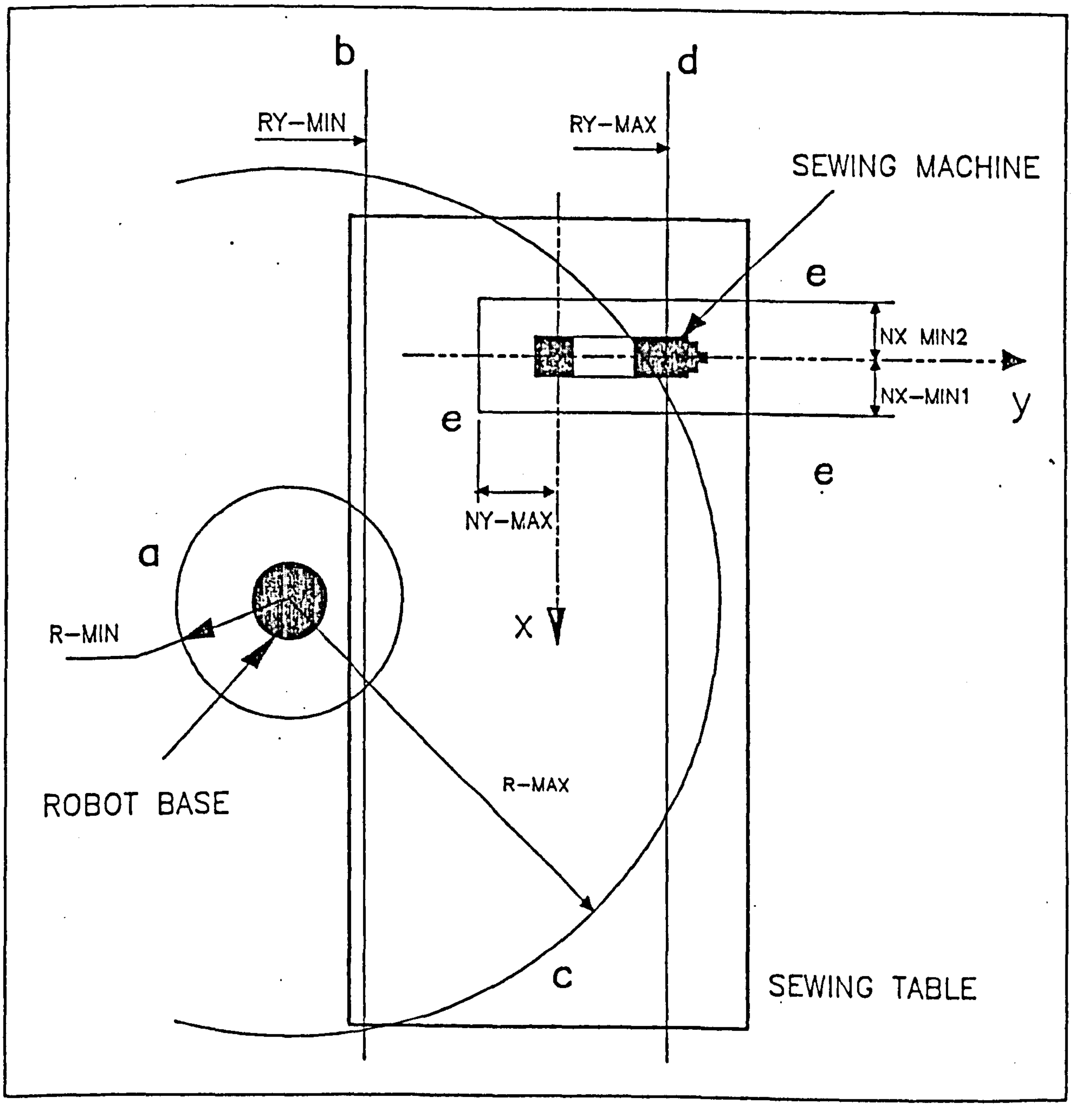


Fig. 5-19: Safe Envelope for Robot Motion

The envelope, shown in fig. 5-19, was bounded by five curves :-

a If the robot approached too close to its own base, then either the end-effector would collide with the base, in the case of a wide end-effector, or the robot would pass through a wrist singularity region

b If the robot moved too far to the left, it would go past the end of the table

c If the robot arm was too far outstretched, then it would reach an elbow singularity region

d If the arm moved too far to the right, then the x coordinate of the TOOL would exceed the 1024 limit (section 2.8.2.3.), and VAL II would abort ALTER

e If either of the two fingers approached the area surrounding the sewing head, there was danger of a collision

Since the first four boundaries constituted a serious restriction to the seam width control, these limitations were implemented carefully, so as to minimize the interference to robot motion. The ALTER data was limited so that the robot decelerated as it approached a boundary. When the robot approached or moved away from the c boundary, then the high inertia loading of the end-effector on the outstretched arm caused serious wobbling. This was corrected by reducing the acceleration and velocity limitations in this region (section 3.5.1.).

Boundaries a and c were applied to the position of the centre of the flange on the end of the robot. Boundaries b and d were applied to the position of the main finger and boundary e was applied to each of the two fingers. The variable names used in the IBM AT software that define these limitation are given in fig. 5-19.

5.4.3. Software Implementation

The SEW Task, in which both the seam width control and the cloth tension control calculations were performed, had the following basic algorithm :-

```
Initialisations
Trigger Z80 to "take a picture"
Start sewing machine sewing slowly

WHILE (seam not complete) DO
BEGIN
     calculate average update rate

          /* control of sewing machine */
     accelerate sewing speed if near beginning of seam
     decelerate sewing speed if near end of seam

          /* cloth tension control calculations */
     read shaft encoder counter
     calculate x increment to track sewing revs
     read cloth tension
     calculate x increment to maintain constant tension

          /* seam width control calculations */
     check if Z80 finished, if not - wait until it is
     transfer pixel data to local RAM
```

```
        trigger Z80 to take a new picture
        calculate x, y and rot(z) increments

            /* ensure safe robot motion */
        apply acceleration and velocity limits to ALTER data
        limit ALTER data if approaching envelope boundary
        install new ALTER message for COMM Task to transmit
    END

    Stop sewing machine
```

The processing overheads required for one update cycle were such that one update was performed every two handshakes, approximately, i.e. an update rate of 0.5 hs^{-1}.

Several embellishments were added to this basic algorithm, such as calculation of sewing speed and standard deviation of seam width and tension errors. Setting up the cloth and the robot for the sewing operation was performed by the higher level MAKE Task (section 6.3.).

5.4.4. Prevention of Buckling

If the cloth panel buckled and lost its rigidity, the robot could no longer rotate the cloth about the needle, i.e. the robot lost control of the panel. Consequently, the prevention of buckling was critical. In addition to the cloth tension control system, described in Chapter 4, several other factors were found helpful in controlling buckling.

5.4.4.1. Cloth Takeup

As the cloth emerged from the sewing head, it sometimes required a smooth and gentle pull, to ensure that the cloth did not "pile up" just past the needle. In many semi-automatic commercial seaming units, this function is performed by a series of driven belts that may be placed on the top or bottom surface of the cloth panel.

This approach is unsatisfactory for this application, since buckling of the cloth is only prevented in the vicinity of the needle. However, the belts would encourage buckling between the robot and the sewing head, since they inhibit rotation of the cloth.

A more satisfactory solution would be a matrix of flotation nozzles, inserted into the table surface, and directed to give the cloth a slight push away from the sewing head. This gentler action would not inhibit rotation of the cloth panel. Although, flotation was not incorporated into the sewing table during this first phase of the project, it is planned to do so when the project is continued.

5.4.4.2. Table Friction

The table friction aggravated buckling and the polished table surface was kept free from dust and grease during the performance tests in order to minimize table friction. Experience with the FIGARO system suggests that the addition of flotation to the table in front of the needle would also be beneficial.

5.4.4.3. Finger Loading

Excessively high spring loading on the fingers aggravated buckling by increasing the effects of table friction. Too low a spring force also encouraged buckling by permitting slipping between the cloth and the finger. The satisfactory range for spring constant was found to be

$$5 < K_s < 100 \text{ g/mm.}$$

5.4.4.4. Damped Motion

Fast lateral motion or oscillatory motion of the cloth, under the robot's control, tended to encourage buckling. This was reflected in the low optimum gain values found experimentally, which effectively restricted the robot motion to gentle and smooth corrections of the cloth incidence angle.

5.4.5. Close Sewing Technique

When a human operator holds the far end of the cloth panel during sewing, he can only cope with gradual curvatures, even with an edge guide. In order to sew a seam in regions of greater curvature, the operator holds the cloth against the table with one hand alongside the needle and one hand in front of the needle. This position facilitates rotation of the cloth panel and prevents it from buckling.

Similarly, the robot could only track gentle cloth edge contours when positioned at the far end of the cloth. A close sewing technique was derived from the far sewing technique described above, so that much greater curvatures

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

could be tackled. The auxiliary finger was positioned alongside the needle, and the end-effector was rotated 90° so that the main finger held the cloth further down the panel (fig. 5-20).

In this position, the cloth could be rotated through much larger correction angles before an envelope boundary was encountered, and the cloth panel had less tendency to buckle. However, the sewing length was limited by the distance between the two fingers, since the fingers had to be repositioned once the main finger had passed beyond the needle. Furthermore, the cloth tension could no longer be measured in this position using the tension sensor, and the cloth tension control was restricted to the open loop control system (section 4.2).

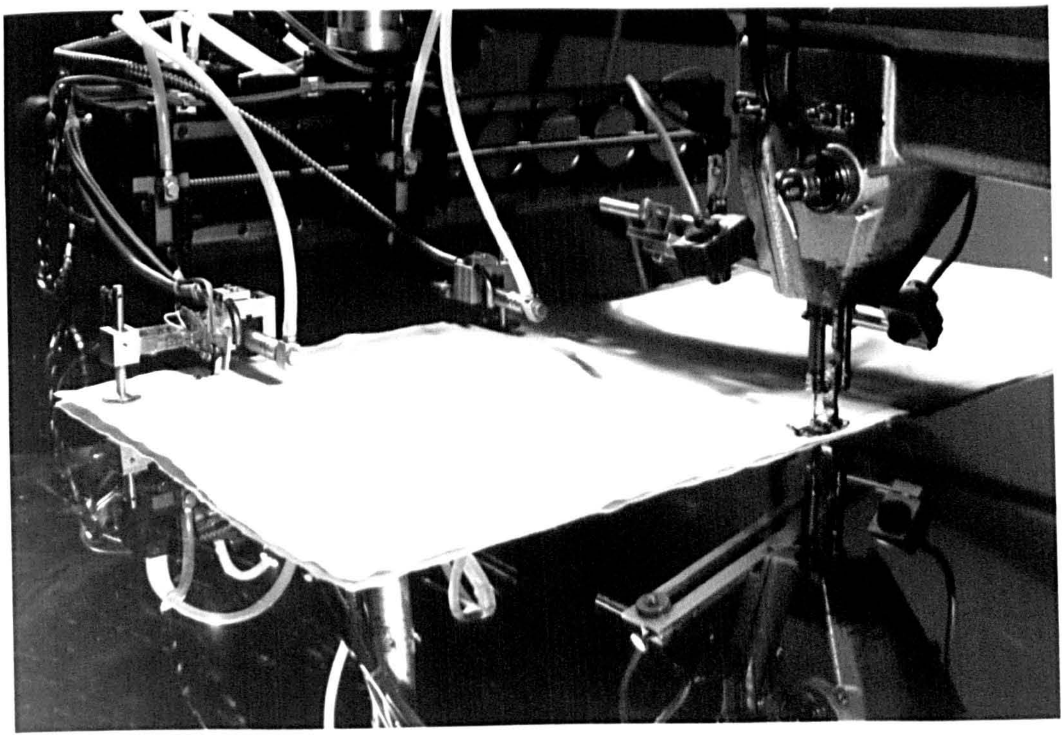


Fig. 5-20: Initial Position of End-Effector for Close Sewing

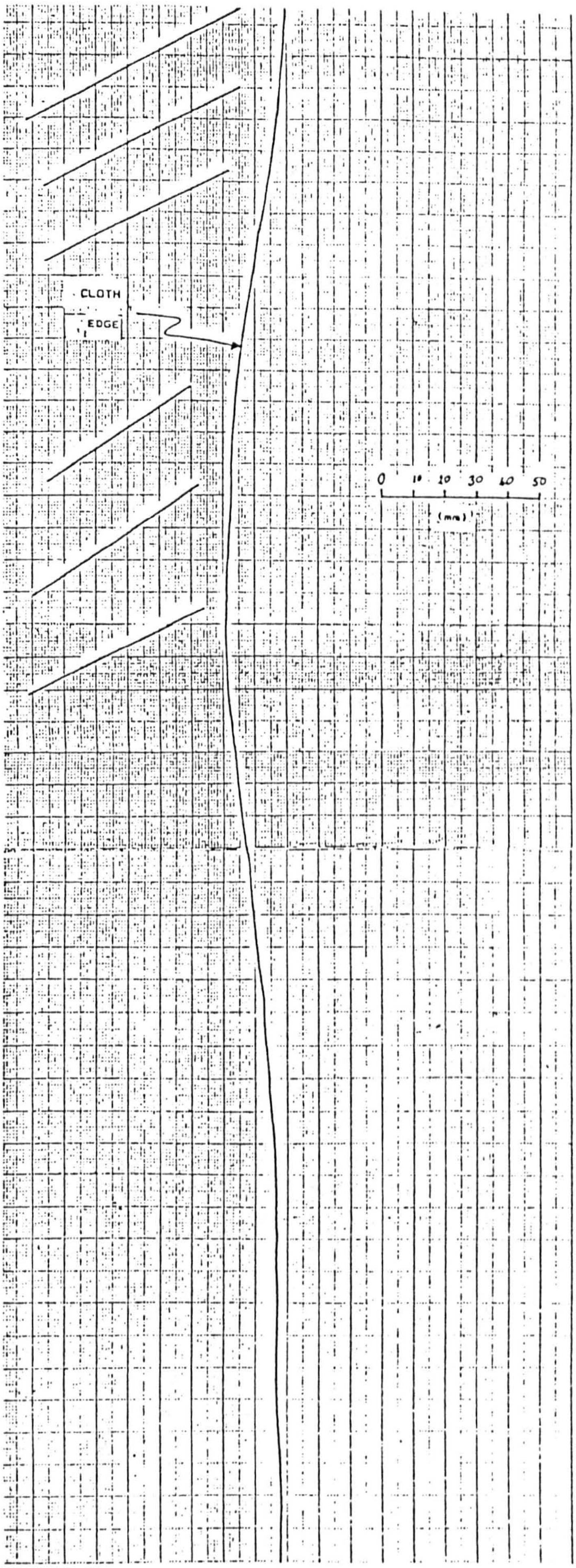


Fig. 5-21: Edge Contour of Test Panel

5.5. Control System Performance

5.5.1. Performance Tests

Extensive performance tests were carried out on a two-ply cloth panel for a range of sewing speeds, in order to produce performance plots of the same style as presented for the simulation results (section 5.2.2.3). Performance plots were also obtained to determine the effect on performance of robot motion limitations, and the number of plies. Exploratory tests were performed to investigate the system's sensitivity to fabric type.

For the vast majority of the performance tests, the test cloth panel was the same fabric as the test fabric used in the tension control performance tests (section 4.4.4). The edge contour of the test panel, shown in fig. 5-20, is representative of contours found on trouser, jacket and skirt panels.

The initial seam width error and incidence angle, ß, were kept to a minimum throughout the tests using the fine adjustment techniques, developed in the setting up operation which is described in the next chapter.

5.5.1.1. Performance Index

As with the cloth tension control (section 4.5.1), either the standard deviation or the average seam width error could be used as a performance criteria. The standard deviation was selected as the performance criterion since fluctuations in the seam width, even a gradual undulating seam, are unacceptable aesthetically, whereas a small constant offset (e.g. producing a 12 mm seam instead of a 13 mm seam) is perfectly acceptable.

The performance curves were plotted according to a performance index of 0.6 mm, i.e. the seam width control performance was considered unacceptable if the standard deviation of the seam width error exceed 0.6 mm.

5.5.1.2. Sample Printout

A typical printout of the robotic sewing test program, with the details of the performance of the seam width and tension control systems, is given in fig. 5-22.

5.5.2. Performance Results

Figures 5-23, 5-24 and 5-25 show the effect of sewing speed, number of plies and robot velocity limitation, respectively, on seam width control performance. The performance curves indicate the regions within which the performance criterion is satisfied (section 5.5.1.1).

The parameter settings that were used for these tests are listed in table 5-4.

03 JAN/87 20:19:14

Robotic Sewing Development Program
Version 2.10

Input Data

Parameters Set At Compile Time

robot stopping dist	= 120 mm	pixel width - cam #1	= 0.430 mm
maximum RHS motion	= 251 mm	pixel width - cam #2	= 0.670 mm
maximum LHS motion	= 160 mm	dist. between 2 fingers	= 156 mm
deceleration length	= 130 mm	inter camera distance	= 30.0 mm
stitch length	= 3 mm	seam width	= 12.0 mm

Parameters Set By User

pixel row no. - cam #1	= 4	tensn servo, propnl gain	= 0.00075
pixel row no. - cam #2	= 7	tensn servo, intgrl gain	= 0.00001
x axis offset - cam #1	= 2	request cloth tension	= 70
x axis offset - cam #2	= 2	seam servo, propnl gain	= 0.050
robot velocity limitatn	= 4	seam servo, deriv gain	= 0.300
robot accelrtn limitatn	= 2		

Parameters Set At Run Time

seam length	= 483 mm	sewing speed	= 1910.6 rpm
tension offset	= 2	sewing speed	= 92.02 mm/s

Output Data

Processor Performance Data

no. ALTER handshakes	= 244	no. feedback loops	= 118
handshakes/update rate	= 2.07	time period for speed	= 64 ticks

Robotic Sewing Performance Data

seam width servo		cloth tension servo	
standard deviation	= 0.374	standard deviation	= 47.160
sum of mean deviation	= 17.9	sum of mean deviation	= 26253
sum of average error	= -13.78	sum of average error	= -523
maximum error	= 0.91	maximum error	= 147
minimum error	= -0.91	minimum error	= -70

Fig. 5-22: Sample Printout of Edge Seaming Program

Parameter	fig 5-23	fig 5-24	fig 5-25
nominal seam width, R_s	12	12	12
camera distance, x_{CAM}	30	30	30
pixel width #1, y1_pixel	0.43	0.43	0.51
pixel width #2, y2_pixel	0.65	0.65	0.49
cloth speed, V_c		40	80
tension prop. gain, K_1	0.000750		0.00150
tension deriv. gain, K_2	0.000015		0.00003
number of plies	2		1
acceleration limitation	3	3	
velocity limitation	8	8	

Table 5-4: Parameter Settings for Performance Tests

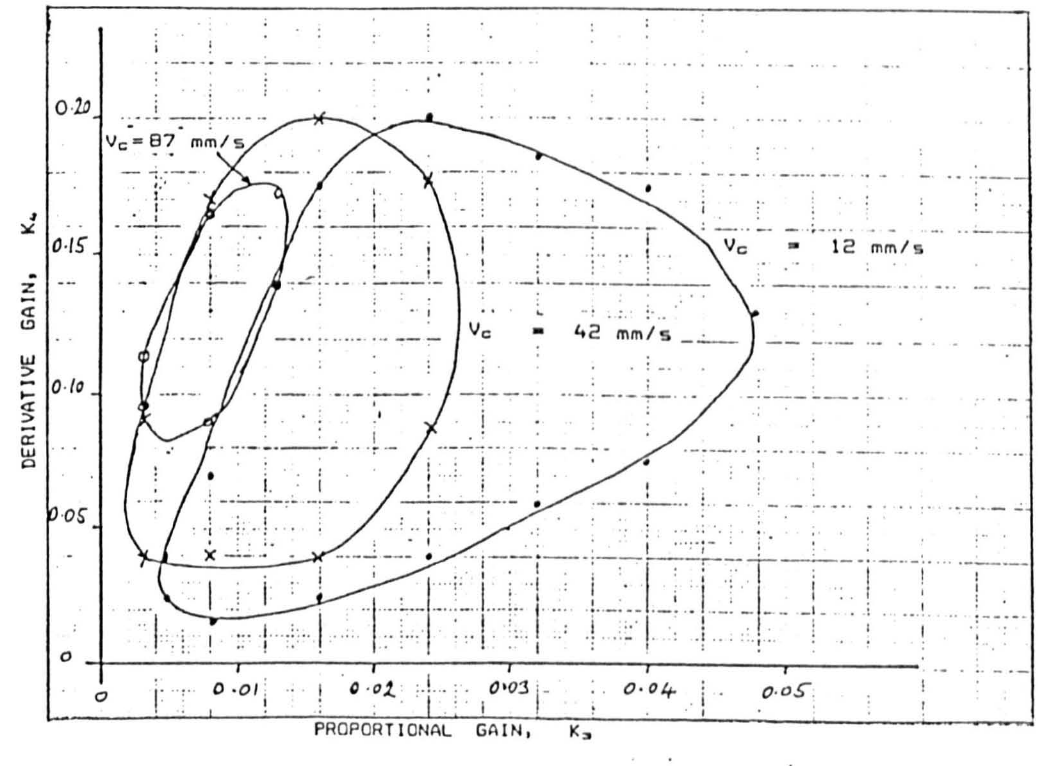


Fig. 5-23: Effect of Cloth Speed on Seam Width Control Performance

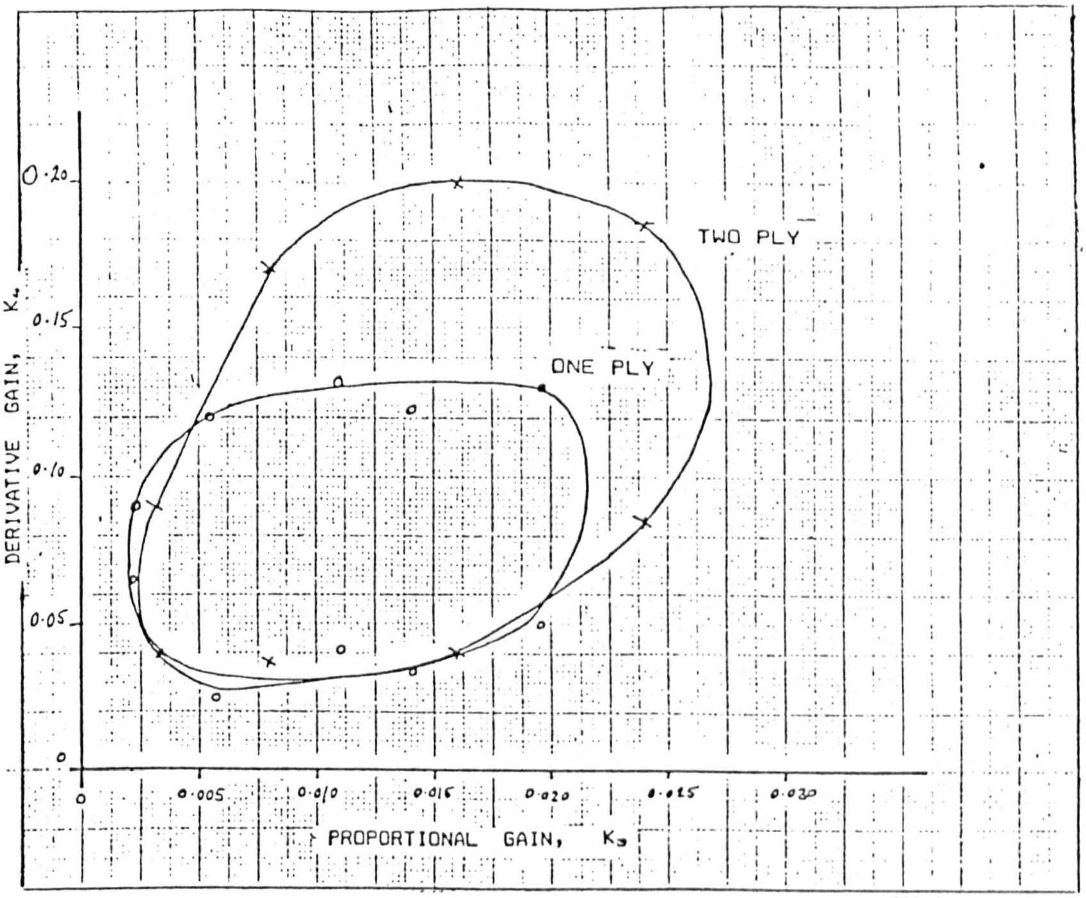


Fig. 5-24: Effect of No. of Plies on Seam Width Control Performance

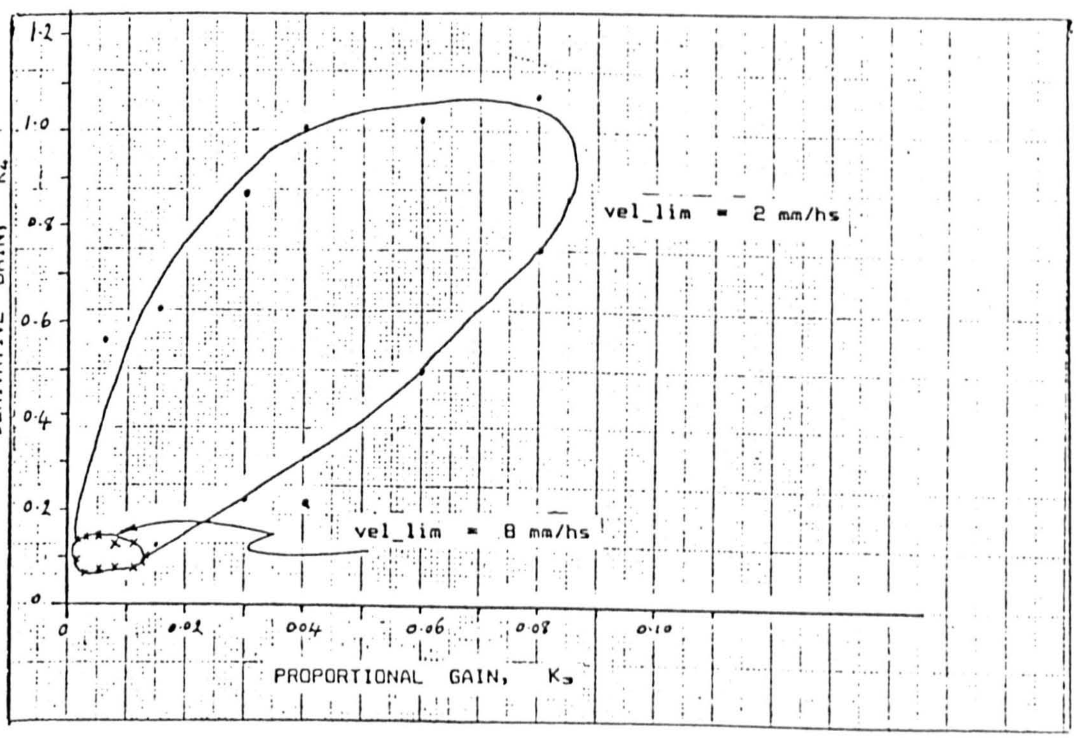


Fig. 5-25: Effect of Velocity Limitation on Seam Width Control Performance

5.5.3. Summary

The results of the performance tests are summarized as follows :-

1) An increase in the cloth speed reduces the stability margin of the seam width control system, reduces the optimum proportional gain value and increases the optimum derivative gain value.

2) A two-ply cloth panel has less tendency to buckle than a single-ply panel, due to its increased stiffness and extra weight. This was reflected in the performance results which showed that the two-ply panel had a larger stability margin.

3) The tendency to buckle was observed to be different for different fabrics; heavy or tightly structured fabrics exhibited greater stiffness than light or open structured fabrics.

4) When the robot's motion was damped down by reducing the velocity limitation, the stability margin was improved enormously.

5.6. Discussion

5.6.1. Comparison of Performance with Simulation Results

The performance curves of the actual system show a similar pattern to the simulated performance curves (compare figs. 5-23 and 5-8). The effect of damping the robot's motion with excessive dynamic limitations was as predicted by the simulation experiments (compare figs. 5-25 and 5-10).

When the actual and simulated performance results are compared quantitatively, the optimum gains have quite different numerical values. The optimum gain values produced from the simulation program were approximately 20 times those found experimentally. There are several factors that contribute to this apparent discrepancy :-

1) Both the actual and simulated systems generate a correction angle, $\delta\alpha$, from the gain values using equation (5.11). The simulated system then rotated the cloth by $\delta\alpha$ after the system delay, δt. However, due to the real time considerations of the ALTER facility, the actual system directed the robot to rotate the cloth at an angular rate of $\delta\alpha$ rads/hs. For most of the simulation runs, δt was set at 140 ms, so that the simulated rotation was performed at approximately 5 $\delta\alpha$ rads/hs. This accounts for a factor of 5 between the simulated and actual gain values.

2) The simulation program was based on a global system time delay, which accounted for the delay between the measurement and actuation processes. However, in practise, the two processes occurred in parallel and with different associated delays. the actuation delay was determined by the ALTER facility, and the sampling delay was determined by the vision system and the update rate of the servo control calculations loop.

   The accuracy of the simulation model could be improved by differentiating between the sampling rate (i.e. the delay between obtaining new feedback measurements) and the actuation delay (i.e. the delay between obtaining a new measurement and making a correction).

3) The simulation results were based on a system time delay of 140 ms, which was estimated by assuming an update frequency of 0.5 update/hs and a well behaved vision system. However, the camera system's erratic behaviour (section 5.3.7), caused the effective system time delay to vary between 140 ms and 224 ms.

4) The simulation model was based on the assumptions that the cloth panel was stiff and did not buckle, and that the vision system produced perfect and accurate images of the edge of the cloth. The effect of the cloth's lack of stiffness and of the poor performance of the vision system on the seam width control was unpredictable; these random factors constitute a noise input to the system (section 5.6.2).

   Derivative control systems are particularly sensitive to noise [57], although the effects of noise can be countered by damping down the system. This is confirmed by the considerable improvement in stability margin obtained by damping down the robot's motion (fig. 5-25).

5) The performance plots for the simulation results and for the actual system were plotted according to different performance indices (sections 5.2.2.1 and 5.5.1.1).

Comparison of the simulated and actual systems suggest that the seam width control could be improved by

- reducing the signal noise level in the system
- reducing time delays in the system.
- reducing actuation errors.

5.6.2. Signal Noise

Occasionally, the cloth panel would buckle, when rotated about the needle, in such a way that the edge of the cloth panel would curl up around the presser foot, and the vision system would have an erroneous image of the cloth edge. Excessive cloth tension and inaccurate robot rotation, in particular, caused this type of buckling, in addition to the influence of the presser foot itself. The closer the fingers were to the presser foot, the greater the inhibiting effect of the presser foot on the rotation of the cloth.

The other cause of noise in the image of the cloth edge was the unstable and erratic image produced by the vision system itself, as discussed in section 5.6.1.

5.6.3. System Time Delays

Both the sampling delay and the actuation delay are detrimental to the control system's performance.

When the SEW Task routines were optimized and tuned for fast execution speed, the update rate was kept down to 0.5 updates/hs. The vision system provided a new picture every 2 to 4 attempts, and, although each attempt could be performed within half a handshake, the present version of the software only triggers the Z80 once per update (section 5.3.2). Consequently the effective sampling delay is between 4 and 8 hs (i.e. 112 ms and 224 ms).

In addition to replacing the vision system with a more competent one, the sampling rate could be improved by triggering the Z80 more often than once per update.

Ideally, the vision system should refresh the camera frame stores continuously without any external triggering from the IBM AT, so that the frame stores contain images that are as recent as possible. A "second best" arrangement would be an interrupt system, so that the vision processor could interrupt the IBM AT when a new image was available.

Even with the present vision system, the sampling rate could be improved. A Timer routine could be included that retriggers the Z80 every 14 ms, to exploit the fast capture time of the vision system.

5.6.4. Actuation Errors

When the cloth buckled between the robot fingers and the sewing needle, the servo-controlled robot trajectory did not produce the anticipated rotation of the cloth. The factors that affect the tendency of the cloth panel to buckle, and preventative measures that were implemented, were discussed in section 5.4.4.

Despite good tension control, some buckling of the cloth was observed when the robot rotated the cloth, under the FAR sewing technique. Buckling of the cloth was more pronounced with the CLOSE sewing technique, and gross buckling occurred when a fabric handling technique was developed to rotate a cloth panel through 90° about a stationary sewing needle (section 6.5).

The major reason for buckling of the cloth under these circumstances was the inherent inaccuracies in the robot and its control system (section 2.4.1). The robot's poor accuracy affected the handling and sewing techniques differently, because of the following factors :-

1) The closed loop tension control system minimized the robot's errors in the x direction.

2) The visual measurement of the seam width error and the incidence angle minimized errors in the y direction.

3) The seam width control only required the end-effector to be rotated within a narrow angular range (± 30° which minimized errors due to rotation of the end-effector.

Some buckling of the cloth was always present during a FAR edge seaming operation. The CLOSE edge seaming technique generated much more buckling of the cloth because it had only open loop tension control and a larger angular range of rotation. Thus, both FAR and CLOSE edge seaming techniques would benefit from a more accurate robot, although the CLOSE technique is particularly sensitive to robot inaccuracies.

5.6.5. FAR and CLOSE Sewing Techniques

The FAR and CLOSE techniques have different advantages and disadvantages. The FAR technique can sew long lengths of cloth without stopping the sewing machine and repositioning the fingers. However, it cannot sew contours that require the robot to rotate the cloth through too big an angle, nor can it sew with the fingers within 150 mm of the sewing needle. The CLOSE technique can accommodate much larger curvatures and can sew right up to the end of the cloth, but it can only be used to sew relatively short seam lengths (up to 300 mm).

A combination of the FAR and CLOSE techniques should be able to produce a quality edge seam on the vast majority of cloth panel contours found in the clothing industry. To confirm this, a panel was cut out in the shape of a jacket sleeve and, using the CLOSE technique, a high quality seam was sewn around the shoulder curve. The shoulder curve had a radius of curvature of 85 mm and an angular extent of 160°

For a particular cloth panel contour, there will be an optimum strategy for sewing along the edge. This strategy would specify the following :-

a) number of segments,
b) the length of each segment
c) CLOSE or FAR technique
d) the position and orientation of the fingers on the panel for each segment
e) the sewing speed for each segment

A technique was developed for automatically repositioning the robot's fingers between segments of a sewing operation to facilitate segmented production of an edge seam, and is described in section 6.3.3. A decision making algorithm was developed which automatically specified a sewing strategy for a particular seam based on its length.

The concept of a segmented seam production can be compared to the manual techniques employed by sewing operators, who often change hand position on a cloth panel during sewing. For example, when producing the long seam on a trouser panel, initially the operator usually grips the cloth close to the beginning of the seam, in order to control the cloth accurately during the initial high curvature section. When the long straight section has been reached, the operator will either grip the end of the cloth and accelerate the sewing machine, or will hold the cloth with alternate hands as the cloth is fed into the machine.

5.6.6. Damped Robot Motions

The performance tests showed that the stability margin of the seam width control was vastly increased when the robot's motion was damped by reducing the velocity limitation. However, excessive damping also reduced the performance of the control and the optimum velocity limitation depends on the cloth velocity and on the contour to be sewn.

At present, the velocity and acceleration limitations are set by the user, at the initialization phase. A fully automatic version of the software would set the limitations internally according to the sewing speed. Ideally, an adaptive control technique should be employed to vary the control parameters, during the sewing operation, according to circumstances.

5.6.7. Adaptive Control

Since the seam width control was sensitive to the sewing speed, exaggerated and unstable behaviour was often observed at the beginning and end of a sewing segment, when the sewing machine was accelerating up to the nominal velocity or when it was decelerating.

Since the sewing speed is not always held constant during a sewing operation, and since the sewing speed can be changed externally by a control knob on the sewing machine, a more robust version of the control system would vary the control parameters automatically with variations in the sewing speed. This adaptive control capability could be implemented by relating the velocity limitation to the cloth velocity either with an empirical formula, or using a look-up table. The look-up table could also relate the optimum values of K_3, K_4 and acceleration limitation to the cloth velocity.

5.6.8. Conclusions

a) A seam width control system has been developed that, in conjunction with the tension control (Chapter 4) and the ALTER channel (Chapter 3), can adaptively perform the edge seaming operation on a cloth panel with an edge profile of arbitrary contour.

b) The system can accurately sew edge seams at speeds up to 150 mm/s (or 4500 rpm for 2 mm stitch length), without pucker, for cloth contours with only a slight curvature. Cloth contours which are moderately curved, such as for trouser and skirt panels, can be sewn accurately at 100 mm/s (or 3000 rpm for 2 mm stitch length).

c) A CLOSE technique has been developed to accommodate cloth panels that have intricately curved contours, and to perform the final segment of seams that extend right up to the end of the cloth.

d) The system is unsuitable, in its present form, for fabrics with poor lateral stiffness, such as knitted fabrics, since the cloth edge tends to buckle or curl up around the presser foot. The system performs best with shirting or worsted woven fabrics which have a reasonable resistance to buckling.

e) Similarly, the system performs better when sewing up two-ply panels, which resist buckling better than single-ply panels.

f) The optimum settings of the control parameters are sensitive to the cloth velocity, and these parameters are set manually in the present version of the software. An adaptive control scheme is recommended for future versions.

CHAPTER 6

THE DEVELOPMENT OF FABRIC HANDLING TECHNIQUES

In addition to the robotic sewing techniques that have been described above, several fabric handling techniques were developed, so that the setting up of the cloth panel for a seaming operation and the rotation of the panel about the needle could be performed automatically.

A ply separation device was incorporated into the FIGARO system, so that the robot could pick up fabric plies from a stack and place them on the table. The automatic manufacture of an irregularly shaped three-sided sub-assembly was demonstrated using the techniques developed in this project.

6.1. Software Organization

The hierarchical organization of the IBM AT software for the robotic sewing operations was described in section 3.3.2. The VAL II software required for these operations was relatively simple :-

```
start ALTER mode
wait until interrupted
end ALTER mode
```

However, the robot motions required for the fabric handling operations did not need complex sensory feedback control, and therefore, instead of using the ALTER channel, the robot motions were generated directly by VAL II programs. Closer co-operation and synchronization was now necessary between the IBM AT and the VAL II controller using the GPC channel described in section 2.6.

6.1.1. IBM AT Implementation

Two levels were added to the software hierarchy described in section 3.3.2.; the complete software model is shown in fig. 6-1. This model was designed to provide a clear, logical and modular structure, which would facilitate modification of the software to include new techniques, or to make a different sub-assembly.

The CONT Task was responsible for the overall operation of the FIGARO sewing station, including the following functions :-

a) initialization and termination of the GPC channel
b) management of interface to supervisor/operator
c) receive data on batch quantities and product type
d) instruct relevant MAKE Task to make required product
e) error recovery

The MAKE Task was responsible for the sequence of operations required to make a specific sub-assembly. A separate version of the MAKE Task is required for each product type.

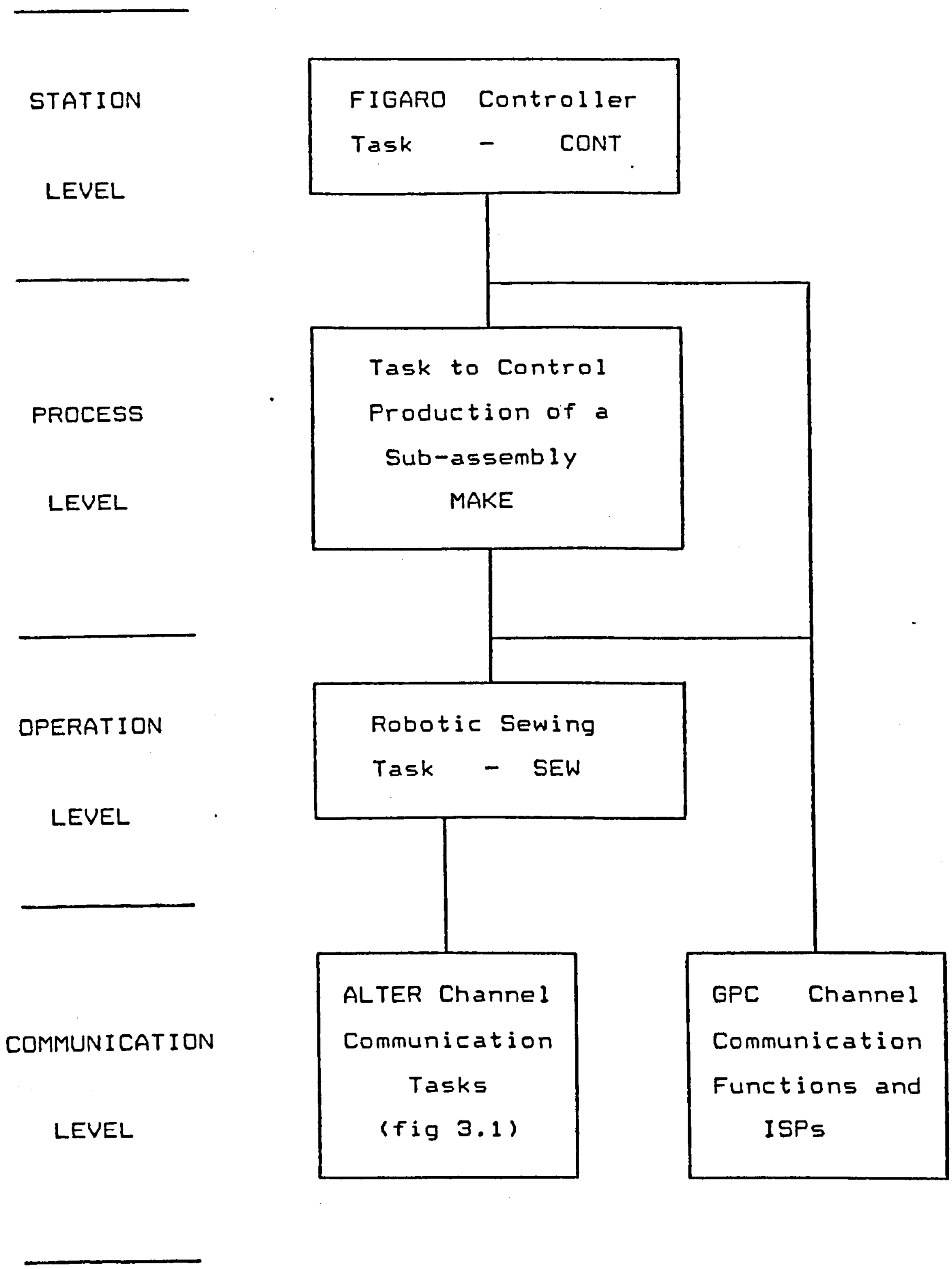


Fig. 6-1: Hierarchical Organization of IBM AT Software

6.1.2. VAL II Implementation

As described in section 2.2., a master-slave relationship was required between the IBM AT and the VAL II controller. This was achieved by splitting the VAL II software into functions that could be individually requested by the IBM AT via the GPC channel. A VAL II program called MAIN acted as the interface between the VAL II functions and the GPC channel. The MAIN program waited until it received a function request, and then it would call the relevant VAL II subroutine. When the subroutine had terminated, the MAIN program returned either the function number or zero to the IBM AT to signal either the successful or unsuccessful completion of the function.

6.2. Second Prototype of FIGARO End-Effector

During the development of the fabric handling techniques, several improvements to the simple early prototype end-effector were considered. An improved end-effector was assembled which incorporated two improvements :-

a) In place of the original manual adjustment, the distance between the two fingers could be changed automatically under program control.

b) the high profile photocells were replaced with a low profile design so that they could be located closer to the fingers.

6.2.1. Programmable Finger Distance

The ideal position for the two fingers which ensured that the cloth panel did not buckle, was to place one finger at each corner of the end of the cloth panel (fig. 6-2). Thus, the optimum distance between the fingers was dependent on the width of the cloth panel. During a typical sequence of operations, the robot would hold the cloth panel along both the narrow and the wide sides, and therefore a facility for changing the finger distance automatically during a sub-assembly was desirable.

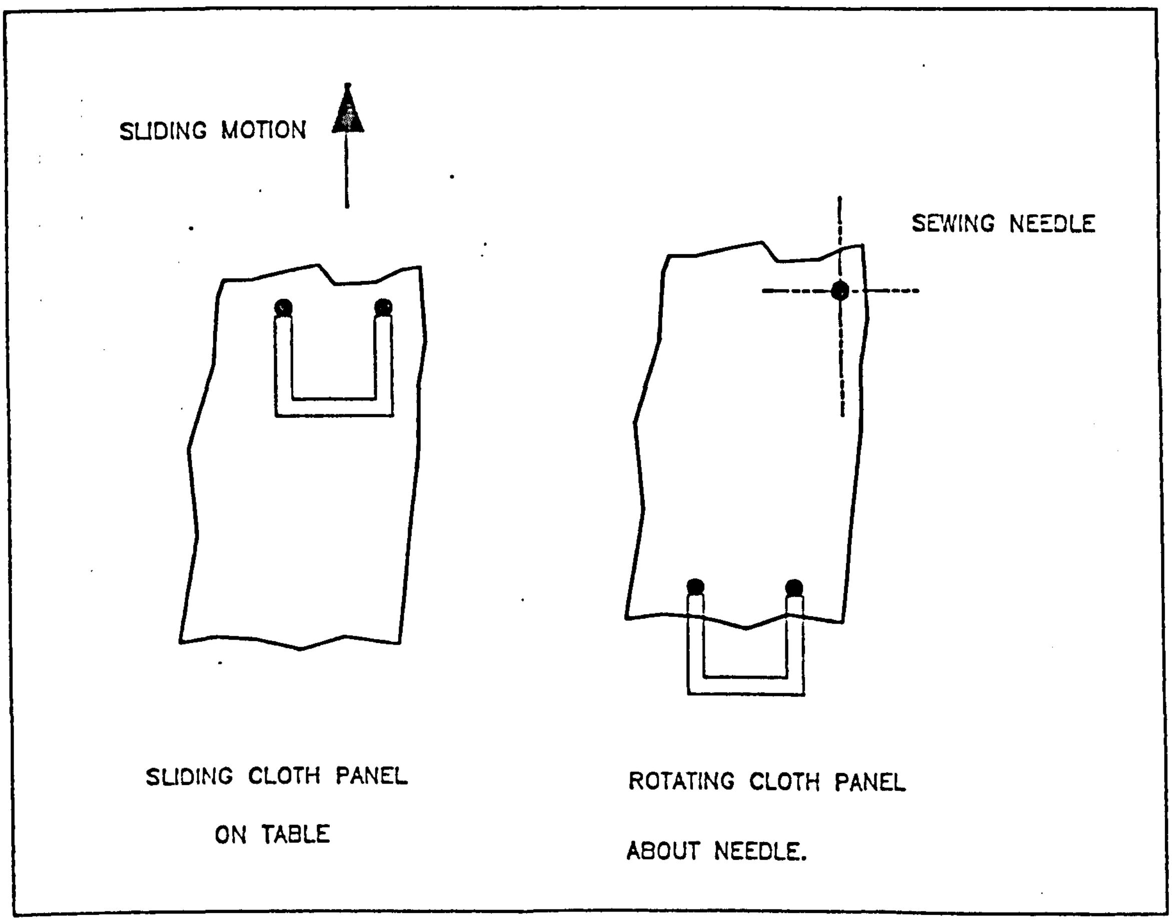


Fig. 6-2: Optimum Location of Fingers

6.2.2. Low Profile Photocells

CLOTH... ...ERS' LIBRARY
UNIVERSITY OF LEEDS

The end-effector was designed with a low profile near the fingers so that they could approach the sewing head without any collisions (section 2.8.2.2). The two original photocells were 95 mm high and therefore they had to be mounted 150 mm behind the fingers.

The photocells were used to locate the edge of the cloth panel in order to place the fingers correctly on the cloth. An additional robot motion was required during the search sequence to accommodate the large offset between the photocells and the fingers. When low profile photocells were installed close to the finger pads, the offset correction motion could be eliminated and the sequence was simplified and faster.

6.2.3. Design of Second Prototype

6.2.3.1. The Leeds Ply Separation Device

Towards the end of the project, an industrial prototype of the University of Leeds ply separation device [72] became available. The ply separation device included two bayonet assemblies and a dc servomotor which could vary the distance between the two bayonets. It was realized that the ply separation device could be easily modified to perform the functions of the FIGARO end-effector, and at the same time provide a programmable finger distance facility.

In addition, the ply separation device could extend the usefulness of the FIGARO system by adding the following handling capabilities :-

a) picking up a ply from a stack

b) placing one ply on top of the other

c) folding a cloth panel

6.2.3.2. Modifications to Ply Separation Device

The second prototype FIGARO end-effector, shown in fig. 5-20, was based on the ply separation device. The instrumented finger was mounted on the fixed bayonet housing and the auxiliary finger was mounted on the movable bayonet. A miniature fibre optic sensor head was mounted on each finger assembly to perform the same function as the original photocells. A fibre optic cable connected each sensor head to a conventional infra-red variable photocell which was mounted on the robot's forearm.

6.3. Setting Up for the Edge Seaming Operation

The first handling operation that was automated on the FIGARO system was the setting up of the cloth panel for the edge seaming operation.

6.3.1. Sequence for Setting Up Operation

The sequence for the setting up operation is listed in table 6-1 and consists of three sections :

- Place the cloth corner under the needle
- Measure the cloth length and decide on a strategy
- Place fingers on the cloth and make final adjustments

Sequence of Functions	Function	VAL II routine	IBM AT routine
1. Place cloth corner under needle.			
lift sewing m/c presser foot			
find cloth on table	3	findcloth	
find top right hand corner	4	corner	
slide cloth corner under needle	5	uptoneedle	
fine adjust for seam width	19	fine.adj	fine_adj
put sewing needle down			ndle_down
remove robot from needle zone	23	remove	
2. Measure cloth length and decide on strategy			
find cloth end & cloth length	25	end.cloth	
report robot's position	11	calc.where	where
decide seam sewing strategy			DecideSeam
3. Place fingers on cloth and make final adjustments			
IF using FAR technique THEN			
find bottom right hand corner	6	go.far.st	
IF using CLOSE technique THEN			
position fingers for close sew	17	go.close.st	
fine adjust for cloth angle, ß	20	angle.adj	angle_adj

Table 6-1: Sequence for Setting Up Operation

6.3.2. Placing Cloth Corner Under Needle

The routines for placing the top right hand corner of the cloth panel under the sewing needle, which are described below, can accommodate almost any size and shape of cloth panel placed anywhere on the sewing table, within the following limitations (fig. 6-4):-

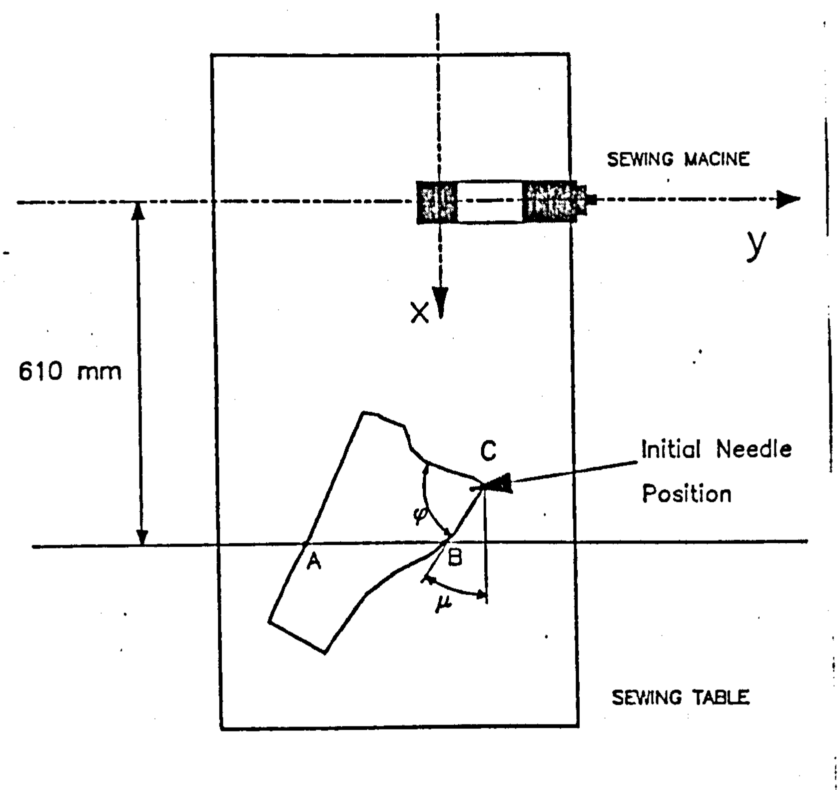


Fig. 6-3: Starting Conditions for the Setting Up Operation

1) The panel should be placed down on the table so that the edge to be seamed is on the right hand side, and the inclination of that edge to the x axis, ϕ, is within 30°.

2) The panel should be placed so that the x = 610 mm line is covered (i.e. approximately opposite the robot's base).

3) The seam starts at the top right hand corner. The angle, θ, between the top edge and the right hand edge is between 80° and 110°, i.e. the top right hand corner should be approximately square.

The initial position of the cloth panel and the terms and symbols used in the description of the routines are defined in fig. 6-4.

6.3.2.1. Finding Cloth Panel

1) The robot scans the table along the line x = 610 mm until pcell1, the photocell mounted close to the main finger, detects a transition from "cloth absent" to "cloth present", at location A.

2) The y coordinate of the first edge is noted and then the robot continues to scan as before until pcell1 detects the opposite transition, at location B.

3) The apparent cloth width along that line is calculated and the robot is moved back so that the two photocells are centred over the apparent centre of the cloth.

6.3.2.2. Finding Top Right Hand Corner

1) The robot scans along the cloth in the x direction until one of the photocells detects the top end of the cloth.

2) The robot's TOOL transformation is reset so that its z axis is coincident with the photocell that detected the edge.

3) The robot rotates the end-effector about the first photocell until the other photocell also detects the end of the cloth. The end-effector is now aligned to the cloth's top edge.

4) The robot moves 30 mm back, perpendicular to the cloth edge, and then traverses parallel to the cloth edge until pcell1 detects the right hand edge.

5) The end effector is now aligned to the top edge and its position relative to the top right hand corner is known. The robot is lowered until the fingers hold the cloth, with the main finger close to the top right hand corner and the auxiliary finger close to the top left hand corner (fig. 6.2a).

6.3.2.3. Moving Cloth up to Needle

Once the robot had put its fingers down relative to the top right hand corner, the robot was directed to slide the cloth panel to a taught location, under_ndle. The robot location transformation, under_ndle, was defined such that the fingers held the cloth panel with the initial sewing point approximately under the needle, and with the top edge aligned to the sewing machine's y axis.

Thus, this handling operation moved the cloth panel from an unknown location and orientation to a known location and orientation defined in terms of the sewing machine. Since the sliding motion was predominantly forwards and the sideways and rotational components of the motion were gradual, buckling forces on the cloth panel were insignificant. The tendency to buckle would be further reduced if flotation was incorporated into the sewing table (section 6.7.2).

6.3.2.4. Fine Adjustment of Seam Width

The sequence so far has positioned the cloth with the initial sewing point approximately under the needle, with a repeatability of up to ± 3 mm. The following factors contributed to this inaccuracy :-

1) The PUMA 560 is inaccurate (section 2.4.1), particularly when programmed "off-line" and for changes in orientation.

2) When a photocell detects the cloth edge, the robot overshoots, and this braking distance depends on the initial robot velocity.

3) The cloth panel has curved edges of arbitrary contour in addition to an arbitrary starting position and orientation. However, a technique based on only two photocells to align the end-effector to the cloth edge, assumes that the cloth edge is a straight line.

The simulation program confirmed that a large initial seam width error could make the seam width control go unstable. Consequently, a fine adjustment function was required to minimize this initial error.

This function involved close interaction between the I-SIGHT vision system, the IBM AT and the VAL II system. The IBM AT used the two I-SIGHT cameras to provide a measurement of the seam width error, which it communicated to VAL II. The robot was directed to move the cloth to reduce the error, and on completion of the move, VAL II returned an acknowledgement to the IBM AT. This cycle was repeated until the seam width error was under 0.5 mm.

Following the fine adjustment function, the IBM AT drives the sewing machine until the needle reaches the "down" position, piercing the cloth. Once the cloth is held in position by the needle, the robot's fingers are carefully removed from the needle zone, without pulling on the cloth or colliding with the front camera assembly.

6.3.3. Deciding on Sewing Strategy

The FAR and CLOSE edge seaming techniques, as described in Chapter 5, have different advantages and disadvantages. The FAR technique is suitable for sewing long seams of gentle curvature up to 150 mm of the sewing needle. The CLOSE technique can only be used to sew short segments of a seam (up to 300 mm), but can accommodate much larger curvatures and can sew right up to the end of the cloth.

Many edge seaming operations will require a combination of FAR and CLOSE techniques, and the DecideSeam function sets up a data structure which contains the number of seam segments, the sewing technique for each segment, the sewing speed and the segment length. A sophisticated version of the DecideSeam function, which would generate a sewing strategy based on the cloth profile, is discussed in section 7.2.3.3.

The present version of DecideSeam implemented in the MAKE Task, was based on the length of the cloth panel in the direction of sewing. If the cloth was less than 300 mm then the whole seam was sewn using the CLOSE technique. If the cloth was longer, then the seam was sewn in two sections, a FAR section up to 200 mm before the needle, and a CLOSE section to complete the cloth. The FAR section was sewn at top speed and the CLOSE section was sewn at half top speed, and the actual top speed was set manually from the control knob on the sewing machine.

The cloth lenqth was easily determined by searching for the far edge of the panel (the cloth.end function) and calculating the distance between the main finger and the needle (the calc.where function).

6.3.4. Placing Fingers on Cloth Panel

The starting position for the robot's fingers for the FAR and CLOSE techniques are shown in figs. 5-1 and 5-20 respectively. The go.far.st function which places the fingers at the FAR starting position, searches for the far right hand corner in a similar fashion to the corner function, except that the end-effector is not aligned to the cloth edge.

The go.close.st function places the fingers at the CLOSE starting position as follows :

1) The end-effector is rotated by 90° in a zone free from obstructions.

2) The robot searches for the lower left hand corner of the panel.

3) If the left hand edge is within 180 mm of the needle, then the robot places the fingers down with the main finger in the bottom left hand corner and with the

second finger as close to the top left hand corner as possible.

4) If the left hand edge is further from the needle, then the fingers are placed along the y = 180 mm line, with the main finger close to the lower edge and the auxiliary finger close to the top edge.

6.3.5. Fine Angular Adjustment

Once the fingers have been placed down on the cloth, one final adjustment is required before the sewing operation can start. Although the cloth was accurately positioned so that the needle was put down at the correct starting position, the orientation of the cloth (i.e. the incidence angle of the cloth edge, ß) was still only approximate.

The ang.adj function was identical to the fine.adj function, except that the IBM AT conveyed measurements of the angle ß to VAL II, and the robot rotated the cloth panel about the needle to reduce the angular error. The rotation of the cloth about the needle was performed by the rotate.ndle routine, which is described below in section 6.5.

6.4. Completing the Edge Seaming Operation

In order to incorporate the edge seaming function into a fully automatic sequence of operations, additional developments were required.

6.4.1. Segmented Seam Production

As explained in section 6.3.3, most seams require a combination of CLOSE and FAR techniques and the DecideSeam function provides a sequence of CLOSE and FAR segments for a particular seam. The MAKE Task contained the following loop structure immediately after the DecideSeam function in order to obtain the desired segmented production of the seam :-

```
for each segment of the seam
begin
     if CLOSE segment
          go.close.st
     else
          go.far.st
     angle.adj
     start up ALTER communications channel
     start SEW Task and wait until it is completed
     end ALTER communications channel
end
```

6.4.2. Sewing Up to the End of the Cloth

Most seams are terminated a short distance before the end of the cloth (seldom more than 10 mm). Consequently, a technique was required to terminate the sewing operation at an accurate distance from the cloth end.

Although the distance between the cloth edge and the needle was known accurately at the start of the sewing operation, this distance could not be accurately calculated during the sewing operation for the following reasons :-

1) The sewing machine revolutions did not give an accurate measure of the cloth feed due to the imprecise feed mechanism.

2) The robot had poor absolute accuracy.

3) The motion of the end-effector in the x direction did not accurately reflect the cloth edge to needle distance since slight slipping between the finger pads and the cloth occasionally occurred.

Consequently the cloth end had to be detected using a sensor. If the robot had been more accurate than the position of the cloth end could have been calculated from the ALTER data in the x direction with reasonable accuracy since the slipping between the fingers and the cloth was not a significant source of error. However, the use of a sensor to detect the cloth end, provides additional feedback information which improves the robustness of the system.

6.4.2.1. Detection of the Cloth End

Initial attempts to use the I-SIGHT cameras to detect the end of the cloth failed because of their narrow field of view. The cloth edge occasionally disappeared totally from the image of the forward camera during sewing due to large radius of curvature or excessive rotation of the panel. Consequently, the forward camera would occasionally give a false indication of the cloth end. Similarly the primary camera could not provide a cloth end detection capability since it would detect the cloth end some time after the seam width control system had already reacted to an apparent severe step change in the cloth contour.

The cloth end was detected by an additional photocell mounted on the sewing machine so that it gave 28 mm early warning (i.e. the x component of the photocell to needle distance). The photocell was mounted 15 mm to the left of the sewing needle to ensure that the photocell was not prematurely affected by the rotation of the cloth panel during the sewing operation.

6.4.2.2. The inch Function

The cloth end had to be detected before the needle reached the end of the seam, so that the seam width control system did not generate erratic robot motion when the cloth end passed by the field of view of the forward camera. The SEW Task was therefore terminated as soon as the cloth end was detected, and the seam finished 28 mm from the cloth end.

The inch function completed the remainder of the seam length by "inching" along at slow speed. First the robot moved the fingers forward by the required distance, which caused the cloth to loop upwards and removed any cloth tension. The sewing machine was then operated for a specific number of stitches which accurately finished off the seam. Since there was no cloth tension and the sewing speed was very slow, the feed mechanism was effective and repeatable.

The number of stitches was obtained from a calibration test, and was only dependent on the position of the stitch length knob on the sewing machine. For a stitch length nominally set at 3 mm, seven sewing revolutions would extend the seam up to 10 mm from the cloth end. Unfortunately, the sewing machine did not have a stitch condensation facility which would have permitted the control of the stitch width from the IBM AT.

A stitch condensation facility is recommended for future prototypes, so that the accuracy of the inch function can be improved, and so that the sewing station is more independent of manual adjustments.

6.5. Rotating Cloth Panel about Needle

A common handling fabric operation which follows an edge seaming operation is to rotate the panel about the sewing needle, which was left in the "down" position at the termination of the previous seam, until the adjacent edge is aligned up ready for seaming.

6.5.1. VAL II Implementation

Rotation of the cloth about the needle during the edge seaming operation was performed using the ALTER facility since the rotation of the cloth was required within a real time sensory feedback control system (section 5.4.1). When the robot was required to rotate the cloth panel as a pure handling operation, sensory feedback and the ALTER facility were not required, so that the entire function could be controlled from a VAL II routine.

The rotation was performed by the VAL II program, rotate.ndle, which was based on the procedural motion control mode (section 2.4.1). As described in section 5.4.1, this rotation operation is composed of two simultaneous motions; rotation of the main finger about the needle and rotation of the end-effector about the main finger. The routine was written in a general format, so that any angle of rotation could be specified.

6.5.2. Effect of Robot Inaccuracy

When the rotate.ndle routine was executed to rotate the cloth by 90°, large errors were observed in the final position and orientation of the end-effector, which caused the cloth to buckle. The errors were particularly high because under the procedural motion control mode the overall motion is the result of the interpolation of many intermediate motions, and the intermediate errors are cumulative.

When the IBM AT software, developed for the seam width control system, was used in conjunction with the ALTER channel to perform the same function, identical errors were observed. This showed that the rotate.ndle function was equivalent to the ALTER version and that the robot's poor absolute accuracy was to blame. The effect of the robot's inaccuracy on cloth buckling is further discussed in section 5.6.4.

6.5.3. Accommodating Robot Inaccuracy

The effect of the robot's inaccuracy was accommodated by adopting the following procedure :-

1) The cloth was buckled intentionally by moving the cloth in towards the needle.

2) The cloth was rotated by 90° using rotate.ndle. Since

the cloth was excessively slack, the inaccurate rotation did not generate any pulling of the fabric between the needle and the fingers.

3) The buckled cloth was straightened out by the straighten routine.

4) The final orientation of the cloth was adjusted by the angle.adj function (section 6.3.5), so the accuracy of the rotate.ndle routine was not critical.

6.5.4. The straighten Routine

A straighten routine was developed based on a directional air jet which was incorporated in the support of the main finger of the first prototype end-effector. The jet was directed at approximately 45° to the vertical and along the finger support beam.

The air jet was placed at a location called blow.position, in which the jet was positioned over the cloth panel, close to the sewing machine, and directed along the line $x = y$ and away from the needle. This technique was only partially successful at straightening out buckled single-ply panels, and it was even less effective with two ply panels and with heavy fabrics.

The reliability of the air jet technique could be considerably improved by the simultaneous use of flotation under the cloth panel to reduce the table to cloth friction.

A different technique for straightening the buckled cloth was developed with the second prototype end-effector, which utilized the pneumatic actuators in the end-effector to lift the fingers off the cloth individually. The "straighten" function raised the main finger off the table, and the auxiliary finger stroked the end of the cloth away from the needle, at 45° to the sewing machine axes. This straightforward method proved successful and reliable.

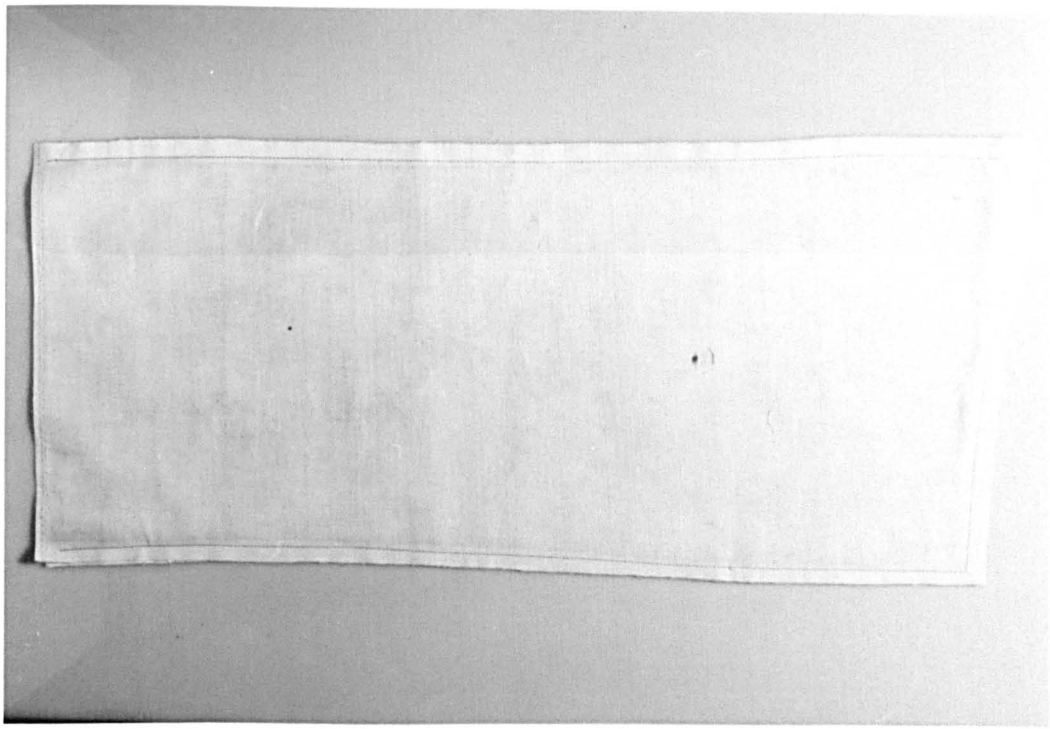


Fig. 6-4: Demonstration of Automatic Production of a Sub-assembly

6.6. Demonstration Assembly

The robotic edge seaming technique and the fabric handling techniques developed during the FIGARO project were demonstrated in the production of an irregularly shaped sub-assembly, in which 3 adjacent edge seams of arbitrary contour are produced to form a bag. The software implementation is shown in Appendix E, and a photograph of the typical results of the sub-assembly production is shown in fig. 6-4.

6.7. Discussion

6.7.1. Overhead Camera

The handlinq techniques developed above used only 2 photocells and the I-SIGHT cameras for locating or confirming the location of cloth panel features. Although the techniques performed the required operations satisfactorily (when the panel was placed on the table within the limits given in section 6.3.2), an industrial implementation would require a more robust and reliable system that would have more visual feedback.

An overhead camera system might provide a more reliable and quicker measurement of the location and orientation of the cloth panel, than using the searching strategies developed above. The limits on the initial position and orientation of the cloth panel, listed in section 6.3.2, could be relaxed considerably. In addition, an overhead camera could provide a measure of the contour profile which could be used in a more sophisticated version of DecideSeam to provide an optimum sewing strategy automatically (section 7.2.4). However, a static overhead camera would require a very high resolution in order to have a field of view that covered most of the sewing table and to locate the cloth edge within a few millimetres.

A more practical solution, that has been applied to other robotic assembly systems [62], is to use a combination of a static overhead camera and an end effector mounted vision system. In this application, an overhead camera could provide the gross position and orientation of the panel, and the two photocells could provide a fine measurement capability using the techniques developed above.
Overlapping redundant sensory feedback systems are often a feature of commercial robotic cells, since they improve the general robustness of the system.

6.7.2. Buckling Prevention

A more accurate robot and the installation of flotation nozzles in the sewing table would make a major reduction in the tendency of the cloth to buckle during handling operations. However, techniques for ensuring that the cloth panel is straight and flat (section 6.5.4), would still be necessary to provide high reliability.

Another measure that reduced the buckling tendency was the programmable finger distance feature of the second prototype end-effector (section 6.2.1). Additional fingers would be advantagous when rotating large panels about the needle but the multiple finger arrangement should be configurable under program control. Furthermore, the design of the multi-fingered end-effector should not reduce the flexibility of the system to perform other handling and sewing operations.

Torgerson and Paul [65] developed an algorithm for the automatic generation of an optimum configuration of a multi-fingered end-effector according to the shape of the cloth panel. Although, the end-effector developed in the $(TC)^2$ project has a measure of programmable reconfigurability [64], it is large and bulky and was not intended for handling operations in the vicinity of the sewing needle.

CHAPTER 7

DISCUSSION

This chapter reviews the achievements of this project to date and discusses the potential of the FIGARO approach and techniques for future developments.

7.1. Review

7.1.1. Objective

The experimental robotic sewing station and the automatic sewing and handling techniques described above, were developed in accordance with an adaptive robotic approach to flexible clothing automation (section 1.5.4.), in which the robot controls the fabric panel using sensory feedback. The objective of this investigation was to ascertain whether this flexible automation approach could, after further research and development, become the basis of a commercially viable, industrial, flexible automatic sewing cell.

7.1.2. The FIGARO Robotic Sewing System

A block diagram of the FIGARO system is shown in fig. 7-1.

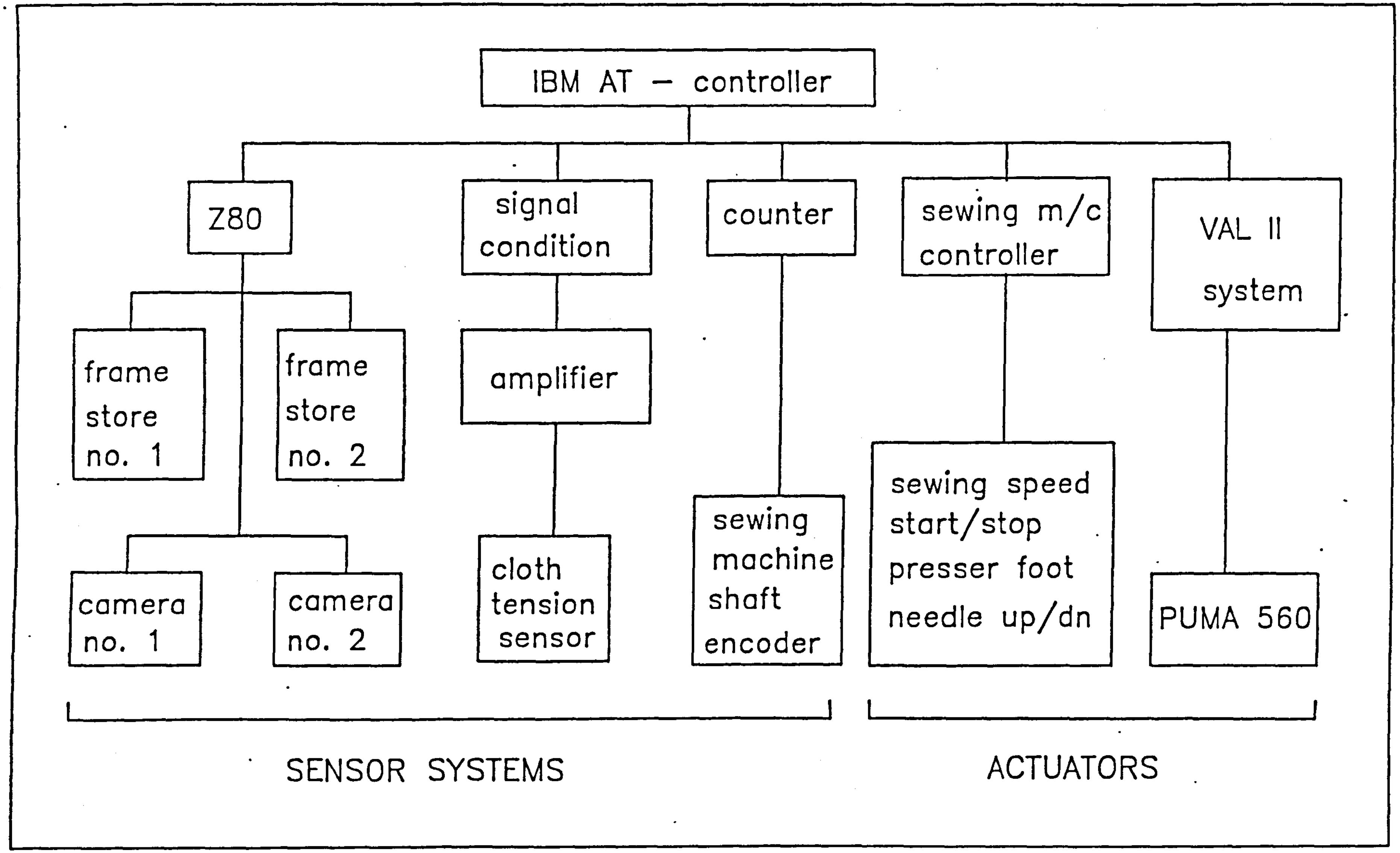


Fig. 7-1: Block Diagram of FIGARO Robotic Sewing System

A control hierarchy was established so that the robot and sewing machine could be controlled in real time in conjunction with multi-sensory feedback.

An IBM AT was used as the cell controller, but its processing power was found to be insufficient for this application. More suitable configurations are recommended in section 7.4.1. The software for the cell controller was developed for execution within a real time multi-tasking environment, allowing different processes to run concurrently.

Two communication channels were set up between the station controller and robot controller; the ALTER channel was dedicated to conveying robot motion data in real time, and the GPC channel was used to provide additional communication facilities.

The system was based on the PUMA 560 robot, because of its advanced programming and control system, VAL II. However, the robot was found to be unsuitable for this application because of poor absolute accuracy. A more suitable robot is recommended in section 7.4.2.

The sewing machine was interfaced to the cell controller so that the various functions (e.g. stop/start, sewing speed, presser foot up/down, needle up/down etc.) could be controlled from the IBM AT.

A sewing table was constructed around the sewing machine and was covered with a smooth, mirror surface stainless steel sheet. The position and height of the robot base relative to the sewing table were deliberately chosen, to minimize the effects of the robot's limited workspace and to avoid singularity regions.

Two prototypes of a special purpose end-effector were developed for handling and manipulating limp fabric on a table. The end-effector was required to control the fabric sensitively, in close proximity to the sewing head, without interfering in the sewing operation or limiting the system's flexibility.

Two spring-loaded fingers were incorporated into a low profile design and their separation distance could be changed under program control. The fingers were tipped with a high friction rubber pad so that the cloth was gripped by the finger without increasing the table surface friction. Photocells and microswitches were installed on the end-effector, in order to locate the cloth panel and as a safety precation.

7.1.3. Adaptive Control of the Robot

The high speed ALTER communications protocol was implemented on the IBM AT in a modular fashion, along the lines of the OSI Reference Model. The software was interrupt driven and optimized to minimize the communication overheads.

The dynamic characteristics of the PUMA 560/VAL II system under ALTER control was investigated experimentally. In order to obtain smooth and linear motions, velocity and acceleration limitations and, in the case of non-cumulative mode, an interpolation algorithm had to be applied to raw ALTER data. The maximum velocity and acceleration had to be reduced when the arm was outstretched, to limit dynamic errors.

7.1.4. Cloth Tension Control System

A robotic sewing technique was developed, in which the robot held the free end of a cloth panel and the robot moved with the cloth during the sewing operation. The robot motion was synchronized with the sewing machine feed mechanism by tracking the sewing machine shaft encoder signal. The buffered shaft encoder signal was interfaced to the IBM AT via a counter circuit. The IBM AT computed the required robot motion and transmitted it to VAL II via the ALTER channel.

This cloth feed tracking system was capable of producing good quality short straight seams, once the stitch length had been manually adjusted for a specific speed. However, under most circumstances, this system was unsatisfactory because the cloth slipped in the sewing machine feed mechanism in an unpredictable way, and the robot would either lag or lead the cloth feed. This either resulted in excessive cloth tension or in a slack and buckled panel. The problem was solved by developing a closed loop control system in which the cloth tension was measured and the robot motion was modified to maintain a constant cloth tension.

A cloth tension sensor was designed to provide high sensitivity in the direction of sewing, and high insensitivity in all other directions. The sensor's signal was amplified and interfaced to the IBM AT via an ADC. The cloth tension signal was found to undulate synchronously with the sewing revolutions durinq sewing, due to the intermittent nature of the feed mechanism. A digital peak detector was incorporated into the ADC circuit so that the cell controller could sample the peak tensions.

Initial experiments with a closed loop cloth tension control highlighted instability problems due to the non-linear behaviour of fabric under tension and due to the table friction. The system time delay and the initial start-up acceleration of the sewing machine caused high initial cloth tensions, which upset the tension control. A gradual controlled start-up acceleration corrected this problem. When the robot attempted to tension the cloth by moving away from the needle, the table friction created an apparent cloth tension even though the cloth was still slack. To avoid this, the robot motion was limited to the sewing direction only.

A proportional and integral control was required to limit tension variations within an acceptable range and to prevent tension build-up, in order to produce good seams. Since satisfactory control could not be obtained through trial-and-error experimentation, the range in which the optimum gain values were likely to be was obtained using a Bode design procedure. The Bode procedure required the open loop frequency response, which was measured by imposing a sinusoidal forcing function on the open loop system.

The performance of the cloth tension control was found to depend on the fabric's mechanical properties, the sewing speed, system time delays and the number of plies. Woven fabrics sewn along the bias and knitted fabrics, although operating under good tension control, produced unacceptable buckling during the sewing operation. The buckling was due to their high extensibility at the average tensions suitable for the control system developed. Good performance was obtained with a variety of woven fabrics at the maximum sewing speed of 5000 rpm. Fabrics, which were pucker sensitive, produced good seams at reduced speeds.

7.1.5. Seam Width Control System

The robotic sewing technique was extended to sew seams parallel to an edge of arbitrary contour by including a vision-based seam width control system in the adaptive control of the robot. A simulation technique was developed which accounted for system non-linearities due to the vision system, system time delays and robot motion limitations. The simulation program showed that a design based on a single camera or on two photocells would not produce stable control. The simulation showed that the system was sensitive to the system time delay, pixel resolution and the initial seam width error. The simulation provided a specification for the vision system and an insight into the control problem.

Two miniature cameras were mounted on the sewing machine and interfaced to the IBM AT. A lighting arrangement was developed which provided a clear black-and-white image of the cloth edge, regardless of the fabric colour. A comprehensive calibration technique was developed to facilitate the setting up of the system, and to ensure accurate and stable edge seam production.

The seam width control required that the robot corrected the orientation of the cloth panel during sewing. This was achieved by superimposing two motion elements; rotation of the main finger about the sewing needle, and rotation of the auxiliary finger about the main finger. The robot's workspace constraints limited the maximum cloth edge curvature that could be tracked. To minimize the effects of these constraints, the robot's permissable envelope was carefully defined and when the robot approached one of the bounds of the envelope, the robot was decelerated smoothly.

Buckling of the cloth panel was a serious problem in the development of the edge seaming technique. When the cloth buckled, it lost its rigidity and the robot effectively lost control. The tendency of the cloth to buckle was minimized by reducing the table friction, reducing the spring loading on the fingers and damping down the motion of the robot. The poor absolute accuracy of the robot contributed significantly to the buckling problem. When the end of the seam approached, the buckling tendency increased due to the effect of the presser foot.

A CLOSE sewing technique was developed to sew the last 100 mm of a seam or to sew intricately curved seams. In this technique, the fingers were positioned on the cloth alongside the sewing head to manipulate the cloth more effectively, although the tension control system had to be restricted to open loop control.

Accurate edge seams were produced at speeds up to 100 mm/s for typical contours. Fabrics with relatively high buckling stiffness gave good performance. Fabrics with high extensibility, such as knitted fabrics, were unsuitable in the present system, due to the cloth edge curling up around the presser foot. Two-ply panels gave better performance than single-ply panels because of their higher stiffness.

7.1.6. Handling Techniques

Techniques were developed to set up a cloth panel for the edge seaming operation. The robot located a cloth panel placed down approximately on the table, and slid it into place with the needle accurately positioned at the start of the seam. Two photocells and the two cameras mounted on the sewing machine provided visual feedback during the handling operation.

A technique for rotating the cloth about the needle was developed which was used to reduce the initial angular error of the cloth panel and to set up one cloth edge after sewing up the adjacent edge. The robot's poor absolute accuracy caused problems for this operation since sensory feedback could not be used to compensate for the robot's inadequacy.

Segmented seam production was permitted by dividing a seam up into FAR and CLOSE segments and repositioning the fingers between segments. The sewing and handling techniques were demonstrated in the production of a three-sided panel of arbitrary contour.

7.2. Capabilities and Limitations of FIGARO system

7.2.1. Introduction

An ideal flexible automatic sewing cell would have the following features :-

* Flexibility to process different shapes, sizes and fabrics.
* Capability to perform a wide range of sewing and handling operations.
* No manual intervention required between different operations or products.
* Minimal manual adjustments or maintenance.
* High reliability.
* Automatic error detection and recovery.
* Easy to integrate into a CIM environment.

Most of these features could be integrated into a commercial version of the FIGARO system. The hierarchical control arrangement that was adopted in the FIGARO system, can easily be incorporated into a CIM environment by developing an additional communication channel between the cell controller and a process supervisor. Automatic error detection and recovery capabilities require redundant and overlapping sensor systems, and extensive processing capabilities, and the FIGARO system could be extended to include these facilities. Recommendations regarding the sewing machine, which is the most unreliable component in the system and which requires frequent manual adjustment, are given in sections 7.3.3 and 7.4.3.

The flexibility of the FIGARO system and its multi-function capability is discussed in the following sections.

7.2.2. Multi-Function Capabilities

7.2.2.1. Present Capabilities

Techniques have been developed for the FIGARO system, which perform the following functions :-

1) Sewing a seam parallel to an edge of arbitrary contour.
2) Sewing a straight seam anywhere on the cloth.
3) Setting up a cloth panel for the edge seaming operation, from an approximate initial position and orientation.
4) Rotating a cloth panel about the sewing needle.
5) Withdrawinq a cloth panel from the sewinq machine after the sewing operation.

A ply separation device, developed in a separate project, was inteqrated into the end-effector in order to provide the capability to separate and pick up a single ply from a stack. A vision-based technique for placing one ply accurately on top of another is being developed in a parallel project, which could also be integrated into the FIGARO system.

7.2.2.2. Potential Capabilities

a) Additional Sewing Functions

Seams with fullness could be produced if the drop feed sewing machine was exchanged for a machine with a programmable differential feed. If a button sewing machine and a button hole machine could be added to the sewing table, without affecting the performance of any of the sewing or handling operations already developed, then two very useful functions would be added to the FIGARO repertoire. These additional machines would probably require an extension of the sewing table and inverted mounting of the robot (section 2.8.3.3). It may be necessary to mount the robot on a programmable gantry platform, which is a technique that is often used to increase a robot's working envelope.

b) Foldinq and Unfolding

Foldinq a cloth panel prior to a sewing operation and unfolding it after the operation were identified by the $(TC)^2$ project team as useful handling capabilities, which can be used, besides other purposes, to reduce the surface area of large panels to facilitate the sewing operation [17]. It is anticipated that some modification of the ply separation device will be necessary to realize these capabilities.

Since folding and unfolding are functions in which a human operator must use both hands, a robotic solution must include a degree of assistance external to the single-handed robot. A simple and effective solution might be to use the table's flotation nozzles to apply suction to the panel, at the critical stage in the handling operation. Alternatively, a portion of the extended sewing table could be designed as a folding/unfolding station based on assistance devices.

c) Pocket Setting

The existing system would require some additional development in order to set and sew up a pocket onto a panel. For example, the vision system would have to detect the edge of the pocket against the panel. Since the table's mirror surface could not be used to detect the edge, a structured light approach might be successful, in which a laser beam is projected as a narrow line from a low elevation angle. The vision system could then measure the position of the cloth edge by detecting the step in the line of light, which is due to the height differential.

7.2.3. Flexibility

Besides offering a greater range of functional capabilities, a system based on robotics and sensory feedback is more flexible and adaptable to changes in the shape, size or characteristics of the workpiece, when compared to hard automation solutions.

7.2.3.1. Present System's Flexibility

In the sewing up of a three sided sub-assembly (section 6.6), the FIGARO system demonstrated some flexibility, in that panels of different sizes and with different edge contours were successfully sewn up without any manual mechanical adjustments or software alterations. The sensory feedback control systems accommodated minor changes in fabric characteristics without requiring changes in the control parameters.

7.2.3.2. Flexibility to Shape

The present end-effector has two fingers that can be configured optimally under program control for a specific panel shape. A multi-fingered end-effector would improve the system's performance for a wider range of shapes, but the more complex device should be designed in accordance with the comments made in section 6.2.

7.2.3.3. Flexibility to Edge Contour

Although the vast majority of edge profiles found on garment panels could be sewn up satisfactorily with an optimum combination of CLOSE and FAR seam segments, the seam strategy generator (SSG) implemented in the present version of the software is unsophisticated and it will only generate a satisfactory strategy for moderately curved contours. The current SSG is embodied in the DecideSeam function, and it generates either a FAR-CLOSE or a CLOSE strategy, depending on the seam length.

In order to sew along an edge with intricately curved features, a sewing strategy would have to be specified by a programmer by writing a new version of DecideSeam for the particular seam profile, in which the combination of FAR and CLOSE segments was based on the seam profile. This is not a very satisfactory situation since the programmer would either have to arrive at a successful strategy through trial-and-error experimentation, or he would require expert knowledge of the system, its dynamic characteristics and its limitations.

Consequently, if the FIGARO system is to be used to its maximum potential and yet maintain simple task specification requirements, a much more sophisticated SSG is required to automatically generate the optimum sewing strategy for specific edge profiles (section 7.2.4).

7.2.3.4. Flexibility to Fabric Characteristics

The robotic sewing operations are sensitive to the mechanical properties of the fabric. In order to simplify the requirements of the user interface to the FIGARO system, different fabrics should be classified according to their mechanical properties, so that the optimum control parameter settings could be found experimentally for each category. Consequently, when the system is in operation, the software could automatically select the optimum control for a specified fabric category.

The present system cannot satisfactorily sew knitted fabrics or woven fabrics cut along the bias, due to excessive shear buckling. This limitation might be removed if the tension control system was improved so that the cloth tension could be kept at a much lower level (of the order of 2 to 10 gf).

The high table-to-fabric friction, which is the major factor preventing the reduction of the controlled tension level, can be reduced by adding flotation to the sewing table,or it can be eliminated by picking up the end of the panel and holding it in the air between clips (section 4.3.1).

7.2.4. A Sewing Strategy Generator (SSG)

The requirement for an automatic, optimizing, sewing strategy generator was described in section 7.2.3.3. This sophisticated SSG would require a reasoning and decision-making capability, which could be developed using artificial intelligence (AI) techniques.

The SSG would require knowledge of the edge contour, which could be provided in one of two ways. An overhead camera system could provide an image of the cloth panel which would be interpreted in real time by a vision processing system. The edge contour shape would be extracted from the image using an edge detection algorithm.

Alternatively, in an advanced CIM system, the shape of all the cloth panels would already be on record in the CADCAM database which generated the program for the automatic cutting machine, and this database could be interrogated by the sewing cell controller.

Several experimental AI programs have been reported which can perform the "Robot Task Planning" function [73,74]. A Task Planner is given a description of the goal (e.g. "put the red block on top of the white block", or in this application "perform an edge seam on the left hand edge of the panel"), and it will decide how the robot can achieve the goal and specify the robot motion sequence, relevant locations and other parameters (in this case, the seam strategy).

A Task Planner requires a World Model, a Knowledge Base and a reasoning algorithm. In the case of an SSG, the World Model would be a description of the edge contour and knowledge about the limitations and capabilities of the FIGARO system, and the Knowledge Base would contain a set of empirical rules to guide the reasoning process to find the optimum sewing strategy. An AI programming language, such as PROLOG which is based on predicate logic and has a built-in backtracking inference engine, would facilitate the development of the SSG.

7.3. Commercialization Considerations

The FIGARO development is based on an ambitious approach to solving the clothing automation problem, and at this early stage, the development of technical solutions and an investigation into the fundamental handling problems are the foremost requirements. Although, the present experimental system is not expected to be commercially attractive, some comments can be made as to the potential for commercial exploitation of the developments in the future.

7.3.1. Speed

Sewing speeds of 3000 rpm have been achieved for moderately curved cloth panels, and implementation of modifications recommended above should increase the sewing speed or the rates of curvature further. This performance is comparable to the sewing speed that a human operator can achieve under similar circumstances, but an operator using an edge guide and dedicated automatic edge seamers can achieve up to 6000 rpm for similar curvatures.

Dedicated automation devices are usually faster than the equivalent flexible automation system because there is a trade-off between speed and flexibility.

The system can locate and accurately set up a panel for an edge seaming operation within 20 to 30 seconds, and there is considerable scope for reducing the times for this and other handling operations. Since fabric handling accounts for up to 80 % of an operator's time [10], improving the fabric handling times is more important than improving the sewing speeds. An overhead camera, with associated vision processing hardware and software, and a faster and more accurate robot should reduce the fabric handling times to timings comparable with a human operator.

7.3.2. Cost

7.3.2.1. General Comments

The FIGARO approach is inherently expensive when compared to hard automation solutions, since it involves an adaptive robot, complex sensor systems, multi-processor architecture, extensive real time processing, and large and complex software support. This is common, however, to most applications of robotics and flexible automation, and particularly in the case of complex systems involving adaptive or intelligent control. The high initial costs have to be justified commercially by high life expectancy and utilization of the system [75].

Simulation experiments can assist in determining the commercial viability of different production methodologies.

The small batch flexibility of a robotic cell is best exploited within a CIM environment, and therefore the viability of complex intelligent robotic assembly cells is closely tied to the development and implementation of CIM systems.

7.3.2.2. Comments Relating to the Clothing Industry

The Clothing Industry has a relatively low level of investment in plant and machinery as a proportion of total sales over time, compared with other sectors of industry [1]. Several factors encourage this situation, such as low added value ratio on products, unacceptability of shift working among the work force, short batch production, etc. Consequently, the commercial viability of a sophisticated robotic sewing system is unlikely in the near future.

Nevertheless, there are several factors that indicate that this situation will change :-

a) Complex and expensive CAD/CAM equipment is becoming commonplace in cutting rooms (section 1.3.1).

b) Semi-automatic sewing units are in widespread use despite their limited flexibility and relatively high cost (section 1.3.2.2).

c) Computerized conveyor systems have been adopted and integrated into production control systems, which is an important step towards developing a CIM environment (section 1.3.3).

d) Large scale R & D projects are underway in Japan, Europe and the USA to develop flexible clothing automation (section 1.4), confirming that it is widely perceived that this technology is required urgently.

The $(TC)^2$ approach, which is technically more conservative than the adaptive robot approach, has the disadvantage that an expensive robot is restricted to handling operations, and that a complex expensive sewing module is also required. The adaptive robot approach, which maximizes the use of the expensive robot so that the sewing machine and other peripherals can remain relatively cheap and simple, is much more ambitious.

If all the technical problems can be solved so that the cell's handling time can match that of a human operative, then it will replace three operatives, assuming round-the-clock (i.e. three-shift) operation of the cell. The current cost of three operatives is approximately £30,000 per year, and the FIGARO project has shown that an industrial version could well have a capital cost below the £60,000 target, which gives a two year payback. Consequently, the adaptive robot approach is well worth pursuing.

7.3.3. Other Considerations

Sewing machines are notoriously unreliable and they have frequent stoppaqes for thread and needle breakages, tension adjustments and bobbin replacements. This characteristic is a major problem in the automation of the sewing room, which can be tackled in two ways.

a) Each sewing machine fault could be detected, identified and rectified automatically. Automatic bobbin changers and needle threading mechanisms have been developed [8] which could be integrated into the cell. An artificial intelligence capability may be necessary to ensure that system faults are interpreted correctly and that suitable corrective action is chosen.

b) Alternatively, each sewing cell could have two sewing heads, either of which could be rotated into place. One of the sewing heads could then be threaded and adjusted manually without holding up production. Nilsson [16] describes a sewing room with general purpose sewing cells, in which the material flow could be modified automatically as cells were removed from production for rethreading etc.

7.4. Recommendations

7.4.1. Robot

The PUMA 560 robot is unsatisfactory for robotic sewing and handling applications, due to its poor off-line programming

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

accuracy (section 2.4.1). Since the end-effector is maintained in a perpendicular orientation relative to the sewing table for all robot motions, a 4 axis robot would suffice and the PUMA robot has 2 redundant degrees of freedom. A 4 axis SCARA type robot is inherently stiffer and more accurate than the PUMA design, and its real time motion control calculations are simpler since there are only 4 axes to control.

The major attraction of the PUMA robot was its VAL II control system which permits real time path control of the robot. The Adept SCARA robot is now available with the VAL II control system, and the Adept robot system achieves very high off-line programming accuracy by incorporating the actual dimensions and angular offsets of each specific robot into the control system's model. The advantages of the Adept robot over the PUMA for the FIGARO application, are summarized below :-

* 16 ms handshake cycle time, instead of 28 ms
* higher accuracy
* no singularities
* faster maximum tool velocity and acceleration
* higher rigidity

7.4.2. Cell Controller

The workload on the IBM AT was considerable, and the performance of the robotic sewing operation suffered from insufficient processor power. A commercial implementation would require much more processor power for additional communication channels, automatic error detection and correction, etc.

A far more powerful processor could be selected for the cell controller e.g. the new 32-bit micro-processors, (80386, 60030, etc.). The workload on the cell controller should be further reduced by delegating the management of the ALTER and supervisory communication channels to dedicated processors, e.g. a microcontroller and a chip of dual ported RAM could provide a communications support sub-system (section 3.3.3).

7.4.3. Sewing Machine

Additional sewing functions and a reduction in the number of manual adjustments required could be obtained by replacing the lockstitch machine with a machine that can provide differential top and bottom feed under external programmable control and that can provide a programmable stitch length. The differential top and bottom feed would permit production of seams with fullness, and the programmable stitch length would permit production of condensed stitching and reduce the need for frequent manual adjustments and check-up.

7.4.4. Workstation

Flotation nozzles should be incorporated into the table before and after the sewing head. The nozzles after the sewing head should be directed to push the cloth away from the needle during sewing (section 5.4.4.1). If the nozzles in the main area of the table could be programmable to provide either suction or floatation, then the system will have additional flexibility and reliabilty.

The robot could be mounted inverted from a gantry to increase its workspace.

7.4.5. Future Work

Many recommendations for further research and development have been suggested earlier, and they are summarized as follows :-

1) Extend tension control to a wider range of fabrics.
2) Improve edge seaming performance
3) Reduce timings for fabric handling operations.
4) Develop an SSG to provide AI task planning capability.
5) Develop folding and unfolding techniques.
6) Add overfeed and stitch condensation capabilities.
7) Demonstrate production of a jacket sleeve.
8) Measure mechanical properties of fabrics and determine tension control parameters for each fabric category.
9) Add button-hole and button-sewing machines.
10) Develop handling and sewing techniques for setting and sewing up a pocket on a back panel.

7.5. Conclusion

An experimental flexible robotic sewing cell was developed which consisted of an adaptively controlled robot, a hierarchy of controllers, and several sensory inputs.

Techniques for sewing contoured edge seams (and of course straight seams) were developed, based on sensory feedback control systems which maintain the cloth tension and the seam width during sewing. A cloth tension sensor, vision processing software and a two-fingered fabric steering end-effector were developed for the robotic sewing operations.

Fabric handling techniques have also been developed including detecting a cloth panel, presenting it to the sewing machine, accurately setting up the cloth for an edge seam operation, rotating the cloth about the needle, and removing the cloth from the machine after the sewing operation.

The project has successfully demonstrated technical solutions to the flexible automation of clothing assembly, in which the robot performed all the fabric handling and control needed in the sewing assembly operations. Future developments of this approach to clothing automation have been clarified as a result of this research.

REFERENCES

1 Cave P., NEDO, "Economic Overview of the Garment Industry", Proc. Conf. on Automation of Garment Manufacture, Leeds 1986, pp 6-11.

2 Walter C.H., Marks & Spencer PLC, " A Retailer's View of the Requirement of Research in Garment Manufacture", Proc. Conf. on Automation of Garment Manufacture, Leeds 1986, pp 42-50.

3 Tyler D.J., "Flexible Apparel Automation and Japanese Initiatives", Hollings Apparel Industry Review, 1985, Vol 2, No 1, pp 7-24.

4 Weston L., "Some Observations on the Swedish Clothing Industry", Hollings Apparel Industry Review, 1985, Vol 2, No. 2, pp 153-170.

5 Saibel M., "Research & Development Program for the Japanese Apparel Industry", AAMA Apparel Research Journal, Bobbin, October 1977, p 138.

6 "Research & Development in the Apparel Industry", AAMA Apparel Research Journal, Bobbin, October 1977, pp 123-137.

7 Anon, "Automation in Apparel", Bobbin, 1982, Vol 23, No 5, pp 66a-66h.

8 Lower J.M., Singer Inc., "Automation Heard around the World", Bobbin, 1985, Vol 26, No 8, pp 78-81.

9 Tredwin P., "Computerised Garment Manufacture", Proc. World Conf., The Textile Inst., May 1985, London.

10 Grills R., Brown S., "Productivity in Sewing Operations", Shirley Institute Publication S20, 1975.

11 Ogawa S., "Japan's Automated Sewing System: A National Research and Design Project",Bobbin, 1984, Vol 25, No 6, pp 82-102.

12 Sinclair D., "Stand Up and Sew", Apparel International, 1982, Vol 2, No 2, pp 4-7.

13 Wong P.C., Hudson P.R.W., "The Australian Robotic Sheep Shearing Research and Development Program", Robots 7, 1983, pp 10-56 to 10-63.

14 Hauber F.W., Pfaff, "An Organised Method for Looking at Sewing Machines", Bobbin, August 1978, pp 114-118.

15 "Technology and the Garment Industry", NEDO report, 1971.

16 Edberg B., Nilsson N., "Computerised Clothing Manufacturing: A Means for Survival", Proc. Annual World Conf., The Textile Institute, May 1985, London.

17 Abernathy F.H., Pippins D., "(TC)² Apparel, Textile and Education at its best", Bobbin, 1986, Vol 28, No 1, pp 162-168.

18 Bernardon E., Kondolean A., "Real Time Robotic Control for Apparel Manufacturing", Charles Stark Draper Laboratory Inc., 1985, pp 1-20. or Robots 9, Detroit 1985, pp 4-46 to 4-66

19 Berkstresser G.A., Takeachi K., "Japan's Automated Apparel Manufacturing System Research Project", Bobbin, 1983, Vol 25, No 3, pp 75-84.

20 "PUMA 500 MK2 Robot System Technical Manual", Unimation (Europe) Ltd., England, 506-9057/58.

21 "Unimate PUMA Robot, Vol. 1 Technical Manual 398H1A", Unimation Inc., USA, 1981.

22 El-Zorkany H., Liscano R., Tondu B., Sawatzky G., "Sensor-based Location and Trajectory Specification and Correction in Robot Programming", Proc. Conf. ISIR 16, Brussels 1986, pp 643-656.

23 Desroches A., "Introduction to Robot Dynamics and Control", IEEE Control Systems, Vol 22, No 1, 1984.

24 Mudge T.N., Turney J.L., "Unifying Robot Arm Control", IEEE Trans. Industr. Applic. Vol 1A-20, No. 6, 1984.

25 Paul R.P., "Robot Manipulators: Mathematics, Programming and Control", Cambrige Press, 1981.

26 Dubowsky S., Desforges D. T., "The application of model-referenced adaptive control to robotic manipulators.", Jnl. Dynamic Systems Measurement and Control, Vol 101, 1979, pp 193-299.

27 Freund E., "Fast non-linear control for robots", Proc. International Research on Robotics Research", Vol 1, No 1, 1982, pp 65-78.

28 Critchlow A.J., "Introduction to Robotics", Macmillan, 1985.

29 Johnson D.G., Hill J.J., "Sensory Level Programming: A New Software System for Improved Control of a Sensory

Industrial Robot", Proc. Conf. ROVISEC 5, Amsterdam, 1985, pp 383-391.

30 Dupourque V., Ishacian O., "Controlling Multi-Robot Applications from UNIX", Proc. Conf. ISIR 16, Brussels 1986, pp 197-208.

31 Van Brussel H., De Winter D., Thielemans J., Valckenaers P., Claus H., "Introducing Flexibility in Assembly Systems", Proc. Conf. ISIR 16, Brussels 1986, pp 557-567.

32 Smith R.C., Nitzan D., "A Modular Programmable Assembly Station", Proc. Conf. ISIR 13, Chicago 1983, pp 5.53-5.75.

33 Albus J.S., McLean C.R., Barbera A.J., Fitzgerald M.L., "Hierarchical Control for Robots in an Automated Factory", Proc. Conf. Robots 7, 1983, pp 13.29-13.43.

34 Symcox G., "Interfacing Robots with and without MAP - a Case Study", Proc. Conf. ISIR 16, Brussels 1986, pp 209-218.

35 Paul R.P., Shimano B., Mayer G.E., "Kinematic control equations for simple manipulators", IEEE Trans. Systems Man Cybernetics, Vol SMC-11, June 1981, pp 449-460.

36 Bazerghi A., Goldenberg A.A., Apkarian J., "An Exact Kinematic Model of PUMA 600 Manipulator", IEEE Trans. Systems Man Cybernetics, Vol SMC-14, No 3, May/June 1984, pp 483-487.

37 "AMX-86 Multitasking Executive Reference Manual,

PN855-9", KADAK Products Ltd., 206-1847 West Broadway Avenue, Vancouver, B.C., Canada, 1983.

38 "User's Guide to VAL II, Version 1.1, 398T1", Unimation Inc., USA, 1984.

39 Gruver W.A., Soroka B.I., Craig J.J., Turner T.L., "Industrial Robot Programming Languages: A Comparative Evaluation", IEEE Trrans. Systems Man and Cybernetics, Vol SMC-14, No 4, July/August 1984, pp 565-570.

40 "IBM AT Technical Reference Manual, PN 1502243", 1984.

41 Tanenbaum A.S., "Computer Networks", Prentice Hall, 1981, pp 15-21.

42 Elgazzar S., "Efficient Kinematic Transformations for the PUMA 560 Robot", IEEE Jnl. Robotics and Automation, Vol RA-1, No 3, 1985, pp 142-151.

43 Demers K.P., Walsh P.M., "Sensor-based Real-time Robot Control Systems", Robotics Today, Vol 5, No 3, 1983, pp 69-72.

44 Hill J., Park W.T., "Real Time Control of a Robot with a Mobile Camera", Proc. Conf. ISIR 7, October 1977, pp 233-246.

45 Shimano B.E., Geschke C.G., Spalding C.H., Smith P.G., "A Robot Programming System Incorporating Real Time and Supervisory Control: VAL II", Proc. Conf. Robots 9, Vol 2, Detroit 1984, pp 20.103-20.119.

46 Dario P., De Rossi D., "Tactile Sensors and the Gripping Challenge", IEEE Spectrum, Aug 1985, pp 46-52

47 Harmon L.D., "Automated Tactile Sensing", Intl. Jnl. of Robotics Research, Vol 1, No 2, 1982, pp 3-32.

48 Van Brussel H., Belian H., Thielemans H., "Force Sensing for Advanced Robot Control", Proc. Conf. ROVISEC 5, Amsterdam, 1985, pp 59-68.

49 Lestelle D., "Gripper with Finger Built-in Force/Torque Sensors", Proc. Conf. ROVISEC 5, Amsterdam, 1985, pp 69-77.

50 Rosen C.A. et al., "Exploratory Research in Advanced Automation", Reports 1 to 5, Stanford Research Institute under National Science Foundation Grant G138100X, Dec 1973 to Jan 1975.

51 Feldmann K., Classe D., "Sensor Aided Robot Programming", Proc. Conf. ROVISEC 5, Amsterdam, 1985, pp 369-382.

52 Watson P.C., Drake S.H., "Pedestal and Wrist Force Sensors for Automatic Assembly", Proc. Conf. ISIR 5, Chicago, 1975, pp 501-511.

53 "Materials Selector", September 1972, pp 35-134.

54 "Kyowa Strain Gauge and Temperature Sensor, Instruction Manual", Kyowa Corp., Japan.

55 Horowitz P., Hill W., "The Art of Electronics", Cambridge University Press, 1980.

56 MacCarthy B.L., Sharp J.M., Burns N.D., "A Constrained Optimization Technique to Improve the Performance of

Strain Gauge Transducers", Proc. Instn. Mech. Engnrs., Vol 200, No C2, 1986.

57 Kuo B.C., "Automatic Control Systems", Prentice-Hall, 1975, pp 295-302.

58 "A.I.E.E. Committee Report", Elec. Eng., Vol 70, October 1951, p 905.

59 Nordby, H.A., "The Load-Elongation Properties of Fabrics with Special Reference to Hysteresis", Ph.D Thesis, University of Leeds, 1968.

60 Hearle J.W.S., Grosberg P., Backer S., "Structural Mechanics of Fibres, Yarns, and Fabrics", Volume 1, Wiley Interscience, 1969, pp 339-369.

61 "Users Manual - IBM AT Interface Card for the I-SIGHT Cameras", Electronic Automation Ltd., Hull, 1986.

62 Loughlin C., Hudson E., "Eye in Hand Robot Vision", Proc. Conf. ISIR 13, 1983, pp 263-270.

63 Naylor, P., Private Communication, Electronic Automation Inc., 1986.

64 Porat, I., Private Communication, Univ. of Leeds, 1986

65 Torgerson E., Paul F.W., "Vision Guided Robotic Fabric Manipulation for Apparel Manufacturing", to be presented at the 1987 IEEE Intl. Conf. on Robotics and Automation.

66 Parker J.K, Dubey R., Paul F.W., Becker R.J., "Robotic Handling for Automated Garment Manufacturing", Trans.

ASME Jnl. of Engineering for Industry, 1982, pp 1-6.

67 Gershon D., Porat I., "Robotic Sewing Using Multi-Sensory Feedback", Proc. Conf. ISIR 16, Brussels, 1986, pp 823-834.

68 Di Stefano J., Stubberud A., Williams I., "Feedback and Control Systems", Schaum Series, McGraw Hill, 1976, pp 295-302.

69 Healy M., "Principles of Automatic Control", The English Universities Press, 1975, pp 131-156.

70 Ben Ari M., "Principles of Concurrent Programming", Prentice Hall, 1982.

71 Baker F., "The Causes of Seam Pucker", Bobbin, 1978, Vol 20, No 3, Nov, pp 188-192.

72 Porat I., Iype C., Gershon D., Moghaddassi M.N., Grosberg P. "Clothing Automation at the University of Leeds", Proc. Conf. on Automation of Garment Manufacture, Leeds 1986, pp 36-40.

73 Lozano-Perez T., "Task Planning", in "Robot Motion: Planning and Control", ed. Brady M., MIT Press, 1982, pp 473-498.

74 Winston P., "Artificial Intelligence", Addison Wesley, 1977, pp 158-165.

75 Scott P.B., Little A.D., "Guidelines for Economic Justification of Flexible Automation", Proc. Conf. ISIR 16, Brussels, 1986, pp 1045-1056.

APPENDIX A

MISCELLANEOUS SOFTWARE MODULES

A.1. Software Versions

The version numbers of the various software products that were used in this project are listed in table A-1.

Product	Vendor	Version	Year
AMX-86	KADAK Ltd.	1.1	1985
C compiler	Lattice Corp.	3.1	1986
LINKER	Microsoft Inc.	2.4	1983
TURBO PASCAL	Borland Intnl	3.01A	1985
Assembler	IBM	1.0	1981

Table A-1: Software Version Nos.

A.2. AMX C Interface Prefix File

AMX-86 requires a prefix file to be used at link time, which ensures that the AMX segment definitions are compatible with the C compiler (see section 1.10 of AMX C Interface Manual). The prefix file provided was intended for version 1.15 of the Lattice C compiler and is incompatible with the different segment naming convention implemented in version 3. A small modification of the AMX prefix file rendered it compatible with version 3 of the C compiler, and the modified file is listed below :-

```
        NAME    AMX2P
        PAGE    60,132          ;PAGE/LINE SIZE
        TITLE   AMX2P - PREDEFINE SEGMENTS FOR LATTICE C LINKING
;
;       This version is a modification of KADAK's AMCF865P.ASM
;       (version 1.1, 1985).
;       This prefix file has been modified so that AMX is now
;       compatible with version 3 of the LATTICE C compiler,
;       the IBM linker and the Microsoft linker.
;
;
REV     EQU     11H             ;REVISION 1.1
;
; DEFINE DUMMY SEGMENTS WHICH WILL RESULT IN ALL AMX86 SEGMENTS
; AND ALL LATTICE C SEGMENTS BEING LOADED INTO MEMORY IN THE
; CORRECT ORDER.
;
; THE C STACK SEGMENT MUST BE THE ONLY STACK SEGMENT
; WITH CLASS 'STACK'.  IT MUST ALSO BE LOCATED AS THE
; LAST SEGMENT IN THE LINKED MODULE IN ORDER TO PROPERLY
```

```
; ALLOCATE STACK AND HEAP.
;
; NOTE: THE AMX86 MEDIUM TASK STACK SEGMENT MAY BE FORCED BY
;       YOUR AMX86 CONFIGURATION MODULE TO BE PART OF DGROUP.
;       IN THIS CASE, SEGMENTS OF CLASS 'MSTACK' WILL BE
;       AT THE BASE OF DGROUP AND SEGMENTS OF CLASS 'DATA'
;       WILL IMMEDIATELY FOLLOW THEM IN DGROUP.
;
; NOTE: THE IBM MASM ASSEMBLER ORGANIZES THE SEGMENTS
;       ALPHABETICALLY BY SEGMENT NAME.  THEREFORE, SEGMENT
;       NAMES HAVE BEEN CHOSEN TO DEFINE THE PREFERRED ORDER
;       OF THE SEGMENTS IN THE OBJECT MODULE.
;       THE MICROSOFT LINKER ALLOCATES SEGMENTS IN THE ORDER
;       IN WHICH SEGMENT NAMES AND CLASSES ARE ENCOUNTERED.
;
AAACODE SEGMENT BYTE 'CODE'       ;AMX86 CODE SEGMENT
AAACODE ENDS
;
AAACODL SEGMENT BYTE              ;LATTICE C CODE SEGMENT
AAACODL ENDS
;
;    The following segment declaration forces the linker
;    to arrange the segments in the correct order
;
AAACODP SEGMENT BYTE PUBLIC 'PROG'    ;LATTICE C SEGMENT (v. 3)
AAACODP ENDS
;
AAASTK1 SEGMENT WORD 'TSTACK'    ;AMX86 LARGE MODEL TASK STACK
AAASTK1 ENDS
;
AAASTK2 SEGMENT WORD 'MSTACK'    ;AMX86 MEDIUM MODEL TASK STACK
AAASTK2 ENDS
;
;                                ;AMX86 PC SUPERVISOR DATA SEGMENT
AMPCDATA SEGMENT WORD PUBLIC 'DATA'
AMPCDATA ENDS
;
        END
```

A.3. AMX Configuration Module

The AMX executive requires a configuration module to be loaded with each application, as described in section 2.3.2.7. The configuration details are summarized below, followed by the listing of the actual configuartion module :-

A.3.1. Summary of Configuration Details

TASK #	TASK NAME	TASK ADDR	TASK STACK	TASK MODEL	LEV0	QUEUE LEV1	DEPTH LEV2	LEV3
0	TMR	AMTMRT	400	LARGE	0	0	0	0
1	RXMG	STRXMG	400	LARGE	0	0	0	0

2	TXMG	STTXMG	400	LARGE	0	0	0	0
3	COMM	STCOMM	500	LARGE	0	0	0	0
4	SEW	STSEW	500	LARGE	0	0	0	0
5	MAKE	STMAKE	500	LARGE	0	0	0	0
6	CONT	STCONT	500	LARGE	0	0	0	0
7	POST	STPOST	500	LARGE	0	0	0	0
8	PRNT	STPRNT	400	LARGE	4	3	320	3

RESTART PROCEDURES:

AMTDRR
AMRMRR
AABIA
RTIMER
RPCOM
RPCAMR
RPSEW

CLOCK FREQUENCY IN HZ. IS 18.
CLOCK TICKS PER SYSTEM TICK IS 1.
TIME/DATE MAINTENANCE IS INCLUDED.
TIME/DATE PERIOD IN SYSTEM TICKS IS 18.

TIMERS AND TIMER PROCEDURE ADDRESSES:

TIMER	TIMER PROCEDURE
TMTD	AMTDTR
TMN01	

RESOURCE MANAGER IS INCLUDED.

BUFFER MANAGER IS INCLUDED.

POOL#	# BUFFERS	SIZE
0	200	150
1	10	160

500 SYSTEM QUEUE PARAMETER BLOCKS ALLOCATED.
EXECUTIVE STACK IS 400 WORDS.
INTERRUPT SERVICE PROCEDURE STACK IS 450 WORDS.

A.3.2. Configuration Module

```
        TITLE   CONT1.C     5/2/87
;
;AN AMX86 CONFIGURATION MODULE DEFINING ALL
;TASKS, TIMERS, QUEUES, STACKS, ETC. REQUIRED
;BY AMX86 FOR PROPER OPERATION
;
```

```
;TASK ADDRESSES
;
        EXTRN   AMTMRT:FAR        ;TASK # 0 AMX86 TIMER TASK
        EXTRN   STRXMG:FAR        ;TASK # 1
        EXTRN   STTXMG:FAR        ;TASK # 2
        EXTRN   STCOMM:FAR        ;TASK # 3
        EXTRN   STSEW:FAR         ;TASK # 4
        EXTRN   STMAKE:FAR        ;TASK # 5
        EXTRN   STCONT:FAR        ;TASK # 6
        EXTRN   STPOST:FAR        ;TASK # 7
        EXTRN   STPRNT:FAR        ;TASK # 8
;
;RESTART PROCEDURE ADDRESSES
;
        EXTRN   AMTDPC:FAR        ;TIME/DATE FOR IBM PC DOS
        EXTRN   AMRMRR:FAR        ;RESOURCE MANAGER
        EXTRN   AABIA:FAR         ;BUFFER MANAGER
        EXTRN   RTIMER:FAR        ;USER RESTART PROCEDURES
        EXTRN   RPCOM:FAR
        EXTRN   RPCAMR:FAR
        EXTRN   RPSEW:FAR
;
;APPLICATION TIMER PROCEDURES
;
        EXTRN   AMTDTR:FAR        ;TIME/DATE TIMER PROCEDURE
;
;
        PAGE
;THE AMX86 PARAMETER SEGMENT
AMXPAR  SEGMENT WORD 'CODE'
;
;ENTRY POINTS REQUIRED BY AMX86
;
        PUBLIC  AMTDT             ;TASK DEFINITION TABLE
        PUBLIC  AMRPL             ;RESTART PROCEDURE LIST
        PUBLIC  AMNUMQ            ;NUMBER OF QUEUE BLOCKS
        PUBLIC  AMCLKP            ;CLOCK PERIOD = # OF INTERRUPTS
        PUBLIC  AMTMRR            ;TIMER PROCEDURE LIST
        PUBLIC  AMISTP            ;AMX86 INTERRUPT STACK POINTER
;
;TIME/DATE PARAMETER TABLE ENTRY POINTS
;
        PUBLIC  AMTDFQ            ;TIMER FREQUENCY
        PUBLIC  AMTDTM            ;DISPLACEMENT OF TIME/DATE TIMER
        PUBLIC  AMTDRA            ;A(TIME/DATE RAM BLOCK)
        PUBLIC  AMTDSH            ;A(USER TIME/DATE SCHEDULER)
;
;TABLE OF APPLICATION TIMER DISPLACEMENTS ENTRY POINTS
;
        PUBLIC  TMTD              ;TIME/DATE TIMER
        PUBLIC  TMNO1
;
;TABLE OF INTEGER TASK NUMBERS ENTRY POINTS
;
        PUBLIC  TNTMR             ;TASK # 0 AMX86 TIMER TASK
        PUBLIC  TNRXMG            ;TASK # 1
        PUBLIC  TNTXMG            ;TASK # 2
        PUBLIC  TNCOMM            ;TASK # 3
```

```
        PUBLIC  TNSEW               ;TASK # 4
        PUBLIC  TNMAKE              ;TASK # 5
        PUBLIC  TNCONT              ;TASK # 6
        PUBLIC  TNPOST              ;TASK # 7
        PUBLIC  TNPRNT              ;TASK # 8
;
;RESOURCE MANAGER ENTRY POINTS
;
        PUBLIC  AMRDT               ;RESOURCE DEFINITION TABLE
;
;BUFFER MANAGER ENTRY POINTS
;
        PUBLIC  AAPDT               ;POOL DESCRIPTION TABLE
;
        PAGE
;AMX86 TASK DEFINITION TABLE
AMTDT   LABEL   DWORD
;
;AMX86 TIMER TASK (#0) IS THE HIGHEST PRIORITY
;TASK # 0
        DD      AMTMRT              ;A(AMX86 TIMER TASK)
        DD      SPTMR               ;A(TIMER TASK STACK)
        DW      0                   ;TASK ATTRIBUTES
        DW      0                   ;LEVEL 0 (UNUSED)
        DW      0                   ;LEVEL 1 (UNUSED)
        DW      0                   ;LEVEL 2 (UNUSED)
        DW      0                   ;LEVEL 3 (UNUSED)
;
;TASK # 1
        DD      STRXMG              ;START ADDRESS
        DD      SPRXMG              ;STACK ADDRESS
        DW      0                   ;TASK ATTRIBUTES
        DW      0                   ;LEVEL 0 (UNUSED)
        DW      0                   ;LEVEL 1 (UNUSED)
        DW      0                   ;LEVEL 2 (UNUSED)
        DW      0                   ;LEVEL 3 (UNUSED)
;
;TASK # 2
        DD      STTXMG              ;START ADDRESS
        DD      SPTXMG              ;STACK ADDRESS
        DW      0                   ;TASK ATTRIBUTES
        DW      0                   ;LEVEL 0 (UNUSED)
        DW      0                   ;LEVEL 1 (UNUSED)
        DW      0                   ;LEVEL 2 (UNUSED)
        DW      0                   ;LEVEL 3 (UNUSED)
;
;TASK # 3
        DD      STCOMM              ;START ADDRESS
        DD      SPCOMM              ;STACK ADDRESS
        DW      0                   ;TASK ATTRIBUTES
        DW      0                   ;LEVEL 0 (UNUSED)
        DW      0                   ;LEVEL 1 (UNUSED)
        DW      0                   ;LEVEL 2 (UNUSED)
        DW      0                   ;LEVEL 3 (UNUSED)
;
;TASK # 4
        DD      STSEW               ;START ADDRESS
        DD      SPSEW               ;STACK ADDRESS
```

```
        DW      0               ;TASK ATTRIBUTES
        DW      0               ;LEVEL 0 (UNUSED)
        DW      0               ;LEVEL 1 (UNUSED)
        DW      0               ;LEVEL 2 (UNUSED)
        DW      0               ;LEVEL 3 (UNUSED)
;
;TASK # 5
        DD      STMAKE          ;START ADDRESS
        DD      SPMAKE          ;STACK ADDRESS
        DW      0               ;TASK ATTRIBUTES
        DW      0               ;LEVEL 0 (UNUSED)
        DW      0               ;LEVEL 1 (UNUSED)
        DW      0               ;LEVEL 2 (UNUSED)
        DW      0               ;LEVEL 3 (UNUSED)
;
;TASK # 6
        DD      STCONT          ;START ADDRESS
        DD      SPCONT          ;STACK ADDRESS
        DW      0               ;TASK ATTRIBUTES
        DW      0               ;LEVEL 0 (UNUSED)
        DW      0               ;LEVEL 1 (UNUSED)
        DW      0               ;LEVEL 2 (UNUSED)
        DW      0               ;LEVEL 3 (UNUSED)
;
;TASK # 7
        DD      STPOST          ;START ADDRESS
        DD      SPPOST          ;STACK ADDRESS
        DW      0               ;TASK ATTRIBUTES
        DW      0               ;LEVEL 0 (UNUSED)
        DW      0               ;LEVEL 1 (UNUSED)
        DW      0               ;LEVEL 2 (UNUSED)
        DW      0               ;LEVEL 3 (UNUSED)
;
;TASK # 8
        DD      STPRNT          ;START ADDRESS
        DD      SPPRNT          ;STACK ADDRESS
        DW      0               ;TASK ATTRIBUTES
        DW      4               ;LEVEL 0
        DW      3               ;LEVEL 1
        DW      320             ;LEVEL 2
        DW      3               ;LEVEL 3
;
        DW      2 DUP(0FFFFH)   ;END OF TASKS
;
;TABLE OF INTEGER TASK NUMBERS
;
TNTMR   DW      0
TNRXMG  DW      1
TNTXMG  DW      2
TNCOMM  DW      3
TNSEW   DW      4
TNMAKE  DW      5
TNCONT  DW      6
TNPOST  DW      7
TNPRNT  DW      8
;
;AMX86 RESTART PROCEDURE LIST IN ORDER OF EXECUTION
;
```

```
        EVEN
AMRPL   LABEL   DWORD
        DD      AMTDPC          ;TIME/DATE FOR IBM PC DOS
        DD      AMRMRR          ;RESOURCE MANAGER
        DD      AABIA           ;BUFFER MANAGER
        DD      RTIMER          ;USER RESTART PROCEDURES
        DD      RPCOM
        DD      RPCAMR
        DD      RPSEW
;
        DW      2 DUP(OFFFFH)   ;END OF LIST
;
AMNUMQ  DW      500             ;# OF SYSTEM Q PARAMETER BLOCKS
AMCLKP  DW      1               ;CLOCK PERIOD = # OF INTERRUPTS
AMISTP  DD      AMISTK          ;AMX86 INTERRUPT STACK POINTER
;
;AMX86 APPLICATION TIMER PROCEDURE LIST
;
AMTMRR  LABEL   DWORD
        DD      AMTDTR          ;TIME/DATE TIMER PROCEDURE
        DD      TRDMY
;
        DW      2 DUP(OFFFFH)   ;END OF LIST
;
TRDMY   PROC    FAR
        RET
TRDMY   ENDP
;
;TABLE OF APPLICATION TIMER DISPLACEMENTS
;
TMTD    DW      0               ;TIME/DATE TIMER
TMNO1   DW      2
;
;TIME/DATE USER PARAMETER TABLE
        EVEN
;
AMTDFQ  DW      18              ;TIMER FREQUENCY
AMTDTM  DW      0               ;DISPLACEMENT OF TIME/DATE TIMER
AMTDRA  DD      TDRAM           ;A(TIME/DATE RAM BLOCK)
AMTDSH  DW      2 DUP(OFFFFH)   ;NO USER TIME/DATE SCHEDULER
;
;AMX86 RESOURCE DEFINITION TABLE
;
        EVEN
AMRDT   LABEL   WORD
        DW      0               ;NUMBER OF RESOURCES
;
;RESOURCE IDENTIFICATION NUMBER TABLE
;
;
;BUFFER POOL DESCRIPTION TABLE
;
        EVEN
AAPDT   LABEL   WORD
        DW      2               ;NUMBER OF POOLS
        DD      RAMO            ;POINTER TO RAM AREA FOR POOL # 0
        DW      200             ;NUMBER OF BUFFERS IN POOL # 0
        DW      150             ;SIZE OF BUFFERS IN POOL # 0
```

```
        DD      RAM1                ;POINTER TO RAM AREA FOR POOL # 1
        DW      10                  ;NUMBER OF BUFFERS IN POOL # 1
        DW      160                 ;SIZE OF BUFFERS IN POOL # 1
;
AMXPAR  ENDS                        ;END OF AMX86 PARAMETER SEGMENT
;
        PAGE
;THE AMX86 DATA SEGMENT
;
AMXDATA SEGMENT WORD 'DATA'
;
        PUBLIC  AMDATA              ;ENTRY POINT FOR AMX86 USE
;
AMDATA  LABEL   WORD
NT      EQU     9                   ;# OF TASKS IN SYSTEM
QB      EQU     500                 ;# OF QUEUE BLOCKS IN SYSTEM Q
TQ      EQU     346                 ;# OF WORDS REQUIRED FOR TASK Q'S
NTM     EQU     2                   ;# OF APPLICATION INTERVAL TIMERS
;
        DW      32 DUP(?)           ;AMX86 PRIVATE STORAGE
        DW      (NT*32)+2 DUP(?)  ;TASK CONTROL BLOCKS
        DW      (QB*9)+4 DUP(?)   ;AMX86 SYSTEM QUEUE
        DW      TQ DUP(?)           ;TASK QUEUE STORAGE
        DW      NTM DUP(?)          ;TIMER LIST
;
;TIME/DATE RAM BLOCK
;
TDRAM   DB      9 DUP(?)
;
AMXDATA ENDS                        ;END OF AMX86 DATA SEGMENT
;
        PAGE
;AMX86 STACK SEGMENTS
;
AMXESTK SEGMENT WORD 'TSTACK'
        PUBLIC  AMESTK
        DW      400 DUP(?)
AMESTK  LABEL   WORD                ;AMX86 EXECUTIVE STACK
AMXESTK ENDS
;
AMXISTK SEGMENT WORD 'MSTACK'
        DW      450 DUP(?)
AMISTK  LABEL   WORD                ;AMX86 INTERRUPT STACK
AMXISTK ENDS
;
AMXTSTK SEGMENT WORD 'MSTACK'
        DW      400 DUP(?)
SPTMR   LABEL   WORD                ;AMX86 TIMER TASK STACK
AMXTSTK ENDS
;
;AMX86 LARGE TASK STACK SEGMENTS
;
RXMGTSTACK SEGMENT WORD 'TSTACK'
        DW      400 DUP(?)
SPRXMG  LABEL   WORD                ;STACK FOR TASK # 1
RXMGTSTACK ENDS
;
TXMGTSTACK SEGMENT WORD 'TSTACK'
```

```
        DW      400 DUP(?)
SPTXMG  LABEL   WORD                ;STACK FOR TASK # 2
TXMGTSTACK ENDS
;
COMMTSTACK SEGMENT WORD 'TSTACK'
        DW      500 DUP(?)
SPCOMM  LABEL   WORD                ;STACK FOR TASK # 3
COMMTSTACK ENDS
;
SEWTSTACK SEGMENT WORD 'TSTACK'
        DW      500 DUP(?)
SPSEW   LABEL   WORD                ;STACK FOR TASK # 4
SEWTSTACK ENDS
;
MAKETSTACK SEGMENT WORD 'TSTACK'
        DW      500 DUP(?)
SPMAKE  LABEL   WORD                ;STACK FOR TASK # 5
MAKETSTACK ENDS
;
CONTTSTACK SEGMENT WORD 'TSTACK'
        DW      500 DUP(?)
SPCONT  LABEL   WORD                ;STACK FOR TASK # 6
CONTTSTACK ENDS
;
POSTTSTACK SEGMENT WORD 'TSTACK'
        DW      500 DUP(?)
SPPOST  LABEL   WORD                ;STACK FOR TASK # 7
POSTTSTACK ENDS
;
PRNTTSTACK SEGMENT WORD 'TSTACK'
        DW      400 DUP(?)
SPPRNT  LABEL   WORD                ;STACK FOR TASK # 8
PRNTTSTACK ENDS
;
        PAGE
;AMX86 RESOURCE CONTROL TABLE
;
AMRMDATA SEGMENT WORD 'DATA'
;
        PUBLIC  AMRCT
;
        EVEN
AMRCT   DW      1 DUP(?)            ;ALLOCATE STORAGE
;
AMRMDATA ENDS
;
        PAGE
;BUFFER POOL STORAGE AREAS
;
AABMDATA SEGMENT WORD 'DATA'
;
RAM0    DB      30806 DUP(?)        ;RAM FOR POOL # 0
RAM1    DB      1646 DUP(?)         ;RAM FOR POOL # 1
;
AABMDATA ENDS
;
        END
```

A.4. Header File for C Language Modules

The code for the IBM AT was divided up into several C language modules and one assembly language module. The following header file was included in all the C language modules :-

```
#include        "stdio.h"
#include        "dos.h"
#include        "math.h"
#include        "limits.h"

#define VERSION    2.3
#define U8259     0x20      /* 8259 Interrupt controller port */
#define UEOI      0x20      /* end-of-interrupt command */
#define U8259M    0x21      /* 8259 interrupt mask register */
#define UIRQ3M    0x08      /* IRQ 3 mask (serial comm. port #2 */
#define UIRQ4M    0x10      /* IRQ 4 mask (serial comm. port #1 */
#define UIRQ5M    0x20      /* IRQ 5 mask (GPC interrupts)      */
#define UCLK      0x40      /* clock port (timer 0 on 8253 CTC */
#define UCLKC     0x43      /* 8253 clock control */
#define UCLKV     8         /* clock interrupt type */
#define UCOMV     12        /* communicat. port #1 interrupt type*/
#define UGPCAV    13        /* General Purpose Communication int */
#define UGPCBV    11        /* General Purpose Communication int */
#define ONESEC    18        /* no. of AMX86 ticks in one sec */

#define UKBD      0x60      /* keyboard data port */
#define UKBDC     0x61      /* keyboard control port */
#define UKBDR     0x80      /* keyboard reset command */
#define UKBDV     9         /* keyboard interrupt type */

#define TNTMR     0         /* AMX Timer Task Number */
#define TNRXMG    1         /* Receive Message Task */
#define TNTXMG    2         /* Transmit Message task */
#define TNCOMM    3         /* Communication Supervisor Task */
#define TNSEW     4         /* Adaptive robotic sewing Task */
#define TNMAKE    5         /* Task to make one sub-assembly */
#define TNCONT    6         /* FIGARO Controller Task */
#define TNPOST    7         /* Post Mortem Report Generator Task */
#define TNPRNT    8         /* Print messages task    */
#define TIMER1    0         /* Timer used for speed calc */

#define POOL1     0         /* Buffer pool for print messages */
#define POOL2     1         /* Buffer pool for txmit messages */
#define MAXLINE   81        /* max. no. characters on a line  */

#define PORT_A    0x304     /* definitions for I/O card ports */
#define PORT_B    0x305
#define PORT_C    0x306
#define SPEED_P   0x307     /* sewing m/c speed analogue signal */
#define PORT_E    0x308
#define PORT_F    0x309
#define PORT_G    0x30a
```

```
#define CB_IO_1    0x30b        /* control port for ports E, F & G  */
#define CB_IO_2    0x30f
#define CB_COUNTR  0x30e        /* contrl port for counters & latches*/
#define LO_COUNT   0x300        /* lo byte of counter */
#define HI_COUNT   0x301        /* hi byte of counter */
#define FING1      0x310                /* port address for finger #1*/

            /* General Purpose Communcation Channel Functions */
#define INIT_GP      1          /* initialize GP communications        */
#define TERM_GP      2          /* terminate GP communications          */
#define FINDCLOTH    3          /* request robot find cloth             */
#define CORNER       4          /* request robot find upper RH corner */
#define UPTO_NDLE    5          /* put cloth corner under needle       */
#define FAR_RH       6          /* find far RH corner                   */
#define MOVEBACK     7          /* request robot move back a distance */
#define ST_ALTER     8          /* request VAL II start up ALTER       */
#define END_ALTER    9          /* request VAL II terminate ALTER      */
#define RETREAT     10          /* robot retreats from ndle with cloth*/
#define WHERE       11          /* VAL II report robot position        */
#define PARAM1      12          /* input parameters  -   version 1      */
#define GO_START    13          /* request robot move to start positn */
#define ALIGN_F     14          /* request robot aligns finger          */
#define DROP        15          /* request robot drops onto cloth      */
#define PARAM2      16          /* input parameters  -   version 2      */
#define GO_NEAR     17          /* request robot move to near.start    */
#define STARTUP     18          /* request startup data        */
#define FINEADJ     19          /* fine adjustment function             */
#define ANGLEADJ    20          /* fine angular adjustment function    */
#define ROTATE90    21          /* rotate cloth by 90 degrees           */
#define INCHMOVE    22          /* inching motion function              */
#define REMOVE      23          /* remove robot from needle zone       */
#define STRAIGHTN   24          /* straighten out cloth        */
#define END_CLOTH   25          /* find end of cloth                    */
#define Q_AGAIN     26          /* ask whether to continue             */

               /*  Sewing Machine Functions */
#define SEW_START   0x01            /* mask for variable sewing speed */
#define SEW_STOP    0x00
#define TRIM_THREAD 0x02            /*           thread trimming        */
#define NEEDLE_UP   0x04            /*           needle up              */
#define SLO_SEW     0x10            /* sew at slow speed        */
#define FAST_SEW    0x20            /* sew at maximum speed             */
#define PRESSER_FT  0x40            /*           presser foot up        */
#define BACKTACK    0x80            /*           backtack               */

#define RESET_CNTR  0x01            /* mask to reset counters */
#define LATCH_EN    0x02            /*          enable latches */
#define PMBAK       100

#define TRUE           1
#define FALSE          0

#define ALTER       0x3f8                       /* serial port #1 */
#define LCR            3
#define I IR           2
#define LSR            5
#define DLL            0
#define DLM            1
```

```
#define IER            1
#define MSR            6
#define MCR            4
#define ALT_LSR        0x3fd
#define ALT_IIR        0x3fa
                                            /* max divisor is 0x0f (char)*/
#define HI_BAUD_RT     0x06                              /* 19200 baud */
#define LO_BAUD_RT     0x0c                              /*  9600 baud */

#define ETX            0203
#define DLE            0220
#define DEL            0377
#define STX            0202
#define SC_FACT          32                 /* scale factor for ALTER par*/
#define NSLOT          200                  /* no. slots in circ. list  */


                                            /* I-SIGHT camera card defns */
#define SEGMNT         0x9c00               /* camera card address segmnt*/
#define CONTRLB        0x3fff               /* control byte address      */
#define TRIGGER        0x00                 /* ctrl byte to trigger pict */
#define FREEZE         0x08                 /* freeze control byte       */
#define BUSFRZ         0x09                 /* mask for bus + freeze     */
#define CAM1_OFS       0x000                /* offset for camera # 1     */
#define CAM2_OFS       0x400                /* offset for camera # 2     */
#define CAM1_FL        0x3f1                /* address of flag of cam #1 */
#define CAM2_FL        0x3f3                /* address of flag of cam #2 */
#define NCAM           2                    /* mo. of cameras             */
#define NROW           30                   /* no. of rows of pixels     */
#define NCOL           32                   /* no. of columns of pixels  */
#define NPIXLS         ROW * NCOL           /* no. of pixels in picture  */

#define print_init       ajbgb(POOL1,&p.mp);p.outpt=5
#define prf__            p.n=sprintf(p.mp,
#define end_print        if(ajcall(TNPRNT,2,&p)<0)crash(1162)
#define displ_init       ajbgb(POOL1,&p.mp);p.outpt=2
#define gpf_start(a)     send_gp((char)a,TRUE)

                                  /* Structure Definitions */

typedef struct SPMESS {            /* print message struct definition */
        short int n;
        char *mp;
        char outpt;
        }PMESS;

typedef char SLOT1;                       /* 1 byte slots in circ.lists*/

typedef struct SCLIST {            /* circular list struct definition*/
        char header[8];
        SLOT1 slots[NSLOT];
        } CLIST;

                        /*  ROBOT specific parameters    */
#define TOANG     (float)284.477044          /* VAL II scaling factor */
#define RAD_TO_A  (float)57.29577951         /* rads to angles conversn */
#define ROT_FACT  (float)(-TOANG*RAD_TO_A)
                                              /* scales from radians to VAL*/
```

```
                    /*  GRIPPER specific parameters    */
#define RIGHT_MAX  251*SC_FACT          /* max modification dist in y*/
#define LEFT_MAX   160*SC_FACT          /* min modification dist in y*/
#define R_MAX      860*SC_FACT          /* max reach of robot        */
#define R_MIN      415*SC_FACT          /* min reach of robot         */
#define R_MID      680*SC_FACT
#define MAX_ANG     35*TOANG            /* limit to z_rot */
#define NX_MAX      80*SC_FACT          /* exclusion zone before ndle*/
#define NX_MIN    -150*SC_FACT          /* exclusion zone after ndle */
#define NY_MAX     120*SC_FACT          /* exclusion zone beside ndle*/

#define STITCH_LEN       3               /* stitch length in mm */
#define TRK_FACT         1               /* tracking proportional gain*/
#define TOP_SPEED        255             /* sewing speed ratio to 256 */
#define MID_SPEED        170             /* sewing speed ratio to 256 */
#define SLO_SPEED        50

#define Y1_PIXEL  (float)(0.43*SC_FACT)         /* cam1, pixel width  */
#define Y2_PIXEL  (float)(0.67*SC_FACT)         /* cam2, pixel width  */
#define SEAM_W     (float)(12*SC_FACT)          /* nominal seam width */
#define CAM2_DIST  (float)(30*SC_FACT)          /* dist Xcam          */
#define F_TO_PC         135*SC_FACT             /* fing to pcell dist */

#define NEAR    TRUE                     /* near technique to be used */
#define FAR     FALSE                    /* far  technique to be used */

        /*   Referencing  all the functions so that debugging information
      is provided by the compiler.                                   */

extern void     main(), rtrack(), rpsew(), gpb_isp(), gpa_isp(),
                stseam(), stsew(), gp_function(int), gpf_end(int),
                read_offset(), set_param(), angle_adj(),
                send_word(int), send_gp(char,int), count_reset(),
                ndle_down(), e_calc(float *, float *), stpost(),
                pr_runtime(), setup_pixels(), stcomm(), inch(),
                set_speed(int), delay(int), rpcamr(), rpiptr(),
                install(int,int,int), take_picture(), read_cam(),
                z80_check(), stprnt(int,char *,char), crash(int),
                pr_alt_st(int), rpcom(), init(int,char), rtimer(),
                strxmg(), sttxmg(), stack(char **), sh_delay(),
                norm_msg(char **), pm(int), pr_heading(), comisp(),
                clkisp(), tx_byte(char), initialise(), adjust(int),
                startup_data(), where(), set1_param(), set2_param(),
                fine_adj(), std_msgs(), CalcSeamSection();
extern char     get_byte(int);
extern int      get_word(), speed_control(int), DecideSeam(int *),
                tension(), limit(int,int), limit2(int,int),
                edge_find(char *,int), find_edge(char *,int),
                intrprt(char), read_count(),limit3(int,int);

extern int      tens_corr(int,int,long *,double),
                x_corr(unsigned int *, double),
                y_corr(int *,int *,int *),
                envelope(int,int,double,int *);
extern float    transf_fn(), StdDev(double,double,int);
extern double   rcos(double);
```

```
     /*  Referencing  global variables,  to make them  accessible  in
    all the modules.  These variables are declared and described
    in module A.                                                */

extern int      sewwait,  GPInWait,  GPOutWait,  completed, ifeed,
                i_hand,     x_total,     y_total,     SeamSection,
                StopDistance,  sp_len,  x_0,  y_0,  max_e,  min_e,
                max_t,  min_t,  flip, i_t_Avg, rq_tens, accel_lim,
                vel_lim,  irow1,  irow2,  ipix1_ofst,  ipix2_ofst,
                in_nbyte,  terminate,  rxwait,  no_int,  newtxpt,
                comwait,  pmarray[],  *pstart,  *pfinish,  *pbuf,
                fing_dist,  f_r,  n_x,  n_y,  cloth_end, acc_dist,
                calc_dist, decel_dist, debug, sew_near, caller;

extern char     b_port,   msg_in[],   *pt_txmg,  *new_pt,  *cc_pt,
                *cccb_pt,  *tp1_pt,  *tp2_pt,  *cam1_pt, *cam2_pt,
                cam1_buf[],      cam2_buf[],      *StartAckMsg_pt,
                *NullMsg_pt;
extern long int  t_MeanDev, t_Avg, z_total;
extern unsigned  int t_period, offst1, count1, count2;
extern float    pixel1[],     pixel2[],     gain_pix[],     pmdat[],
                *pmdata_pt, blp_fact, e_MeanDev, e_Avg, pix1_ofst,
                pix2_ofst, t_gain, int_fact, deriv_gain, pix_gain,
                th_0, f_angle, cos_f, sin_f, s_gain;
```

A.5. Global Variables

All the global variables used in the C language modules, were defined in the first module.

```
                         /*   Global variables    */

        /* flags */
int sewwait;                  /* SEW task waiting for handshake */
int GPInWait;                 /* waiting for GPC IBF interrupt  */
int GPOutWait;                /* waiting for GPC OBF interrupt  */
int completed;                /* cloth length has been sewn up  */
int terminate;                /* flag to teminate COMM Task     */
int rxwait;                   /* RXMG task waiting for COMISP ? */
int newtxpt;                  /* SEW task updated transmit msg? */
int comwait;                  /* COMM task waiting for RXMG ?   */
int cloth_end;                /* end of cloth detected ?        */
int sew_near;                 /* sew section using near technique */

       /* ALTER communication Parameters */
CLIST rxlist,txlist, chlist;
char   msg_in[260];
char *pt_txmg;                               /* pointer to txmit msg */
char *new_pt;                        /* pointer to updated txmit msg */
char *StartAckMsg_pt, *NullMsg_pt;
                                 /* pointers to standard txmit msgs */
int in_nbyte, no_int;
int ifeed, i_hand, x_total, y_total;
```

```
long int z_total;

        /* GPC channel Parameters */
char b_port;                          /* initial contents of PORT_B */
int caller;                           /* Task No. of calling Task */

        /* Post_mortem and crash parameters */
int pmarray[PMBAK];
int *pstart, *pfinish, *pbuf;
float pmdat[4000];
float *pmdata_pt = &pmdat[0];
int debug = FALSE;

        /* camera parameters */
float  pix1_ofst;
float  pix2_ofst;
float pixel1[NCOL+1], pixel2[NCOL+1], gain_pix[NCOL+2];
char *cc_pt, *cccb_pt, *tp1_pt, *tp2_pt, *cam1_pt, *cam2_pt;
char cam1_buf[NPIXLS+2];
char cam2_buf[NPIXLS+2];


        /* Robot startup data Parameteras */
int fing_dist;                              /* dist between two fingers */
int f_r;                                       /* finger-flange radius */
float f_angle, cos_f, sin_f;                    /* finger-flange angle */
int n_x;            /* needle position w.r.t. robot base, x coord */
int n_y;            /* needle position w.r.t. robot base, y coord */

        /* Sewing Task parameters */
int SeamSection;                /* length of seam section to be sewn  */
int StopDistance;               /* dist of seam section end to needle */
unsigned int offst1;            /* finger ADC's offsets at zero load */
int x_0;                          /* initial 1st finger x position */
int y_0;                          /* initial 1st finger y position */
float th_0;                       /* initial theta, 2nd finger angle */
float blp_fact;                   /* converts blips to y displcmnt */
float e_MeanDev;                  /* seam error mean deviation */
float e_Avg;                      /* seam error average */
long int t_MeanDev;               /* tension error mean deviation */
long int t_Avg;                   /* tension error average */
int max_e, min_e, max_t, min_t, i_t_Avg;
int flip;
int acc_dist, calc_dist, decel_dist;

        /* Parameters for calclulating sewing speed */
unsigned int count1, count2;
unsigned int t_period;
int sp_len;

        /* Parameters that are reset by set_param() */
float t_gain;                      /* Tension servo, proportnl gain */
float int_fact;                    /* tension servo, integral gain */
float deriv_gain;                  /* Seam servo, derivative gain   */
float s_gain;                      /* Seam servo, proportnl gain */
float pix_gain;                    /* proportnl gain per pixel */
int rq_tens;                       /* demand tension */
int accel_lim;                     /* acceleration limitation */
```

```
int vel_lim;                     /* velocity limitation */
int irow1;                       /* pixel row no for 1st camera */
int irow2;
int ipix1_ofst;                  /* camera 1 centreline offset */
int ipix2_ofst;                  /* camera 2 centreline offset */
```

A.6. Initialisations

A.6.1. Restart Procedures

Restart Procedures were written for the communication Tasks, the vision system and for the SEW Task, and they are listed in Appendices B, F and D respectively. A simple Restart Procedure for the AMX timer was also required, as follows :-

```
void rtimer()
{       ajmodl();

        ajbia();                 /* initialize all buffer pools */
        rpiptr();                /* set up pointers to ISP's */
}
```

A.6.2. AMX Start Up

The AMX executive was started using the following start-up code :-

```
void _main()                      /* replace Microsoft's _main() */
{       main();
}

void main()
{    extern unsigned int _top;
     int i,j;

     _top = 0xFFF0;                     /* disable stack checking */
                                     /* delay until disk motor off */
     for (j =30; j != 0; j--)
        for (i = 8000; i != 0; i--)
            ;
     amxgo();                                    /* start AMX  */
}
```

A.7. PRNT Task

Messages were displayed on the screen or printed out via the PRNT Task. The Task was given the lowest priority so that higher priority Tasks were not blocked by the printing out process.

```
void stprnt(n,mp,outpt)
int n;                              /* no. of characters in string */
char *mp;                           /* pointer to string */
char outpt;                         /* display or print code */
{
    char *msgp;
    ajmodl();

    for ( msgp = mp; msgp < mp+n+1 ; )
        bdos(outpt,*msgp++);            /* Lattice library function */
    bdos(outpt,0x0a);                 /* carriage return & new line */
    bdos(outpt,0x0d);

    if (ajbrb(mp) != 0)                 /* release  message buffer */
        crash(8086);
}
```

A.8. Miscellaneous Functions

Extensive debugging facilities were developed and incorporated into the code. The crash() function provided a simple error message facility. The pm() function provided a post-mortem facility in which values could be stored during a real time process and printed out afterwards.

Two time delay functions were written, a normal delay() and a short sh_delay().

```
void crash(code)
int code;
{
    char *stp, *msgp, stbuf[120];
    int n;
    PMESS p;

    install(0,0,0);                                        /* stop robot */
    terminate = TRUE;                                   /* stop COMM task */
    ajoutb(PORT_A,0);                                  /* stop sewing m/c */
    stp = &stbuf[0];
    n = sprintf(stp," CRASH detected, crash code = %d", code);
    for (msgp = stp; msgp < stp+n+1;)
        bdos(2,*msgp++);
    bdos(2,0x0a);
    bdos(2,0x0d);
    if (debug) pr_runtime()  ;
}

                /* This routine instals a post-mortem code */
                /* into a buffer for debugging purposes    */
void pm(code)
int code;
{
        *(pbuf++) = code;
```

```
        if (pbuf > pfinish)
                pbuf = pstart;
}


void delay(times)
int times;
{
        int i,j;

        for (i=0; i < times; i++)
                for (j=0; j < 500; j++)
                        ;
}

void sh_delay()
{
        int i;
        for( i=0; i < 10; i++)
                ;
}
```

APPENDIX B

SOFTWARE FOR ALTER COMMUNICATION CHANNEL

B.1. The Restart Procedure

```
void rpcom()                  /* restart procedure for comm. port */
{   PMESS p;
    char m_reg;
    ajmodl();
    displ_init;
    prf__ "restart procedure for ALTER communications task");
    end_print;

    ajdi();
    m_reg = ajinb(U8259M);            /* enable IRQ4 interrupt */
    m_reg = m_reg & ~UIRQ4M;               /* reset IRQ4 mask */
    sh_delay();
    ajoutb(U8259M, m_reg);
    init(ALTER,(char)HI_BAUD_RT);     /* initialize comm. chip */
    ajei();
                                       /* init. circ. lists */
    ajrstl (&rxlist,sizeof(SLOT1),NSLOT);
    ajrstl (&txlist,sizeof(SLOT1),NSLOT);
                                  /* init. post-mortem pointers*/
    pstart = &pmarray[0];
    pfinish = &pmarray[PMBAK-1];
    pbuf = &pmarray[0];
}

               /* This routine sets up the serial port */
void init(port,baud)
int port;
char baud;
{  char byte;
   ajmodl();

   ajoutb(port+IER,0);          /* disable all IER interrupts*/
   byte = ajinb(port+LSR);      /* clear Rx error interrupt */
   byte = ajinb(port);          /* clear Rx data interrupt */
   byte = ajinb(port+IIR);      /* clear Tx interrupt  */
   byte = ajinb(port+MSR);      /* clear modem interrupt */
   ajoutb(port+LCR,0);
   ajoutb(port+MCR,0);
   ajoutb(port+LCR,0x80);       /* set DLAB to access baud */
   ajoutb(port+DLL,baud);       /* set baud rate divisor   */
   ajoutb(port+DLM,0x00);
   ajoutb(port+MSR,0x00);
   ajoutb(port+LCR,0x03);
   ajoutb(port+MCR,0x08);     /* OUT2 must be high for interrpt */
   ajoutb(port+IER,0x07);
}
```

B.2. The COMM Task

```
          /* Communication task - supervises handshaking */
void stcomm()
{
    PMESS p;
    int alt_stat;

    ajmodl();
    displ_init;
    prf__ "communication task started");
    end_print;
               /* initialise Global variables */
    terminate = FALSE;
    rxwait = newtxpt = comwait = FALSE;
    i_hand = 0;
    alt_stat = 5;                        /* ALTER not up yet */
    std_msgs();

               /* infinite loop for handshaking cycle */
    for (i_hand = 0;        ;i_hand++)
    {
        if (ajtask(TNRXMG) != 0)         /* start RXMG Task */
            crash(9080);
        ajshed();
        ajdi();
        comwait = TRUE;
        ajwait();             /* wait until ALTER sends a msg */

        switch(intrprt(msg_in[0]))          /* interpret msg */
        {
           case 0 :                         /* ALTER starting */
                pt_txmg = StartAckMsg_pt;
                if (ajtask(TNTXMG) != 0) crash(9081);
                ajshed();
                pt_txmg = NullMsg_pt;
                break;

            case 1 :                         /* ALTER running */
                if (newtxpt)      /* check if new msg ready ? */
                {    ajdi();
                     newtxpt = FALSE;
                     pt_txmg = new_pt;    /* instal new pointer*/
                     ajei();
                }
                ajtask(TNTXMG);           /* call TXMG Task */
                ajshed();
                break;

           case  2 :                     /* ALTER terminating */
           case 3 :
           case 4 :
                pr_alt_st(alt_stat);
                ajend();
        }
        if (terminate) ajend();
    }
}
```

```
     /* This routine sets up the Standard ALTER messages */
void std_msgs()
{   if (ajbrb(StartAckMsg_pt) < 0);      /* release old buffers */
    if (ajbrb(NullMsg_pt) < 0);

    if (ajbgb(POOL2,&StartAckMsg_pt) != 0)
        crash(6437);
    *StartAckMsg_pt = 1;               /* start acknowledge msg */
    *(StartAckMsg_pt+1) = 0;

    if (ajbgb(POOL2,&NullMsg_pt) != 0)
        crash(6436);

    *NullMsg_pt = 2;                   /* normal acknowledge msg */
    *(NullMsg_pt+1) = 0;
    *(NullMsg_pt+2) = 0;
}

  /* this routine prints out the status of the ALTER comms */
void pr_alt_st(alt_stat)
int alt_stat;
{
   PMESS p;

   switch (alt_stat)
   {
        case 0:
                displ_init;
                prf__ " ALTER starting");
                end_print;
                break;
        case 1:
                displ_init;
                prf__ " ALTER running");
                end_print;
                break;
        case 2:
                displ_init;
                prf__ " ALTER pausing");
                end_print;
                break;
        case 3:
                displ_init;
                prf__ " ALTER terminated");
                end_print;
                break;
        case 4:
                displ_init;
                prf__ " error detected by VAL II");
                end_print;
   }
}

      /* This routine interprets VAL II's ALTER control byte */
intrprt(contrlb)
char contrlb;
{  PMESS p;
```

```
   if ((char)(contrlb & 0x07) == 0)
   {     switch ((int)( (char)(contrlb & 0x60) ))
         {   case  0    : return(1);         /* ALTER running */
             case  0x20 : return(0);         /* ALTER starting */
             case  0x40 : return(2);         /* ALTER pausing  */
             case  0x60 : return(3);         /* ALTER stopping */
          }
   }
   displ_init;
   switch ((int)( (char)(contrlb & 0x07) ))
   {
   case 1:
        prf__ " checksum error detected by VAL");
        break;
   case 2:
        prf__ " framing/format error detected by VAL");
        break;
   case 3:
        prf__ " data overrun detected by VAL");
        break;
   case 4:
        prf__ " too many messages complaint from VAL ");
        break;
   case 5:
        prf__ " protocol error detected by VAL");
        break;
   case 6:
        prf__ " timeout error detected by VAL");
        break;
   default :
        prf__ " undefined VAL error message");
   }
   end_print;
   return(4);
}
```

B.3. The RXMG Task

```
void strxmg()                                  /* Rx message Task */
{
    char  in_msg, dle_det,end_det,start_det, byte, checksum;
    void rx_halt();
    ajmodl();

    if (sewwait)                            /* wake up SEW Task before*/
    {   sewwait = FALSE;                     /* RXMG suspends itself */
        if (ajwake(TNSEW) != 0)
            rx_halt();
    }
    ajdi();                             /* ensure wait state before   */
    rxwait = TRUE;                                     /* setting flag */
    ajwait();                           /* wait for COMISP interrpt   */

    if ((ajrbl(&rxlist,&byte)) < 0)             /* take 1st byte */
```

```
   rx_halt();                          /* off list */
msg_in[0] = byte;

in_nbyte = 1;              /* initialize flags & counters */
in_msg = TRUE;
checksum = 0;
dle_det = FALSE;
end_det = FALSE;
start_det = FALSE;

while (in_msg)
{                               /* check for error conditions */
    if ((in_nbyte > 6 && !start_det)  !!  (in_nbyte > 254))
            rx_halt();

    ajdi();
                                /* remove next byte from list */
    if ( (ajrbl(&rxlist,&byte)) < 0)
    {                                 /* if list empty - wait */
       ajdi();                          /* ensure wait state */
       rxwait = TRUE;                 /* before setting flag */
       ajwait();
       if ( (ajrbl(&rxlist,&byte)) < 0)      /* try again */
            rx_halt();
    }
    ajei();
    msg_in[in_nbyte++] = byte;

    if ( end_det )                           /* end of msg */
    {
            if ( (checksum += byte)  !=  0 )
                    rx_halt();
            in_msg = FALSE;
            in_nbyte = in_nbyte - 3;
    }

    else if (dle_det)
    {
            switch ((int)byte)
            {  case  ETX :  end_det = TRUE;
                            break;
               case  STX :  start_det = TRUE;
                            dle_det = FALSE;
                            in_nbyte = 0;
                            break;
               case  DLE :  in_nbyte -= 1;
                            dle_det = FALSE;
                            checksum += byte;
                            break;
               default   :  rx_halt();
            }
    }

    else if (byte == DLE)
            dle_det = TRUE;

    else if (start_det)
            checksum += byte;
```

```
    }
    if (comwait)
    {
        comwait = FALSE;
        ajwake(TNCOMM);
    }
    else
        rx_halt();                  /* COMM should have been waiting */
    ajend();
}

void rx_halt()
{
        PMESS p;

        displ_init;
        prf__ " error in incoming ALTER message packet");
        end_print;
        crash(7);
}
```

B.4. The TXMG Task

The tx_byte() routine, which transmits a single byte down the ALTER channel, was written in Assembler, and is listed in section B.5.

```
void sttxmg()                           /* Transmit message task */
{
    int nbyte, temp;
    char *pt, checksum;

    ajmodl();

    checksum = 0;
    nbyte =  *pt_txmg;

    /* Check that TxHR on Comm Chip is empty before starting */
    if( !(ajinb(ALT_LSR) & 0x20) )
        crash(66);
    tx_byte((char)DEL);
    tx_byte((char)DLE);
    tx_byte((char)STX);

    for ( pt = pt_txmg + 1; pt  < pt_txmg + nbyte + 1; pt++)
    {
        tx_byte(*pt);
        checksum += *pt;
        if (*pt == (char)DLE)
                tx_byte(*pt);
    }

    tx_byte((char)DLE);
    tx_byte((char)ETX);
```

```
    checksum = (~checksum) + 1;
    tx_byte(checksum);
                  /* accumulate ALTER data in global variables */
    if (nbyte > 3)
    {
      x_total += *(pt_txmg+3) + (*(pt_txmg+4) << 8);
      y_total += *(pt_txmg+5) + (*(pt_txmg+6) << 8);
      temp = *(pt_txmg+7) + (*(pt_txmg+8) << 8);
      z_total += (long)temp;
    }
    ajend();
}
```

B.5. Assembly Module

```
;
        PAGE    60,132
;
;       *********************************************************
;       *                                                       *
;       *   MODULE B  -  ASSEMBLER ROUTINES FOR ROBOTIC SEWING  *
;       *                DEVELOPMENT PROJECT                    *
;       *                                                       *
;       *********************************************************
;
;
;       CONSTANTS DEFINITIONS
;                  (for meanings see header file to C routines)
;
TNRXMG  EQU     1
TNCOMM  EQU     3
ALTER   EQU     03F8H
ALT_IIR EQU     03FAH
ALT_LSR EQU     03FDH
U8259   EQU     20H
UEOI    EQU     20H
UCLKV   EQU     8
UCOMV   EQU     12
ETX     EQU     0203Q
STX     EQU     0202Q
DLE     EQU     0220Q
;
;       AMX86 EXTERNAL DECLARATIONS
;
        EXTRN   AARBL:FAR
        EXTRN   AAATL:FAR
        EXTRN   AAWAIT:FAR
        EXTRN   AACLK:FAR
        EXTRN   AAEND:FAR
        EXTRN   AAINT:FAR
        EXTRN   AAINX:FAR
        EXTRN   AJMODL:FAR
        EXTRN   AAIPTR:FAR
        EXTRN   CRASH:FAR
        EXTRN   AAWAKE:FAR
```

```
;
DGROUP  GROUP   DATA
DATA    SEGMENT WORD  PUBLIC  'DATA'
        ASSUME  DS:DATA
;
        EXTRN   RXLIST:BYTE
        EXTRN   TXLIST:BYTE
        EXTRN   RXWAIT:WORD
        EXTRN   MSG_IN:BYTE
        EXTRN   IN_NBYTE:WORD
        EXTRN   COMWAIT:WORD
;
DATA    ENDS
;
        PAGE
;;
SUPCODE SEGMENT BYTE  'CODE'
        ASSUME CS:SUPCODE
;
        PUBLIC RPIPTR
        PUBLIC COMISP
        PUBLIC CLKISP
        PUBLIC TX_BYTE
;
;;
;       ************************************************
;       *                                              *
;       *       RPIPTR  -  RESTART PROCEDURE TO INSTAL *
;       *                  INTERRUPT POINTERS          *
;       *                                              *
;       ************************************************
;
;
RPIPTR  PROC FAR
;
;       call    AJMODL
        mov     ax,SUPCODE
        mov     es,ax                   ; es = current segment
        mov     bx, OFFSET CLKISP       ; es:bx = address(CLKISP)
        mov     dl, UCLKV               ; dl = clock intrpt type
        call    AAIPTR
        nop
        nop
        nop
        nop
        mov     bx, OFFSET COMISP       ; es:bx = address(COMISP)
        mov     dl, UCOMV               ; dl = port intrpt type
        call    AAIPTR
        nop
        nop
        nop
        nop
        ret
;
RPIPTR  ENDP
;
        PAGE
;
```

```
;       ************************************************
;       *                                              *
;       *       CLOCK INTERRUPT SERVICE PROCEDURE      *
;       *                                              *
;       ************************************************
;
CLKISP  PROC FAR
;
;       call    AAINT                       ; inform AMX
        push    ax
        mov     al,UEOI
        out     U8259,al                    ; end-of-interrupt signal
        pop     ax
        call    AACLK                       ; go to AMX86 clock ISP
;       call    AAINX
        iret                                ; dismiss interrupt
;
CLKISP  ENDP
;
        PAGE
;
;       ***********************************************************
;       *                                                         *
;       *       COMMUNICATIONS INTERRUPT SERVICE PROCEDURE        *
;       *               (ALTER COMMUNICATIONS CHANNEL)            *
;       ***********************************************************
;
;
COMISP  PROC FAR
;
        call    AAINT           ; tell AMX86 about interrupt
        call    AJMODL          ; set data segment
        push    es
        mov     ax,ds
        mov     es,ax
;
TOP:    mov     dx,ALT_IIR
        in      al,dx     ; read in Interrupt Identification Reg
        test    al,01     ; while (!((ajinb(ALTER_IIR)) & 0x01))
        jnz     FININT    ; jump if no interrupt left
;
        cmp     al,04           ; IIR = 4 - byte has been receivd
        jz      RECEIV
        cmp     al,02         ; IIR = 2 - Transmit Hold Reg Empty
        jz      TXMIT
        cmp     al,06           ; IIR = 6 - framing error
        jz      FRAME
;
        jmp     TOP       ; return to check for another interrupt
;
RECEIV: mov     dx,ALTER                      ; read in byte
        in      al,dx
        mov     bx, OFFSET DGROUP:RXLIST         ; address of list
        mov     cl,al                         ; byte to add to list
        call    AAATL               ; add byte to top of circ list
;
        test    ax,ax           ; test for successful call to AAATL
        jns     CONT1
```

```
        mov     ax,0001
        push    ax
        call    CRASH          ; crash(1)  if failure to add to list
        mov     sp,bp
;
CONT1:  cmp     [RXWAIT],0000                    ; if (rxwait) then
        jz      TOP
;
        mov     [RXWAIT],0000                    ; rxwait = FALSE
        mov     dx,TNRXMG
        call    AAWAKE                           ; wake up RXMG task
        test    ax,ax
        jz      TOP
;
        mov     ax,0003
        push    ax
        call    CRASH                   ; crash(3)  if fail to wake
        mov     sp,bp                   ; RXMG when rxwait = TRUE
;
        jmp     TOP
;
TXMIT:  mov     bx,OFFSET DGROUP:TXLIST
        call    AARBL          ; remove byte from TXLIST circ list
        test    ax,ax
        js      TOP                    ; no byte on list, do nothing
;
        mov     al,cl
        mov     dx,ALTER
        out     dx,al                   ; transmit byte
        jmp     TOP
;
FRAME:  mov     dx, ALT_LSR
        in      al,dx                   ; read LSR to dismiss intrpt
        xor     ax,ax
        push    ax
        call    CRASH                   ; crash(0)  if framing error
        mov     sp,bp
        jmp     TOP
;
FININT: pop     es
        mov     ax,UEOI
        out     U8259,al                   ; dismiss interrupt signal
        call    AAINX                      ; return via AMX86
        iret
;
COMISP  ENDP
;
        PAGE
;
;
;       ********************************************************
;       *                                                      *
;       *       TX_BYTE  -  SUBROUTINE TO TRANSMIT A BYTE DOWN *
;       *                   ALTER COMMUNICATION CHANNEL        *
;       *                                                      *
;       ********************************************************
;
TX_BYTE PROC FAR
```

```
;
        call    AJMODL          ; set data segment
        push    bp
        mov     bp,sp
        mov     al, [bp+6]      ; load parameter -  byte
        mov     cl,al
        pop     bp
;
        push    es
        mov     ax,ds
        mov     es,ax
;
        cli
        mov     bx, OFFSET DGROUP:TXLIST
        call    AAATL           ; add byte to circ. list
;
        cli
        test    ax,ax         ; test for successful call to AAATL
        jns     CONT3
        mov     ax,0009
        push    ax
        call    CRASH         ; crash(9)  if can't add byte to list
;
CONT3:  mov     dx,ALT_LSR
        in      al,dx           ; read in Line Status Register
        test    al,20H          ; if Tx Hold Reg is not empty
        jz      FINTXB          ; leave byte on circ. list
;
        mov     bx, OFFSET DGROUP:TXLIST
        call    AARBL
        test    ax,ax
        jns     CONT4
        mov     ax,0008
        push    ax
        call    CRASH           ;  crash(8)  if no byte on list
;
CONT4:  mov     al,cl
        mov     dx,ALTER
        out     dx,al           ; transmit byte to comm port
;
FINTXB: sti                     ; enable interrupts
        pop     es
        ret
;
TX_BYTE ENDP
;
;
;
SUPCODE ENDS
;
;
        END
```

B.6. High Level Interface

High level Tasks, such as the SEW Task, conveyed ALTER data to the COMM Task using the following instal() routine.

```
void install(x_displ,y_displ,z_rot)
int x_displ,y_displ,z_rot;
{
        char *pt;

        if(ajbgb(POOL2,&pt) != 0)
                crash(876);

        y_displ = -y_displ;

        *pt = 8;
        *(pt+1) = 0;
        *(pt+2) = 0x31;
        *(pt+3) = (char)x_displ;
        *(pt+4) = (char)(x_displ >> 8);
        *(pt+5) = (char)y_displ;
        *(pt+6) = (char)(y_displ >> 8);
        *(pt+7) = (char)z_rot;
        *(pt+8) = (char)(z_rot >> 8);

        if (ajbrb(new_pt) < 0)
                ;
        new_pt = pt;
        newtxpt = TRUE;
}
```

APPENDIX C

THE GPC LINK

C.1. Software Support for GP Communications

C.1.1. IBM AT Implementation

C.1.1.1. Interrupt Service Procedures

```
                    /* Interrupt Service Procedure for GPC O/P  */
void gpa_isp()
{
        if (GPOutWait)
        {       GPOutWait = FALSE;
                if (ajwake(caller) != 0)
                        crash(2322);
        }
        ajoutb(U8259, UEOI);
}

                    /* Interrupt Service Procedure for GPC I/P  */
void gpb_isp()
{
        if (GPInWait)
        {       GPInWait = FALSE;
                if (ajwake(caller) != 0)
                        crash(2322);
        }
        if (!cloth_end)
        {       cloth_end = TRUE;
                ajinb(PORT_F);          /* dismiss interrupt */
        }
        ajoutb(U8259, UEOI);
}
```

C.1.1.2. I/O Routines

```
char get_byte(control)
int control;
{       PMESS p;
        char temp_b, Ok;

        Ok = FALSE;
        do
        {                               /* wait until INT clear */
                if( !(ajinb(PORT_G) & 0x01) )
                {       ajdi();
                        caller = ajgetn();
                        GPInWait = TRUE;
```

```
                        ajwait();
                }
                temp_b = ajinb(PORT_F);
                if (control && !(ajinb(PORT_G) & 0x10))
                {
                        displ_init;
                        prf__ "Unexpected data byte = %5d",temp_b);
                        end_print;
                }
                else
                        Ok = TRUE;
        } while (!Ok) ;
        return(temp_b);
}

get_word()
{
        int temp;

        temp = get_byte(FALSE);
        return( temp  + ((int)get_byte(FALSE) << 8) );
}

void send_gp(bite,control)
char bite;
int control;
{
        if ( !(ajinb(PORT_G) & 0x08))        /* wait for OBF clear */
        {       ajdi();
                caller = ajgetn();
                GPOutWait = TRUE;
                ajwait();
        }
        if (control)                          /* set CONTROL high */
                ajoutb(PORT_B,b_port |= 0x80);
        else                                   /* set Control low */
                ajoutb(PORT_B,b_port &= 0x7F);

        ajoutb(PORT_E,bite);                          /* output byte */
        sh_delay();
        ajoutb(PORT_E,bite);              /* repeat for good luck */
}

void send_word(word)
int word;
{
        send_gp((char)word,FALSE);
        sh_delay();
        sh_delay();
        send_gp((char)(word >> 8),FALSE);
}
```

C.2. VAL II Implementation of GPC

```
PROGRAM inword
        tmpbyte = byte
        CALL inbyte
        PC 2016, 8 = byte    ; read low byte into register
        FOR ii = 1 TO 30
        END
        CALL inbyte
        PC 2024, 8 = byte   ; read high byte into register
        word = BITS(2016, 16)       ; recompose word
        byte = tmpbyte              ; restore function code
END

PROGRAM inbyte
        WAIT SIG(-1008)     ; check OUTPUT BUFFER FULL signal
        byte = BITS(1009, 8)    ; read in data byte from bus
        IF SIG(1006) THEN       ; check CONTROL line
            incontrol = TRUE
        ELSE
            incontrol = FALSE
        END
        SIGNAL 8                ; toggle ACKNOWLEDGE line
        FOR ii = 1 TO 20        ; delay
        END
        SIGNAL -8
        FOR ii = 1 TO 20        ; delay for 8255 to respond
        END
        RETURN
END

PROGRAM outbyte
        WAIT SIG(-1007)      ; check INPUT BUFFER FULL line
        IF contout THEN
            SIGNAL -6
        ELSE
            SIGNAL 6
        END
        PC 9, 8 = COM byte       ; put data byte on bus
        SIGNAL 7                 ; toggle STROBE? line on
        FOR i = 1 TO 2           ; short delay
        END
        SIGNAL -7                ; toggle STROBE line off
        RETURN
END

PROGRAM outword
        tmpbyte = byte           ; store function code
        contout = FALSE      ; reset flag to send data byte
        PC 2016, 16 = word
        byte = BITS(2016, 8)
        CALL outbyte; send low byte
        FOR i = 1 TO 20
        END
        byte = BITS(2024, 8)
        CALL outbyte             ; send high byte
        byte = tmpbyte           ; restore function code
```

```
  11        contout = TRUE            ; set control flag
END
```

C.3. Calling VAL II Functions

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

C.3.1. IBM AT Implementation

The following routines were used to call a VAL II function, from any Task :-

```
void gp_function(code)
int code;
{       gpf_start(code);
        gpf_end(code);
}

void gpf_end(code)
int code;
{    PMESS p;
  char temp;

  temp = get_byte(TRUE);
  displ_init;
  switch ((int)temp)
  {
   case 0        : prf__ "VAL II has aborted");                break;
   case INIT_GP  : prf__ "GPC Channel initiated");             break;
   case TERM_GP  : prf__ "GPC Channel terminated");            break;
   case FINDCLOTH: prf__ "VAL II reports finding cloth");      break;
   case CORNER   : prf__ "VAL II reports finding corner");     break;
   case UPTO_NDLE: prf__ "VAL II has put cloth under needle");
   break;
   case FAR_RH   : prf__ "VAL II has found far RH corner"); break;
   case MOVEBACK : prf__ "VAL II has moved back a distance");
   break;
   case ST_ALTER : prf__ "VAL II has started ALTER");          break;
   case END_ALTER: prf__ "VAL II has terminated ALTER");       break;
   case RETREAT  : prf__ "Robot has retreated with cloth"); break;
   case WHERE    : prf__ "VAL II reported robot position"); break;
   case PARAM1   : prf__ "VAL II has input parameters #1"); break;
   case GO_START : prf__ "Robot is at start position");        break;
   case ALIGN_F  : prf__ "Instrumented finger is aligned"); break;
   case DROP     : prf__ "Robot has dropped onto cloth");      break;
   case PARAM2   : prf__ "VAL II has input parameters #2"); break;
   case GO_NEAR  : prf__ "Robot has moved to start.near");     break;
   case STARTUP  : prf__ "VAL II has sent startup data");      break;
   case FINEADJ  : prf__ "Fine adjustment completed");         break;
   case ANGLEADJ : prf__ "Fine angular adjust completed");     break;
   case ROTATE90 : prf__ "Robot has rotated cloth by 90");     break;
   case INCHMOVE : prf__ "Robot has completed inching");       break;
   case REMOVE   : prf__ "Robot has cleared needle zone"); break;
   case STRAIGHTN: prf__ "Robot has straightened out the cloth");
   break;
   case END_CLOTH : prf__ "Robot has found end of cloth");  break;
   default :        prf__ "VAL II sent unrecognisable code - %4d",
```

```
                         temp);
  }
  end_print;
  if ( code != (int)temp)
  {     displ_init;
       prf__ "Program terminated by VAL II -  ",
            "unsuccesful call to function no. %3d",code);
       end_print;
       gpf_start(TERM_GP);                  /* terminate GP comms */
       ajend();
  }
}
```

C.3.2. VAL II Implementation

```
PROGRAM main1
      CALL definitions
      SPEED hi.speed  ALWAYS
      TOOL fing1
      terminated = FALSE
      DO
;         TOOL fing1
          CALL inbyte
          IF incontrol THEN            ; check CONTROL line
              contout = TRUE
              CASE byte OF             ; control codes
                VALUE 1:
                  TYPE "IBM AT has initiated GP communications"
;                 CALL set.param3
                  CALL outbyte
                VALUE 2:
                  TYPE "IBM AT has terminated GP communications"
                  CALL outbyte
                  terminated = TRUE
                VALUE 3:
                  TYPE "IBM AT request - find cloth"
                  CALL findcloth
                  CALL outbyte
                VALUE 4:
                  TYPE "IBM AT request - find cloth corner"
                  CALL corner
                  CALL outbyte
                VALUE 5:
                  TYPE "IBM AT request - put cloth under needle"
                  CALL uptoneedle
                  CALL outbyte
                VALUE 6:
                  TYPE "IBM AT request - find far RH corner"
                  CALL far.rh
                  CALL outbyte
                VALUE 7:
                  TYPE "IBM AT request - move back distance"
                  CALL moveback
                  CALL outbyte
                VALUE 8:
                  TYPE "IBM AT request - start ALTER"
```

```
                    IF testing THEN
                        MOVES SHIFT(HERE BY 0, 0, 100)
                        BREAK
                    END
                    IF testing THEN
                        ALTER (0, 3)
                    ELSE
                        ALTER (0, 19)
                    END
                    REACT 1003, cloth.end
                    CALL outbyte
                    DELAY 800
                    TYPE "leaving start alter loop"
                  VALUE 9:
                    TYPE "IBM AT request - terminate ALTER"
                    BRAKE; remove delay
                    NOALTER
                    IGNORE 1003
                    CALL outbyte
                    BREAK
                  VALUE 10:
                    TYPE "IBM AT request - drag cloth away from needle"
                    CALL retreat
                    CALL outbyte
                  VALUE 11:
                    TYPE "IBM AT request - robot position data"
;                   MOVES SHIFT(start.near BY 0, 0, 100)
                    BREAK
                    CALL calc.where
                    CALL outbyte
                  VALUE 12:
                    TYPE "IBM AT request - enter gain parameters"
                    CALL set.param
                    CALL outbyte
                  VALUE 13:
                    TYPE "IBM AT request - is robot at start positn ?"
                    CALL check.start
                    CALL outbyte
                  VALUE 14:
                    TYPE "IBM AT request - align instrumented finger"
                    CALL align.finger
                    BREAK
                    CALL outbyte
                  VALUE 15:
                    TYPE "IBM AT request - set down 2nd finger"
                    DELAY 1
                    CALL set.down
                    MOVES SHIFT(HERE BY 0, 0, 100)
;                   MOVES start
                    BREAK
                    CALL outbyte
                  VALUE 16:
                    TYPE " set.param version 2 "
                    CALL set.param2
                    CALL outbyte
                  VALUE 17:
                    TYPE "IBM AT requests move to start.near"
                    CALL go.near.start
```

```
                CALL outbyte
              VALUE 18:
                TYPE "IBM AT request startup.data"
                CALL startup.data
                CALL outbyte
              VALUE 19:
                TYPE "IBM AT request - fine adjustment"
                CALL fine.adj
                CALL outbyte
              VALUE 20:
                TYPE "IBM AT request - fine angular adjustment"
                CALL angle.adj
                CALL outbyte
              VALUE 21:
                TYPE "IBM AT request - 90 degree turn"
                CALL rotate.90
                CALL outbyte
              VALUE 22:
                TYPE "IBM AT request - inching motion"
                CALL inch
                CALL outbyte
              VALUE 23:
                TYPE "IBM AT request - remove robot from needle zone"
                CALL remove
                CALL outbyte
              VALUE 24:
                TYPE "IBM AT request - straighten cloth"
                CALL straighten
                CALL outbyte
              VALUE 25:
                TYPE "IBM AT request - find cloth end"
                CALL end.cloth
                CALL outbyte
              VALUE 26:
                answer = 0
                TYPE "Do you want to continue ?"
                PROMPT "To continue enter in  -  1  ", answer
                IF answer <> 1 THEN
                    byte = 0
                END
                CALL outbyte
              ANY
                TYPE "IBM AT requests unknown function = ",/I5,byte
            END
        ELSE
            TYPE "IBM AT sent an unexpected data byte = ",/I5,byte
        END
    UNTIL terminated
END

PROGRAM definitions
 ;
 ;        This routine initialises variables and constants
          TOOL fingl
          table.ht = -498.5        ; z coordinate of table height
          test.level = -484
          hi.speed = 120           ; speed rate for fast motions
          pcdist = 78              ; distance between photocells
```

```
        theta.offset = 1.154 ; ang. offset of fing2 to x axis
        pc.to.fg = 55     ; y offset of finger #1 from pcell1
        fg.to.pc = -20      ; x offset of pcell1 to finger #1
        fing.dist = 156
        testing = TRUE
        straightening = FALSE
        pcell1.on = 1001        ; input no. for photocell #1
        pcell2.on = 1002        ; input no. for photocell #2
        r.max = 850       ; max reach of robot for NULL tool
        RETURN
END
```

C.4. Uplink Facility

The GPC communication link provided a simple method for transfering messages between VAL II programs and the AMX Tasks running on the IBM AT. However, since the UNIMATION Supervisor communication link was not implemented, other facilities, such as downloading programs from the IBM to VAL II, were not available. A method was developed for uploading programs from VAL II to the IBM AT, without requiring the Supervisor channel.

An RS 232C serial port on the IBM AT was linked to the PRINTER port on the back of the UNIMATION terminal. The following program assisted the upload operation :-

```
#include        "FCNTL.H"
#include        "STDIO.H"

#define         TRUE            1
#define         FALSE           0

#define         PUMAT           0x2f8           /* serial port #1 */
#define         LCR             3
#define         IIR             2
#define         PIIR            PUMAT + IIR
#define         LSR             5
#define         DLL             0
#define         DLM             1
#define         IER             1
#define         MSR             6
#define         MCR             4


init(port)
short int port;
{
        outp(port+LCR,0x83);
        outp(port+DLL,0x80);
        outp(port+DLM,0x01);
```

```
        outp(port+MSR,0x00);
        outp(port+MCR,0x00);
        outp(port+LCR,0x03);
        outp(port+IER,0x05);          /* set IER to ignore TxHRE */
        outp(port+IIR,0x01);
}

main()
{
    char byte, *mode = "w+", *name = "d:puma.lst", date[12];
    FILE *fp;

    init(PUMAT);

    fp = fopen(name,mode);
    if (fp == NULL)
        printf("\n error in opening lst file");

    printf("\nPress     PRINTER button   on Unimation terminal.");
    printf("\nEnter   -  PLIST progname   at Unimation terminal.");
    printf("\n\nWhen listing is completed, press CR on both ");
    printf("terminals  (IBM first, then Unimation)\n");

    getdate(&date[0]);


    while(!kbhit())
        if (putc(getbyte(), fp) == EOF)
                printf("Error in writing to file");
}

char getbyte()
{
        char iir1, blank = ' ';

                                            /* wait for interrupt */
        while ( (iir1 = inp(PIIR)) & 0x01 )
                        ;

        if ( iir1 == 0x04)
                return(inp(PUMAT));
        else if (iir1 == 0x06)
                printf("\n framing error");
        else
                printf("\n strange interrupt iir1 = %x",iir1);

        inp(PUMAT+LSR);
        return(blank);
}
```

APPENDIX D

THE SEW TASK

D.1. Restart Procedure

```
     /* Restart Procedure for SEW, CONT, MAKE and POST Tasks */

void rpsew()
{
        PMESS p;
        int i;
        char m_reg;
        static int gpaintcd[16], gpbintcd[16];
        ajmodl();

        displ_init;
        prf__ "restart procedure for sewing task");
        end_print;

        rpiptr();                    /* set up pointers to ISP's */
                                /* send control byte to prog I/O chips*/
        ajoutb(CB_IO_2,0x80);
        sh_delay();
        ajoutb(CB_IO_1,0xAE);              /* mode 1 I/O for GP comms */
        sh_delay();
        ajoutb(CB_IO_1,0x05);                 /* set INTR for port B  */
        sh_delay();
        ajoutb(CB_IO_1,0x0D);                 /* set INTR for port A  */
                                            /* reset counters to zero */
        ajoutb(CB_COUNTR,RESET_CNTR);
        sh_delay();
        ajoutb(CB_COUNTR,0);                     /* ready to count    */
        sh_delay();                /* enable latches to read count */
        ajoutb(CB_COUNTR,LATCH_EN);

        blp_fact = STITCH_LEN*TRK_FACT*SC_FACT*(-1.0)/36.;
        count2 = count1 = 0;
        t_period = 1;
                             /*  initialize I/O ports to sewing m/c */
        ajoutb(PORT_A,0);
        ajoutb(PORT_B,0);
        ajoutb(SPEED_P,0);
        cloth_end = TRUE;        /* disable cloth end signal via GPC */
                                     /* Parameter initialisations */
        GPInWait = GPOutWait = FALSE;
        b_port = y_0 = x_0 = 0;
        pix1_ofst = pix2_ofst = 0;
                                      /* instal interrupt pointer */
        ajiptr(UGPCAV,gpa_isp,gpaintcd);
        ajiptr(UGPCBV,gpb_isp,gpbintcd);

        m_reg = ajinb(U8259M);                 /* unmask interrupt */
        m_reg = m_reg & ~UIRQ5M;
        m_reg = m_reg & ~UIRQ3M;
```

```
        for (i = 300; i != 0; i--);
        ajoutb(U8259M, m_reg);

        if(ajtask(TNCONT))                          /* start CONT Task */
                crash(223);
}
```

D.2. Main Routine of SEW Task

```
void stsew()
{       PMESS p;
        int status, inc_u, inc_x, z_rot, flop, y_displ, dy_old,
              last_cnt, i_store, tens;
        float freq, e_0, b_0;
        unsigned int blp_cnt;
        long int t_intgrl;

        ajmodl();
        displ_init;
        prf__ "SEWing task started");
        end_print;

                        /* Initialisations */
        sewwait = completed = FALSE;
        blp_cnt = i_store = z_rot = ifeed = status = last_cnt
        y_displ = max_e = max_t = t_MeanDev = t_Avg = inc_x
        = i_t_Avg = dy_old = 0;
        e_MeanDev = e_Avg = 0.0;
        min_e = min_t = 10000;
        t_intgrl = 0L;
        flip = FALSE;     flop = TRUE;
        cloth_end = FALSE;                          /* awaiting signal via GPC */
        pmdata_pt = &pmdat[0];
        ifeed = i_hand = 1;

        tens = tension();
        sh_delay();
        tens = tension();
        count_reset();
        set_speed(0);

        take_picture();
        delay(10);
        read_cam();
        e_calc(&b_0,&e_0);


        ajoutb(PORT_A,SEW_START);                     /* start sewing now !! */

            /*  SENSORY FEEDBACK LOOP FOR REAL TIME ROBOT PATH CONTROL */

                                                      /* test for end of cloth */
        while (( x_total > StopDistance) && !cloth_end )
```

```
{                                           /* control sewing speed */
     status = speed_control(status);
                                      /* calc. avg. update frequency */
     freq = (float)ifeed/(float)i_hand;
          /* APPLYING CLOTH TENSION CONTROL */
                                    /* open loop cloth tension control */
     inc_u = x_corr(&blp_cnt,freq);
                                  /* closed loop cloth tension control */
     if (!sew_near)
     {                                     /* update tension if > 1 rev */
          if  ((blp_cnt  - last_cnt) > 36)
          {        last_cnt = blp_cnt;
                   tens = tension();
          }
                        /* reset t_intgrl after slack taken up */
          if (flop && tens > rq_tens)
          {        flop = FALSE;
                   t_Avg = t_intgrl = 0L;
                   t_MeanDev = i_t_Avg = 0;
                   min_t = 10000;
          }
     }
     inc_x = tens_corr(inc_u,tens,&t_intgrl,freq);
                                                /* limit x movement */
     inc_x = limit(inc_x,8*SC_FACT);

                /* APPLY SEAM WIDTH CONTROL */
     read_cam();                 /* transfer pixel data to buffer */
     take_picture();              /* trigger Z80 to read cameras */
     y_displ = y_corr(&inc_x,&z_rot,&dy_old);

                                   /* install new ALTER message */
     install(inc_x,y_displ,z_rot);

     if (ifeed < 100 )                     /* store runtime data*/
     {     *(pmdata_pt++) = (float)x_total;
           *(pmdata_pt++) = (float)y_total;
           *(pmdata_pt++) = (float)i_hand;
           *(pmdata_pt++) = (float)ifeed;
     }
     flip = flip ? FALSE : TRUE;

     ifeed++;
}                                            /* end of update Loop */

ajoutb(PORT_A,SEW_STOP);              /* stop sewing machine !! */
install(0,0,0);
cloth_end = TRUE;                     /* disable signal via GPC */

displ_init;
prf__ "initial error = %8.3f, initial beta = %8.3f",
            e_0/SC_FACT,b_0*RAD_TO_A);
end_print;

if (ajwake(TNMAKE) != 0) crash(2322);
}
```

D.3. Cloth Tension Control Routines

```
       /* This routine calculates an x displacement for the robot */
       /* to track the sewing machine shaft encoder signal. */
x_corr(blp_cnt,freq)
unsigned *blp_cnt;
float freq;
{
        int  inc_x;
        unsigned int new_blip_cnt;

        new_blip_cnt = (unsigned)read_count();
                                        /* check for counter overflow */
        if (new_blip_cnt < *blp_cnt)
                crash(7315);

        inc_x = (float)(new_blip_cnt - *blp_cnt)
                    *blp_fact*freq;
        *blp_cnt = new_blip_cnt;

        return( inc_x );
}


tens_corr(inc_x,tens,t_intgrl,freq)
int inc_x, tens;
long *t_intgrl;
float freq;
{
        int temp;

        temp = rq_tens - tens;
        *t_intgrl += ((float)temp/freq);
        t_Avg += temp;
        max_t = max(max_t,temp);
        min_t = min(min_t,temp);
        t_MeanDev += (temp*temp);
        i_t_Avg++;
                                   /*  Cloth Feed Servo Transfer Function */
        temp = (float)inc_x*(1.0 - (float)*t_intgrl*int_fact -
                                                 t_gain*(float)temp);
        if (temp > 0)                       /* ensure no moves backwards */
                temp = 0;
        return(temp);
}

speed_control(sew_status)
int sew_status;
{
    int x;
    static int  Pos_1, Pos_2, Pos_3;

    x = x_total;
    switch (sew_status)
    {
```

```
   case  0 :   Pos_1 = x_0 - acc_dist;
               Pos_2 = Pos_1 - calc_dist;
               Pos_3 = decel_dist;
               sp_len = 0;
               set_speed(0);
               return(1);

   case  1 :   if (x > Pos_1)
               {  if (sew_near)
                        set_speed(i_hand*2);
                  else
                        set_speed(i_hand*10);
                  return(1);
               }
               else
                  return(2);
   case  2 :   if (sew_near)
                  set_speed(MID_SPEED);
               else
                  set_speed(TOP_SPEED);
               ajtput((int)0, (unsigned)0xffff);
               count1 = (unsigned)read_count();
               Pos_1 = x;
               return(3);

   case  3 :   if ( x < Pos_2 )
               {  t_period = (unsigned)0xffff - ajtget((int)0);
                  count2 = (unsigned)read_count();
                  ajtoff((int)0);
                  sp_len = Pos_1 - x;
                  return(4);
               }
               else
                        return(3);

   case  4 :   if ( x < Pos_3)
               {  set_speed(SLO_SPEED);
                  return(5);
               }
  }
  return(sew_status);
}

void read_offset()
{       offst1 = ajinb(FING1);
        delay(1);
        offst1 = ajinb(FING1);
        delay(15);
        offst1 = ajinb(FING1);
}


void count_reset()
{                                         /* reset counters to zero     */
        ajoutb(CB_COUNTR,RESET_CNTR);
        sh_delay();
        ajoutb(CB_COUNTR,0);                     /* ready to count      */
        sh_delay();
```

```
        ajoutb(CB_COUNTR,LATCH_EN);
                                        /* enable latches to read count */
}


tension()
{       unsigned int tens;
        int tmp;

        tens = ajinb(FING1);
        tmp = (int)(tens - offst1);
        return( (tmp < 0)  ?   0  :  tmp);
}

limit(qty,lim)
int qty,lim;
{       if (qty > lim)
                return(lim);
        lim *= -1;
        return( (qty < lim) ? lim : qty);
}

void set_speed(sp_req)                  /* output speed request to m/c */
int sp_req;
{
        if (sp_req < 0)                      /* no -ve value possible */
                sp_req *= -1;

        if ( sp_req > 255)                 /* max. speed of sewing m/c */
                sp_req = 255;

        ajoutb(SPEED_P,sp_req);               /* change speed setting */
}


read_count()                        /* routine to read sewing m/c count */
{
        unsigned int count;

        ajoutb(CB_COUNTR,0);                        /* disable latches */
        count = ajinb(LO_COUNT);
        count += (ajinb(HI_COUNT) << 8);
        ajoutb(CB_COUNTR, LATCH_EN);                      /* re-enable */

        return((int)count);
}
```

D.4. Seam Width Control Rouines

```
y corr(inc t, z_rot, dy_old)
int *inc_t, *z_rot, *dy_old;
{       PMESS p;
        float  alpha1, alpha2,del_alpha, x1, y1, x2, y2, dx, dy;
        long int z1;
```

```
int  dy_i, dy_lim, h_freq, r, acc_lim;

ajdi();                         /*calc instantaneous position */
x1 = x_total;                            /* calc robot to ndle */
y1 = -y_total ;
z1 = z_total ;
ajei();
alpha1 = (float)z1 / ROT_FACT;

                                     /* apply transfer function */
del_alpha = - transf_fn();
                         /* calc robot position before limiting */
dx = -y1 * del_alpha;
x2 = x1 + dx;
dy = x1 * del_alpha;
y2 = y1 + dy;
dy_i = (int)dy;
alpha2 = alpha1 + del_alpha;

                        /* APPLY VARIOUS LIMITATIONS */
                                         /* velocity limitation */
dy_lim = limit(dy_i,vel_lim);
if (!sew_near)                             /* absolute limiting */
       dy_lim = limit2(dy_lim, (int)y1);
                                /* check robot within envelope */
h_freq = (float)(1.0/freq);
switch (envelope((int)x1,(int)y1,alpha1,&r))
{  case   1 : crash(1236);
              break;              /* fing 1 hits sewing m/c */
   case   2 : crash(1237);
              break;              /* fing 2 hits sewing m/c */
   case   3 : crash(1238);
              break;                        /* robot too far */
   case   4 : crash(1239);
              break;                       /* robot too near */
   case   5 : acc_lim = accel_lim*h_freq;
              break;                        /* close to base */
   case   6 : acc_lim = accel_lim*h_freq/3;
                                           /* far from base */
}
                                    /* acceleration limitation */
dy_lim = limit(dy_lim - *dy_old,acc_lim) + *dy_old;
dy_lim *= h_freq;
dy_lim = limit3(dy_lim, r);                /* absolute limiting */
dy_lim /= h_freq;
                          /* if limited then recalc position */
if (dy_lim != dy_i)
{       dy_i = dy_lim;
        dy = (float)dy_i;
        del_alpha = dy/x1;
        dx = -(y1*del_alpha);
        x2 = x1 + dx;
}

*dy_old = dy_i;                 /* store del_y for accel limit*/
                                  /* return z rot ALTER data */
*z_rot = del_alpha * ROT_FACT;
*inc_t += dx;                     /* return new x ALTER data */
```

```
        return(dy_i);                       /* return new y ALTER data */
}

envelope(x,y,alpha,r_r)
int x;
int y;
float alpha;
int *r_r;
{       int du,dv,r;
        long int  f_x, f_y, rx, ry;
        float sin_a, cos_a, sin_t, cos_t;
        double tt;

        if (HitSewMc(x,y))
                return(1);                          /* main finger hits */

        sin_a = sin(alpha);                /* calc 2nd finger position */
        cos_a = cos(alpha);
        du = (float)fing_dist*sin_a;
        dv = (float)fing_dist*cos_a;
        if (HitSewMc(x - du,y + dv))                /* 2nd finger hits */
                return(2);
                                      /* calc position of robot flange */
        cos_t = (cos_a*cos_f) - (sin_a*sin_f);
        sin_t = (sin_a*cos_f) + (sin_f*cos_a);
        f_x = (float)f_r*cos_t;
        rx = n_x + x + f_x;
        f_y = (float)f_r*sin_t;
        ry = n_y - y - f_y;
        tt = (rx*rx) + (ry*ry);
        tt = sqrt(tt);                      /* calc robot reach radius */
        r = (int)tt;

        *r_r = r;
        if (r > R_MAX)
                return(3);                                 /* too far */
        if (r < R_MIN)
                return(4);                                /* too near */
        if (r < R_MID)
                return(5);                           /* close to base */
        return(6);                                   /* far from base */
}

        /* this routine returns angular correction  based on         */
        /* proportional gain (gain_pix) and derivative gain (beta). */
float transf_fn()
{
        float error,beta;
        int i;

        e_calc(&beta,&error);             /* calc error from pixel data*/

        e_Avg += error;                   /* calc seam error statistics*/
        e_MeanDev += (error*error);
        min_e = min(min_e,(int)error);
        max_e = max(max_e,(int)error);

        for( i = NCOL; (error < pixel1[i-1]) && (i >= 0 ) ; i--)
```

```
                ;

        return((float)((deriv_gain*beta) + gain_pix[i]) );
}

                /* This routine calcs. actual seam width error */
void e_calc(beta_pt, error_pt)
float *beta_pt;
float *error_pt;
{  float  np_1, np_2;
   int icol1, icol2;

   icol1 = find_edge(cam1_buf,irow1);

   np_1 = SEAM_W + pixel1[icol1] + pix1_ofst;
   icol2 = find_edge(cam2_buf,irow2) ;

   np_2 = SEAM_W + pixel2[icol2] + pix2_ofst;

   *beta_pt = (np_1 - np_2) / CAM2_DIST;
   *error_pt = (np_1 * rcos((double)*beta_pt)) - (float)SEAM_W;
}

                        /* This routine applies absolute limiting */
limit2(y_dis,old_y)
int y_dis,old_y;
{
   int sig, retn, temp;

   temp = y_dis + old_y;
   if (temp > 0)
   {     sig = 1;                           /* sign  of y direction */
         temp = LEFT_MAX - temp;          /* dist between limit & y */
   }
   else
   {     sig = -1;                  /* -ve y is on the left hand side */
         temp = RIGHT_MAX + temp;          /* absolute value required */
   }

   if (temp > 1280)                   /* 40 mm  -   well within limits */
        return(y_dis);
   if ((y_dis > 0 && old_y < 0) || (y_dis < 0 && old_y > 0))
        return(y_dis);                           /* approaching centre */
   if (temp > 832)                     /* 26 mm  -   approaching limit */
        retn = 192;                    /*  6 mm    -      deceleration */
   else if (temp > 512)                /* 16 mm */
        retn = 160;                    /*  5 mm */
   else if (temp > 320)                /* 10 mm */
        retn =  96;                    /*  3 mm */
   else if (temp > 192)                /*  6 mm */
        retn =  64;                    /*  2 mm */
   else if (temp > 64)                 /*  2 mm */
        retn = 32;                     /*  1 mm */
   else
        return(0);                             /* dead zone near limit */
   if (retn < abs(y_dis))
        return(retn*sig);                   /* return deceleration speed */
   return(y_dis);                           /* else y_dis is slow enough */
```

```
}


limit3(y_dis,r)              /* This routine applies absolute limiting */
int y_dis,r;
{
   int sig, retn, temp;

   if (r < R_MID)
   {    sig =  2;                         /* sign  of y direction */
        temp = r - R_MIN - 150;           /* dist between limit & y */
   }
   else
   {    sig = -1;
        temp = R_MAX - r - 150;
   }

   if (temp > 1600)                      /* 50 mm  - well within limits */
        return(y_dis);
   if ((y_dis > 0 && r > R_MID) || (y_dis < 0 && r < R_MID))
        return(y_dis);                          /* approaching centre */
   if (temp > 832)                        /* 26 mm  - approaching limit */
        retn = 70;                        /*  6 mm  -      deceleration */
   else if (temp > 512)                   /* 16 mm */
        retn =  50;                       /*  4 mm */
   else if (temp > 320)                   /* 10 mm */
        retn =  30;                       /*  3 mm */
   else if (temp > 192)                   /*  6 mm */
        retn =  10;                       /*  2 mm */
   else if (temp > 64)                    /*  2 mm */
        retn =  5;                        /*  1 mm */
   else if (temp < -194 && sig > 0)
        retn =  -10;
   else if (temp < -64 && sig > 0)
        retn =  -5;
   else
        return(0);                            /* dead zone near limit */
   if (retn < abs(y_dis))
        return(retn*sig);                  /* return deceleration speed */
   return(y_dis);                          /* else y_dis is slow enough */
}


HitSewMc(x,y)
int x,y;
{                                        /* checking sewing m/c envelope */
        if ((y < NY_MAX) && (x < NX_MAX) && (x > NX_MIN))
                return(TRUE);
        return(FALSE);
}


edge_find(cam_buf,irow)
char *cam_buf;
int irow;                         /* no. of pixel row to be searched */
{
        int ipix, icol;
```

```
        ipix = (irow*NCOL);             /* N.B. irow starts at zero  */

        for (icol = 0; icol < NCOL; icol++,ipix++)
        {
                if ( *(cam_buf+ipix) < 0x80 )
                        return(icol);
        }
        return(NCOL-1);
}

find_edge(cam_buf,irow)
char *cam_buf;
int irow;
{
        int icol1, icol2, icol3;

        icol1 = edge_find(cam_buf,irow-1);
        icol2 = edge_find(cam_buf,irow);
        icol3 = edge_find(cam_buf,irow+1);
        return (icol1 + icol2 + icol3)/3;
}


      /* approximation based on first 2 terms of the series expansion */
double rcos(angle)
double angle;
{
        return (1.0 - (angle*angle/2.0));
}
```

APPENDIX E

THE CONT, MAKE AND POST TASKS

E.1. The CONT Task

```
void stcont()
{   PMESS p;

   displ_init;
   prf__ "CONT task started");
   end_print;
                                              /* Initial Sequence */
   debug = FALSE;
   ajoutb(PORT_A,PRESSER_FT);              /* lift presser foot up */
   gp_function(INIT_GP);                     /* initiate GP comms */

   do
   {    gp_function(GO_START);               /* is robot at start? */
        read_offset();
        startup_data();
        set_param();

        ajtask(TNMAKE);
        ajwait();
   } while (again());

    gp_function(TERM_GP);                    /* terminate GP comms*/
}


void startup_data()
{       gpf start(STARTUP);
        fing dist = get word();
        f_r = get_word();
        f_angle = (float)get_word()/180.0;
        n_x = get_word();
        n_y = get_word();
        cos_f = cos(f_angle/RAD_TO_A);
        sin_f = sin(f_angle/RAD_TO_A);
        gpf_end(STARTUP);
}

void set_param()
{       gpf_start(PARAM1);                      /* input parameters */
        initialise();
        set1_param();
        pix1_ofst = Y1_PIXEL * (float)ipix1_ofst;
        pix2_ofst = Y2_PIXEL * (float)ipix2_ofst;
        setup_pixels();
        gpf_end(PARAM1);
}
```

```
void setl_param()
{
  PMESS p;
  int temp;

  do
  {  temp = get_word();
     if (temp != 0) irowl = temp;
     temp = get_word();
     if (temp != 0) irow2 = temp;
     temp = get_word();
     if (temp != 0) ipixl_ofst = temp;
     temp = get_word();
     if (temp != 0) ipix2_ofst = temp;
     temp = get_word();

     displ_init;
prf__ "irowl = %4d, irow2 = %4d, ipixl_of = %4d, ipix2_of = %4d",
              irowl,irow2,ipixl_ofst, ipix2_ofst);
     end_print;

     if (temp != 0) s_gain = (float)temp/(100000.0*SC_FACT);
     temp = get_word();
     if (temp != 0) deriv_gain = (float)temp/10000.0;
     temp = get_word();
     if (temp != 0) int_fact = (float)temp/1000000.0;

     displ_init;
     prf__ "s_gain = %6.4f, deriv_gain = %6.3f, int_fact = %9.6f",
              s_gain*SC_FACT, deriv_gain, int_fact);
     end_print;
     pix_gain = s_gain * Y1_PIXEL / SC_FACT;

     temp = get_word();
     if (temp != 0) t_gain = (float)temp/100000.0;
     temp = get_word();
     if (temp != 0) rq_tens = temp;
     temp = get_word();
     if (temp != 0) accel_lim = temp;
     temp = get_word();
     if (temp != 0) vel_lim = temp;

     displ_init;
     prf__
"t_gain = %8.5f,  rq_tens = %4d, accel_lim = %4.2f, vel_lim = %4d",
     t_gain,rq_tens,(float)accel_lim/SC_FACT,vel_lim/SC_FACT);
     end_print;
  } while (get_word() != 1);
}


       /* This routine initialises global parameters to default values */
void initialise()
{
        t_gain =  0.0015;
        int_fact = 0.00003;
```

```
        deriv_gain = 0.1;
        s_gain = 0.005/SC_FACT;
        pix_gain = s_gain * Y1_PIXEL;
        rq_tens =  70;
        accel_lim = (3.0 * SC_FACT);
        vel_lim =  8 * SC_FACT;
        irow1 = 2;
        irow2 = 8;
        ipix1_ofst = 0;
        ipix2_ofst = 0;
}

again()
{       int ans;

        gpf_start(Q_AGAIN);
        ans = (int)get_byte();
        return( ans == Q_AGAIN ? TRUE : FALSE);
}


void setup_pixels()
{
       /*       PMESS p;                                                  */
        int gain_sign, centre_pix, i, factor;
        int gainswitch;
        float halfpix_gain, half1_width, half2_width;

        centre_pix = NCOL / 2;
        factor = centre_pix;
        half1_width = Y1_PIXEL / 2.0;
        half2_width = Y2_PIXEL / 2.0;
        pix1_ofst = Y1_PIXEL * (float)ipix1_ofst;
        pix2_ofst = Y2_PIXEL * (float)ipix2_ofst;

        gainswitch = 0;
        gain_sign = -1;
        halfpix_gain = (Y1_PIXEL/2.0) * pix_gain ;

       /*       print_init;
        prf__ "\npixel arrangement",
"\n   i     factor    pixel1[]    pixel2[]    gain[]\n");
        end_print;
                                                                      */
        for (i=0; i < NCOL; i++)
        {
                pixel1[i] = half1_width - (Y1_PIXEL * factor);
                pixel2[i] = half2_width - (Y2_PIXEL * factor);
                gain_pix[i] = - (halfpix_gain * gain_sign)  -
                        (pixel1[i - gainswitch] * pix_gain) ;

       /*               print_init;
                prf__ "%4d%10d%13.2f%13.2f%13.3f",
                       i,factor,pixel1[i],pixel2[i],gain_pix[i]);
                end_print;
                                                                          */
                if (factor == 0)
                {
```

```
                    gainswitch = 1;
                    gain_sign  = 1;
            }
            factor--;
        }
        gain_pix[NCOL] = - halfpix_gain  -
                            (pixel1[NCOL - 1] * pix_gain);
       /*       print_init;
        prf__ "%46.5f ",gain_pix[NCOL]);
        end_print;                                     */
}
```

E.2. The MAKE Task

```
#define REMNANT 30*SC_FACT              /* cloth length left to sew  */
#define NSIDES  3

void stmake()
{
   PMESS p;
   int i, section[8], i_sect, no_sections;
   ajmodl();
   displ_init;
   prf__ "SEAM task started");
   end_print;

   gp_function(FINDCLOTH);                           /* find cloth         */
   gp_function(CORNER);                               /* find cloth corner */
   gp_function(UPTO_NDLE);                        /* put cloth under needle */
   fine_adj();
   ndle_down();                      /* put needle down to permit pivot */
   gp_function(REMOVE);
                     /* remove robot from immediate vicinity of needle */

                                  /* Looped sequence */
   for(i=0; i < NSIDES; i++)
   {    gp_function(END_CLOTH);
        no_sections = DecideSeam(&section[0]);
        for (i_sect = 0; i_sect < no_sections; i_sect++)
        {       sew_near = section[i_sect];
                if (sew_near)
                     gp_function(GO_NEAR);
                else
                     gp_function(FAR_RH);
                angle_adj();
                CalcSeamSection();
                if (ajtask(TNCOMM))
                    crash(6543);                /* start ALTER Comms */
                ajshed();
                ajwatm(4);
                gp_function(ST_ALTER);               /* start ALTER up */
                ajtask(TNSEW);
                ajwait();
                gp_function(END_ALTER);            /* terminate ALTER */
```

```
        }
        inch();                        /* finish off last 15 mm of seam */
        if (i == NSIDES-1) break;
        gp_function(ROTATE90);                   /* rotate cloth by 90 */
        gp_function(STRAIGHTN);              /* straighten out cloth */
    }
    ajoutb(PORT_A,TRIM_THREAD);
    delay(10);
    gp_function(RETREAT);                        /* pull cloth back */

       /*  if (ajtask(TNPOST)) crash(1837);                  */

   if (ajwake(TNCONT) != 0) crash(2382);
}


DecideSeam(section)
int *section;
{       where();
        if (x_0 > 150*SC_FACT)
        {       *section = FAR;
                *(section+1) = NEAR;
                return(2);
        }
        *section = NEAR;
        return(1);
}

void CalcSeamSection()
{
        where();
        if (sew_near)
        {       StopDistance = -100*SC_FACT;
                acc_dist = 20*SC_FACT;
                decel_dist = 35*SC_FACT;
                SeamSection = x_0;
        }
        else
        {       StopDistance = 180*SC_FACT;
                acc_dist = 55*SC_FACT;
                decel_dist = 10*SC_FACT + StopDistance;
                SeamSection = x_0 - StopDistance;
        }
        calc_dist = x_0 - decel_dist - acc_dist - 20*SC_FACT;
}

        /* this rotine sews up last 30 mm of cloth after */
        /* photocell uncovered */
void inch()
{       gpf_start(INCHMOVE);
        send_word((int)REMNANT);
        ajoutb(PORT_A,SLO_SEW);
        delay(600);                    /* this delay = 30 mm travel */
        ajoutb(PORT_A,SEW_STOP);
        delay(150);
        ajoutb(PORT_A,PRESSER_FT);
        gpf_end(INCHMOVE);
}
```

```
void ndle down()
{
        ajoutb(PORT_A,SLO_SEW);
        delay(50);
        ajoutb(PORT_A,SEW_STOP);
        delay(150);
        ajoutb(PORT_A,PRESSER_FT);
}

void fine_adj()
{       gpf_start(FINEADJ);
        adjust(TRUE);
        gpf_end(FINEADJ);
}

void angle_adj()
{       gpf_start(ANGLEADJ);
        adjust(FALSE);
        gpf_end(ANGLEADJ);
}

void adjust(width_adj)
int width_adj;
{       PMESS p;
        float beta,error;
        int ack;

        ack = 0;
        while (ack < 9)
        {       take_picture();
                delay(3);
                read_cam();
                e_calc(&beta,&error);
                beta *= RAD_TO_A;               /* convert to degrees */
                displ_init;
prf__ "Fine adj. -  error = %6.2f beta = %6.3f ", error, beta);
                end_print;
                if (width_adj)
                     send_word((int)error);
                else
                     send_word((int)beta);
                ack = get_word();
        }
        if (ack != 10)
        {       displ_init;
prf__ "Program terminated - unsuccesful Fine Adjustment");
                end_print;
                ajend();
        }
}


void where()                        /* VAL II returns robot position data */
{       PMESS p;
```

```
        gpf_start(WHERE);
        x_0 = get_word();                          /* initial x dist */
        y_0 = get_word();                          /* initial y_sc   */
        th_0 = ((float)get_word())/200.0;          /* initial theta  */
        x_total = x_0;                         /* initialise counters */
        y_total = y_0;
        z_total = -(float)(th_0*TOANG);
        gpf_end(WHERE);
        displ_init;
        prf__"x_0 = %5d, y_0 = %5d, th_0 = %6.2f, z_total = %6ld",
              x_0,y_0,th_0, z_total);
        end_print;
}
```

E.3. The POST Task

```
void stpost()
{
    PMESS p;
    float feed_sp, rev_speed, e_StdDev, t_StdDev;
    ajmodl();

    displ_init;
    prf__
    "COMM handshakes = %5d, feedback updates = %5d",i_hand,ifeed);
    end_print;


    feed_sp = (float)sp_len * 18.0 / ((float)t_period * SC_FACT) ;
    rev_speed = (float)(count2-count1)*18.0 * 60.0
                                /((float)t_period*36.0);

                                /* calc error statistics */
    e_MeanDev /= (float)(SC_FACT*SC_FACT);
    e_Avg /= (float)SC_FACT;
    e_StdDev = StdDev(e_MeanDev,e_Avg,ifeed);
    t_StdDev = StdDev((float)t_MeanDev,(float)-t_Avg,i_t_Avg);


    displ_init;
    prf__"no. ALTER handshakes     = %6d
               no. feedback loops      = %6d",i_hand,ifeed);
    end_print;
    displ_init;
    prf__"handshakes/update rate  = %6.2f
                          time period for speed   = %6u ticks",
    (float)i_hand/(float)ifeed, t_period);
    end_print;

    displ_init;
    prf__ "\n\nParameters  Set  At  Run  Time\n");
    end_print;
    displ_init;
    prf__"seam length                = %6d mm
```

```
                  sewing speed                = %7.1f rpm",
SeamSection/SC_FACT,rev_speed);
end_print;
displ_init;
prf__"tension offset           = %6u
            sewing speed             = %7.2f mm/s",offstl,feed_sp);
end_print;
displ_init;
prf__ "\n\nRobotic  Sewing  Performance  Data\n");
end_print;
displ_init;
prf__"seam width servo                   cloth tension servo");
end_print;
displ_init;
prf__"standard deviation       = %7.3f
            standard deviation       = %7.3f",e_StdDev,t_StdDev);
end_print;
displ_init;
prf__"sum of mean deviation    = %7.1f
            sum of mean deviation    = %71d",e_MeanDev,t_MeanDev);
end_print;
displ_init;
prf__"sum of average error     = %7.2f
                 sum of average error     = %71d",e_Avg,-t_Avg);
end_print;
displ_init;
prf__"maximum error            = %7.2f
                          maximum error            = %7d",
(float)max_e/(float)SC_FACT,-min_t);
end_print;
displ_init;
prf__"minimum error            = %7.2f
                          minimum error            = %7d",
(float)min_e/(float)SC_FACT,-max_t);
end_print;

pr_heading();

print_init;
prf__ "\nParameters  Set  At  Compile  Time\n");
end_print;
print_init;
prf__"robot stopping dist      = %6d mm
                      pixel width - cam #1     = %7.3f mm",
StopDistance/SC_FACT,Y1_PIXEL/SC_FACT);
end_print;
print_init;
prf__"maximum RHS motion       = %6d mm
                      pixel width - cam #2     = %7.3f mm",
RIGHT_MAX/SC_FACT,Y2_PIXEL/SC_FACT);
end_print;
print_init;
prf__"maximum LHS motion       = %6d mm
                          dist. between 2 fingers  = %6d mm",
LEFT_MAX/SC_FACT,fing_dist/SC_FACT);
end_print;
print_init;
prf__"deceleration length      = %6d mm
```

```
                              inter camera distance    = %7.1f mm",
decel_dist/SC_FACT,CAM2_DIST/SC_FACT);
end_print;
print_init;
prf__"stitch length             = %6d mm
                    seam width                  = %7.1f mm",
STITCH_LEN,(float)(SEAM_W/SC_FACT));
end print;
print_init;
prf__ "\n\nParameters  Set  By  User\n");
end_print;
print_init;
prf__"pixel row no. - cam #1  = %6d
                tensn servo, propnl gain  = %8.5f",irow1,t_gain);
end_print;
print_init;
prf__"pixel row no. - cam #2  = %6d
            tensn servo, intgrl gain  = %8.5f",irow2, int_fact);
end_print;
print_init;
prf__"x axis offset - cam #1  = %6d pxls
            request cloth tension     = %8d",ipix1_ofst,rq_tens);
end_print;
print_init;
prf__"x axis offset - cam #2  = %6d pxls
                              seam servo, propnl gain   = %8.4f",
ipix2_ofst,s_gain*SC_FACT);
end_print;
print_init;
prf__"robot velocity limitatn = %6d mm/hs
                    seam servo, deriv gain    = %8.3f",
vel_lim/SC_FACT,deriv_gain);
end_print;
print_init;
prf__"robot accelrtn limitatn = %6.1f mm/hs/hs",
                                    (float)accel_lim/SC_FACT);
end_print;
print_init;
prf__ "\n\nParameters  Set  At  Run  Time\n");
end_print;
print_init;
prf__"seam length               = %6d mm
                    sewing speed                = %7.1f rpm",
SeamSection/SC_FACT,rev_speed);
end_print;
print_init;
prf__"tension offset            = %6u
       sewing speed              = %7.2f mm/s",offst1,feed_sp);
end_print;

print_init;
prf__ "\n\n%c%c
                    Output Data%c%c",27,69,27,70);
end_print;
print_init;
prf__ "\nProcessor  Performance  Data\n");
end_print;
print_init;
```

```
   prf__"no. ALTER handshakes     = %6d
                no. feedback loops      = %6d",i_hand,ifeed);
   end_print;
   print_init;
   prf__"handshakes/update rate  = %6.2f
                        time period for speed   = %6u ticks",
   (float)i_hand/(float)ifeed, t_period);
   end_print;

   print_init;
   prf__ "\n\nRobotic  Sewing  Performance  Data\n");
   end_print;
   print_init;
   prf__"%c%cseam width servo
                    cloth tension servo%c%c",27,69,27,70);
   end_print;
   print_init;
   prf__"standard deviation       = %7.3f
         standard deviation       = %7.3f", e_StdDev,t_StdDev);
   end_print;
   print_init;
   prf__"sum of mean deviation   = %7.1f
         sum of mean deviation    = %7ld",e_MeanDev,t_MeanDev);
   end_print;
   print_init;
   prf__"sum of average error     = %7.2f
                sum of average error     = %7ld",e_Avg,-t_Avg);
   end_print;
   print_init;
   prf__ "maximum error               = %7.2f
                              maximum error              = %7d",
   (float)max_e/(float)SC_FACT,-min_t);
   end_print;
   print_init;
   prf__"minimum error              = %7.2f
                          minimum error              = %7d",
   (float)min_e/(float)SC_FACT,-max_t);
   end_print;

   print_init;
   prf__ "%c",12);
   end_print;
   pr_runtime();
   }


float StdDev( x1, x2, n)
float x1, x2;
int n;
{
        return((float)(sqrt((double)((x1 - (x2*x2/(float)n) )
                         /(float)(n-1)) )));
}


void pr_runtime()
{
   PMESS p;
```

```
  int ii,i, ind, no_data;
  float item[10];

  ind = 0;
  print_init;
  prf__
  "%c%c            Sensory  Feedback  Loop  Runtime  Data%c%c",
               27,69,27,70);
  end_print;

  print_init;
  prf__
    "\n error  beta   del_alph  dy_i   inc_x   inc_t  ",
    " y_dis   x_total y_total z_total");
  end_print;

  ifeed = ifeed  + 1;
  no_data = ifeed > 200 ? 200 : ifeed/2;
  for (ii = 0; ii < no_data ; ii++)
  {                                 /* recoup data from storage */
       for(i=0; i < 10; i++)
               item[i] = pmdat[ind++];

       print_init;
       prf__
  "%5.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.2f %7.1f %7.1f %7.1f",
  item[0]/SC_FACT,item[1]*RAD_TO_A,
  item[2]*RAD_TO_A,item[3]/SC_FACT,item[4]/SC_FACT,
  item[5]/SC_FACT,item[6]/SC_FACT,
  item[7]/SC_FACT,item[8]/SC_FACT,item[9]/TOANG);
        end_print;

        ajwatm(5);
  }
}



void pr_heading()
{
        PMESS p;
        struct {
                char sec;
                char mins;
                char hour;
                char day;
                char month;
                char year;
                char day_of_wk;
                char valid;
                } tdbuf;

        ajmodl();
        print_init;
        prf__ "%c%c%c%c",27,67,0,12);
        end_print;
        ajtdg(&tdbuf);
```

```
        print_init;
prf__ "\n\n\n                                                          ");
        p.n += ajtdf(&tdbuf,(int)0x00c2,&p.mp[p.n+1]);
        end_print;
        print_init;
        prf__
"%c%c                        Robotic Sewing Development Program",27,69);
        end_print;
        print_init;
        prf__  "%c%c                                       Version %4.
                                 2f",27,70,VERSION);
        end_print;

        print_init;
        prf__ "\n\n%c%c                                    Input Data%c%c",
                                 27,69,27,70);
        end_print;
}
```

E.4. VAL II Functions

```
PROGRAM angle.adj
             tries = 0
             SPEED 10  ALWAYS
        324  DELAY 0.5
             CALL inword
             ang = word
             TYPE "Angle Adjustment : angle = ", /I6, ang
             IF ABS(ang) < 2 GOTO 325
             IF ABS(ang) > 45 GOTO 326
             angle.req = -ang
             CALL rotate.ndle
             tries = tries+1
             word = tries
             CALL outword
             DELAY 0.5
             GOTO 324
        325  word = 10
             CALL outword
             SPEED hi.speed  ALWAYS
             RETURN
        326  TYPE "excessive angular error"
             HALT
END

PROGRAM calc.where
             DECOMPOSE pt[] = HERE
             x.0 = DX(HERE)-DX(at.ndle)
             y.0 = DY(HERE)-DY(front.ndle)
             th.0 = 90-pt[3]+theta.offset
             TOOL t.store
             TYPE "x coord of finger w.r.t. needle =", /D, x.0
             TYPE "y coord of finger w.r.t. needle = ", /D, y.0
```

```
          TYPE "2nd finger angle          = ", /D, th.0
    ;
          word = x.0*32
          CALL outword
          word = y.0*32
          CALL outword
          word = th.0*200
          CALL outword
          RETURN
END

PROGRAM calc.y.inc
    ;
    ;     This routine calculates the maximum increase in y
    ;     for the present x value.
          SET t.store = TOOL
          TOOL NULL
          y.inc.max = SQRT(SQR(r.max)-SQR(DX(HERE)))-DY(HERE)
          TOOL t.store
    ;
    ;   apply software limitation of short integers scaled by 32
          IF (DY(HERE)+y.inc.max) > 1020 THEN
              y.inc.max = 1020-DY(HERE)
          END
          RETURN
END

PROGRAM check.start
          x = DISTANCE(HERE, start)
          IF (x < 0.3) AND (x > -0.3) THEN
              RETURN
          END
          MOVES start
          DELAY 2.5
          BREAK
          RETURN
END


PROGRAM cloth.end
    ;     CALL outbyte
          RETURN
END

PROGRAM corner
    ; This routine sends robot up cloth length, finds top edge,
    ; aligns hand with cloth, finds LH edge, and places fingers
    ; down on cloth at an offset from top LH corner.
    ;
    ;     move forward until top edge detected
          SET t.store = TOOL
          REACTI pcell1.on
          REACTI pcell2.on
          SPEED 80
    ;
          MOVES SHIFT(HERE BY DX(limit.2)-DX(HERE), 0, 0)
          BREAK
          IF DX(HERE) < DX(limit.2)+30 GOTO 10
```

```
          IGNORE pcell1.on
          IGNORE pcell2.on
 ;
 ; move backwards and repeat search slowly and accurately
          MOVES SHIFT(HERE BY 35, 0, 0)
          BREAK
          REACTI pcell1.on
          REACTI pcell2.on
          SPEED 15  ALWAYS
          MOVES SHIFT(HERE BY -50)
          BREAK
          IGNORE pcell1.on
          IGNORE pcell2.on
 ;
 ;        test to decide on next move
;
; test for error condition, i.e. when neither pcell lit up
          IF SIG(pcell1.on) GOTO 31
          IF SIG(pcell2.on) GOTO 31
          TYPE "error in finding top edge"
          GOTO 10
      31  IF SIG(-pcell1.on) GOTO 30
          IF SIG(-pcell2.on) GOTO 32
          angle = 0
          GOTO 33         ; if both lit up then no need to pivot
 ;
 ;        pivot photocell no. 2 until pcell no. 1 detects  edge
      30  REACTI pcell1.on
          TOOL pcell2
          SET pivot = HERE
          FOR angle = 90 TO 0 STEP -0.5
              MOVE pivot:TRANS(0, 0, 0, angle, -90, 0)
              IF SIG(pcell1.on) GOTO 55
          END
      55  BREAK
          SET pivot:temp = HERE
          IF angle == 0 GOTO 10
          GOTO 34
 ;
 ;        pivot about pcell 1 until pcell 2 detects edge
      32  TOOL pcell1
          REACTI pcell2.on
          SET pivot = HERE
          FOR angle = 90 TO 180 STEP 0.5
              MOVE pivot:TRANS(0, 0, 0, angle, -90, 0)
              IF SIG(pcell2.on) GOTO 56
          END
      56  BREAK
          SET pivot:temp = HERE
          IF angle == 0 GOTO 10
 ;
 ;        calculate angle of cloth
      34  TOOL t.store
          DECOMPOSE pt3[] = temp
          angle = 90-pt3[3]
 ;
 ;        move gripper back a "margin" from top edge
      33  BREAK
```

```
        margin = 20
        x1 = margin*COS(angle)
        y1 = margin*SIN(angle)
        TYPE "cloth orientation angle = ", /I5, angle
        SPEED 80  ALWAYS
        MOVES SHIFT(HERE BY x1, y1, 0); move perpend to edge
        BREAK
;
;       move right to detect corner
        REACTI pcell1.on
        y1 = DY(limit.1)-DY(HERE)
        x1 = -y1*SIN(angle)/COS(angle)
        SPEED 30  ALWAYS
        MOVES SHIFT(HERE BY x1, y1, 0)
        BREAK
        IF SIG(-pcell1.on) GOTO 10
;
;       move backwards to position fingers over cloth
        x.offset = 15
        y.offset = -83              ; offset in y direction
        x1 = x.offset*COS(angle)-y.offset*SIN(angle)
        y1 = x.offset*SIN(angle)+y.offset*COS(angle)
        SPEED 70  ALWAYS
        MOVES SHIFT(HERE BY x1, y1, 0)
        BREAK
;
;       lower fingers onto cloth
        SPEED 12
        drop = table.ht-DZ(HERE)
        MOVES SHIFT(HERE BY 0, 0, drop)
        BREAK
        SPEED hi.speed  ALWAYS
        RETURN
    10  byte = 0
        TYPE "error in finding corner"
        RETURN
END

PROGRAM end.cloth
;
;       This routine moves robot back to find end of the cloth
        SPEED 80  ALWAYS
        REACTI pcell1.on
        MOVES down.line
        BREAK
        IGNORE pcell1.on
        SPEED hi.speed  ALWAYS
        RETURN
END

PROGRAM far.rh
;
;       find right hand corner
        MOVES SHIFT(HERE BY -35, 0, 0)
        BREAK
        IF SIG(pcell1.on) GOTO 10
        REACTI pcell1.on
        SPEED 60  ALWAYS
```

```
        CALL calc.y.inc
        MOVES SHIFT(HERE BY 0, y.inc.max, 0)
        BREAK
        IF SIG(-pcell1.on) GOTO 10
        IGNORE pcell1.on
;
;       put down fingers
        MOVES SHIFT(HERE BY -25, -pc.to.fg-30, 0)
        SPEED 10
        MOVES SHIFT(DEST BY 0, 0, table.ht-DZ(DEST))
        BREAK
;
        SPEED hi.speed  ALWAYS
        RETURN
    10  byte = 0
        TYPE "error in finding far RH corner"
        RETURN
END

PROGRAM findcloth
;
;     this routine finds cloth, calculates width, and places
;       gripper in centre of cloth.
;
        SPEED hi.speed
        MOVES start                       ; high up over table
        BREAK
        MOVES test1             ; down to photocell test level
        BREAK
        REACTI -pcell1.on
        MOVES limit.1           ; scan right until edge found
        BREAK
        IF DY(HERE) > DY(limit.1)-30 GOTO 12
        SET temp = HERE                   ; LH edge of cloth
        REACTI pcell1.on
        SPEED 80  ALWAYS
        MOVES limit.1                     ; scan right
        BREAK
        IF SIG(-pcell1.on) GOTO 12
;
;          calculate centre of cloth and move gripper there
        width = DY(HERE)-DY(temp)
        y2 = (width-pcdist)/2+30
        MOVES SHIFT(HERE BY 0, -y2, 0)
        BREAK
        TYPE "width = ", /I5, width
;
;       check if cloth too close to robot
        IF DY(HERE) < DY(limit.3) GOTO 12
        RETURN
    12  byte = 0
        TYPE "error in placement of cloth"
        RETURN
END

PROGRAM fine.adj
        y.total = 0
        tries = 0
```

```
    324  DELAY 0.5
         CALL inword
         y.error = word
         TYPE "Fine Adjustment : y error = ", /I6, y.error
         IF ABS(y.error) < 12 GOTO 325
         IF ABS(y.error) < 50 THEN
             y.error = y.error/2
         END
         IF y.error > 350 GOTO 326
         y.total = y.total+ABS(y.error)
                                   ; prevent smash into camera
         IF y.total > 1250 GOTO 326
         SPEED 3
         MOVES SHIFT(HERE BY 0, -(y.error/32), 0)
         BREAK
         tries = tries+1
         word = tries
         CALL outword
         DELAY 0.5
         GOTO 324
    325  word = 10
         CALL outword
         SPEED hi.speed  ALWAYS
         DELAY 0.5
         RETURN
    326  TYPE "excessive error at needle"
         HALT
END

PROGRAM go
         CALL main1
END

PROGRAM go.near.start
  ;    This routine moves robot from end of angle.adj routine
  ;        to the start position for the near sewing
         MOVES SHIFT(HERE BY 0, 0, 30)
         MOVES onway.5
         MOVES near.start
         BREAK
         SPEED 40  ALWAYS
  ;
  ;      if cloth uncovered, find edge
         wide.piece = TRUE
         IF SIG(-pcell1.on) GOTO 14
         IF SIG(-pcell2.on) GOTO 13
         wide.piece = FALSE
         REACTI -pcell2.on
         MOVES SHIFT(HERE BY 0, DY(limit.5)-DY(HERE), 0)
         BREAK
         IGNORE pcell2.on
         IF SIG(pcell2.on) GOTO 11
         IF (DY(limit.5)-DY(HERE)) < 30+fg.to.pc GOTO 11
         MOVES SHIFT(HERE BY 0, 15, 0)
         BREAK
  ;
  ;      move forward until edge found
     13  IF SIG(-pcell1.on) GOTO 14
```

CLOTHWORKERS' LIBRARY
UNIVERSITY OF LEEDS

```
        REACTI -pcell1.on
        MOVES SHIFT(HERE BY DX(limit.6)-DX(HERE))
        BREAK
        IGNORE pcell1.on
        IF SIG(pcell1.on) GOTO 11
;
;      move back outwards to place fingers near end of cloth
    14  MOVES SHIFT(HERE BY -pc.to.fg-20, 10-fg.to.pc, 0)
        SPEED 15
        MOVES SHIFT(DEST BY 0, 0, table.ht-DZ(DEST))
        BREAK
        SPEED hi.speed
        RETURN
    11  TYPE "error in finding near.start position"
        byte = 0
        RETURN
END

PROGRAM grip.transf
        TYPE /C1,"PROGRAM TO DEFINE TOOL TRANSFORMATION",/C1

        PROMPT "Revising previously defined tool (1 = yes)
                                                  ? ", answer

        IF answer <> 1 THEN
            TYPE "Move the mounting flange to the reference",
                                                              /S
            TYPE "location.", /S
            TYPE /C1, "Press ", /S
            TYPE "the COMP mode button on the teach pendant
                                                      when ", /S
            TYPE "ready to proceed."

            DETACH; Release the robot to the User

            WAIT (PENDANT(2) BAND ^17) <> 0
            WAIT (PENDANT(2) BAND ^20) <> 0
            ATTACH; Regain control of the robot

            TOOL NULL
            HERE ref.loc; Record the reference location
        END

        TYPE "Instal the new tool, move its tip back to the ", /S
        TYPE "reference location.", /C1, "Press the COMP mode ", /S
        TYPE "button on the teach pendant when ready."

        DETACH; Release the robot to the user

        WAIT (PENDANT(2) BAND ^17) <> 0
        WAIT (PENDANT(2) BAND ^20) <> 0

        ATTACH;Regain control
        TOOL NULL
        SET new.tool = INVERSE(HERE):ref.loc
        TOOL new.tool

        STOP
```

```
END

PROGRAM inch
            CALL inword
            SPEED 5
            MOVES SHIFT(HERE BY -word/32)
            BREAK
            RETURN
END

PROGRAM moveback
            CALL inword
            TYPE "IBM requests a move back of ", /I6, word
     ;      MOVES SHIFT(HERE BY word, 0, 0)
            BREAK              ; NB This routine has been
            RETURN             ; disabled.
END

PROGRAM remove
     ;
     ;   This routine withdraws from the needle zone carefully
     ;
            SPEED 25  ALWAYS
            MOVES SHIFT(HERE BY 0, 0, 12)
            MOVES outway.2
            BREAK
            SPEED hi.speed  ALWAYS
            RETURN
END

PROGRAM retreat
            DELAY 0.5
            SPEED 40  ALWAYS
            MOVES fin.1
            MOVES SHIFT(DEST BY 0, 0, 30)
            SPEED hi.speed
            MOVES fin.2
            SPEED 10
            MOVES SHIFT(DEST BY 0, 0, table.ht-DZ(DEST))
            SPEED 50  ALWAYS
            MOVES finish
            MOVES SHIFT(DEST BY 0, 0, 10)
            BREAK
            SPEED hi.speed
            MOVES start
            RETURN
END

PROGRAM rotate.90
     ;      This routine crumples cloth a bit and then rotates
     ;      the cloth by 90 degrees.
            SPEED 40  ALWAYS
     ;      IF wide.piece THEN
     ;      MOVES SHIFT(HERE BY 0, 0, test.level-DZ(HERE))
     ;      MOVES start.rotate
     ;      SPEED 18
     ;      MOVES SHIFT(DEST BY 0, 0, table.ht-DZ(DEST))
     ;      BREAK
```

```
  10 ;        END
  11 ;
  12          SPEED 40  ALWAYS
  13          MOVES SHIFT(HERE BY 0, 10, 0)
  14          BREAK
  15          angle.req = -90
  16          CALL rotate.ndle
  17 ;
  18 ;        lift robot up to testing level for photocells
  19          SPEED 9
  20          MOVES SHIFT(HERE BY 0, 0, test.level-DZ(HERE))
  21          BREAK
  22          SPEED hi.speed  ALWAYS
END

PROGRAM rotate.ndle
   1 ;    this routine rotates robot about needle a given angle
   2 ;   (angle.req). Locations required are at.ndle, front.ndle.
   3 ;
   4 ;        calculate distance between main finger and needle
   5          x.offst = DX(HERE)-DX(at.ndle)
   6          y.offst = DY(HERE)-DY(front.ndle)
   7          radius = SQRT(SQR(x.offst)+SQR(y.offst))
   8 ;
   9 ;        calculate locations and transformations
  10          SET pivot = SHIFT(HERE BY -x.offst, -y.offst, 0)
  11          SET temp = INVERSE(pivot):HERE
  12          angle.0 = ATAN2(DY(temp), DX(temp))
  13 ;
  14 ;        test required angle for size and direction
  15          IF angle.req > 90 THEN
  16              TYPE "angle.req is too large"
  17              HALT
  18          END
  19          IF angle.req > 0 THEN
  20              istep = 2
  21          ELSE
  22              istep = -2
  23          END
  24 ;
  25          SET temp = pivot:TRANS(radius*COS(angle.0),
                             radius*SIN(angle.0), 0, 90, -90, 0)
  26          z.offst = DZ(temp)-DZ(HERE)
  27          ang.offst = 90-angle.0
  28 ;
  29 ;
  30 ;        perform rotation
  31          FOR angle = angle.0 TO angle.0+angle.req STEP istep
  32              MOVE pivot:TRANS(radius*COS(angle),
                                radius*SIN(angle), z.offst,
                                        angle+ang.offst, -90, 0)
  33          END
  34          BREAK
  35          RETURN
END

PROGRAM set.param
   1          DO
```

```
            TYPE "Enter in parameters as follows:
                          (defaults given in brackets)"
            PROMPT "irow1   (2)  : ", word
            CALL outword
            PROMPT "irow2   (8)   : ", word
            CALL outword
            PROMPT "ipix1_offst (0) : ", word
            CALL outword
            PROMPT "ipix2_offst (0) : ", word
            CALL outword
            PROMPT "PIX_GAIN  (0.002)  :  ", tmp
            word = tmp*100000.
            CALL outword
            PROMPT "DERIV_GAIN  (0.1)  :  ", tmp
            word = tmp*10000
            CALL outword
            PROMPT "INT_FACT   (0.00003)   :  ", tmp
            word = tmp*1000000.
            CALL outword
            PROMPT "T_GAIN    (0.0015)  : ", tmp
            word = tmp*100000.
            CALL outword
            PROMPT "RQ_TENS     (70)   :  ", word
            CALL outword
            PROMPT "ACCEL_LIM (3)      :  ", tmp
            word = tmp*32
            CALL outword
            PROMPT "VEL_LIM (8)     : ", tmp
            word = tmp*32
            CALL outword
            PROMPT "Parameters set correctly ?  (Yes = 1)", word
            CALL outword
        UNTIL word == 1
        RETURN
END

PROGRAM startup.data
        f.r = SQRT(SQR(DX(fing1))+SQR(DY(fing1)))
        f.angle = ATAN2(DY(fing1), DX(fing1))
        nx = DX(at.ndle)
        ny = DY(front.ndle)
        TYPE "distance between fingers = ", /D, fing.dist
        TYPE "finger-flange radius     = ", /D, f.r
        TYPE "finger-flange angle      = ", /D, f.angle
        TYPE "needle position, x coord = ", /D, nx
        TYPE "needle position, y coord = ", /D, ny
        word = fing.dist*32
        CALL outword
        word = f.r*32
        CALL outword
        word = f.angle*180
        CALL outword
        word = nx*32
        CALL outword
        word = ny*32
        CALL outword
END
```

```
PROGRAM straighten
            SET temp1 = HERE
            SPEED 65  ALWAYS
  ;         MOVES blow.position
  ;         BREAK
  ;         OPENI
  ;         DELAY (0.5)
  ;         BREAK;
  ;         CLOSEI
            MOVES SHIFT(temp1 BY 30)
            BREAK
            TYPE "routine straighten has been called", /B
            SPEED hi.speed
            RETURN
END

PROGRAM uptoneedle
  ;
  ;    This routine pushes cloth up to needle and stays there
  ;    so that the presser foot can come down onto the cloth
  ;
            SPEED 70
            MOVES needle
            BREAK
            SPEED hi.speed  ALWAYS
            RETURN
END
```

APPENDIX F

CAMERA ROUTINES AND CALIBRATION PROGRAM

F.1. Camera Routines under AMX

F.1.1. Restart Procedure

```
void rpcamr()
{
        PMESS p;

        ajmodl();

        displ_init;
        prf__ "restart procedure for initialising cameras");
        end_print;
                                        /* set up I-SIGHT pointers */
        ajsseg(&cc_pt,(unsigned int)SEGMNT);
        ajsofs(&cc_pt,(unsigned int)0);
        cccb_pt = cc_pt + (int)CONTRLB;
        cam1_pt = cc_pt + (int)CAM1_OFS;
        cam2_pt = cc_pt + (int)CAM2_OFS;
        tp1_pt = cc_pt + (int)CAM1_FL;
        tp2_pt = cc_pt + (int)CAM2_FL;

        *cccb_pt = BUSFRZ;
        *(cc_pt+0x3f4) = 0;
        *(cc_pt+0x3f6) = 0;
        *cccb_pt = FREEZE;
}
```

F.1.2. Routines to Capture a Frame

```
void take_picture()
{
        int i;

        *cccb_pt = TRIGGER;         /* release FREEZE to trigger Z80  */
        for (i=0; i<400; i++)
                ;
        *cccb_pt = FREEZE;
}


void read_cam()
{
        z80_check();                    /* check that Z80 has finished */
        *cccb_pt = BUSFRZ;
        movmem(cam1_pt,cam1_buf,(unsigned int)NPIXLS);
```

```
        movmem(cam2_pt,cam2_buf,(unsigned int)NPIXLS);
        *cccb_pt = FREEZE;
}


void z80_check()
{
        char test;
        int i;

        i = 0;
        do
        {       i++;
                *cccb_pt = BUSFRZ;
                test = (*tp1_pt != 0 && *tp2_pt != 0);
                *cccb_pt = FREEZE;

                if (!test)
                        delay(1);
                if (i > 20)
                        crash(12345);
        } while (!test);
}
```

F.2. Camera Setup and Calibration Program

The following program was used to initialize the camera card, to set up the exposure levels, and to calibrate the camera mountings. The camera card had to be initialized each time that the IBM AT was powered up.

The program accepts one of the following runtime options :-

- -i   performs initialization
- -v   puts cameras' views on screen

In the default MODIFY mode, the exposure values can be set for each camera. On initialization, both cameras are set to an exposure value of 10.

```
#include        "c:\lc\stdio.h"
#include        "c:\lc\stdlib.h"

#define TRUE     1
#define FALSE    0
#define ESC      27                         /* ESC key on keyboard */
                                        /* I-SIGHT camera card addresses */
#define SEGMNT  0x9c00                      /* pcb card segment address  */
#define CONTRLB 0x3fff                      /* control byte address      */
#define INITZ80 0x02                        /* ctrl byte to initial. Z80 */
#define TRIGGER 0x00                        /* ctrl byte to trigger pict */
#define BUSRQ   0x01                        /* mask for bus request      */
#define FREEZE  0x08                        /* mask for freeze control   */
#define BUSFRZ  0x09                        /* mask for bus + freeze     */
```

```
#define CAM1_OFS0x000                      /* offset for camera # 1     */
#define CAM2_OFS0x400                      /* offset for camera # 2     */
#define MAX_PIX 0x3c0                      /* maximum no. pixels        */
#define CAM1_DT 0x3f0                      /* threshold & invert data   */
#define CAM2_DT 0x3f2
#define CAM1_FL 0x3f1                      /* flag for Z80 done signal  */
#define CAM2_FL 0x3f3
#define NROW    30                         /* no. rows of pixels        */
#define NCOL    32                         /* no. columns of pixels     */
#define NPIXLS  NROW*NCOL                  /* no. of pixels in picture  */
#define NCAM    2                          /* no. of cameras            */

#define L_SCREEN80                         /* graphics mode definitions */
#define A_MEM   0xb800
#define B_MEM   0xba00
#define BLOCK   0xff                       /* fill-in picture element   */
#define BLANK   0                          /* blank   picture element   */

#define MODIFY  0                          /* permit mod. of values     */
#define INITIAL 1                          /* initialise Z80 only       */
#define VIEWING 2                          /* display camera views      */

extern void alpha(), curs_xy(int,int);
extern void init(), set_cam(int), read_cam(), init_cc(),
          view(int), calib(int), delay(int), setup_cam(),
          display(int), set_screen(), z80_wait(), take_picture();

char thresh_b[NCAM] = {0};              /* threshold & invert ctrl byte */
short  int cam_ofs[NCAM] = {CAM1_OFS,CAM2_OFS};
short  int cam_dt[NCAM] = {CAM1_DT,CAM2_DT};
short  int cam_fl[NCAM] = {CAM1_FL,CAM2_FL};
char *cc_pt;                            /* pointer to base of cam. card */
char *cccb_pt;                             /* pointer to cc control byte*/
char cc_init = TRUE;                       /* flag for initializing ctrl*/
char cam1_buf[NPIXLS];
char cam2_buf[NPIXLS];
char *a_screen,*b_screen;
char opts[] = "";

void main(argc,argv)
int argc;
char *argv[];
{       char option, *odata;
        int next = 1, mode = MODIFY;

        odata = argopt(argc,argv, opts,&next,&option);
        if (odata == NULL)
                mode = MODIFY;
        else if (option == 'i')
                mode = INITIAL;
        else if (option == 'v')
                mode = VIEWING;

        init();
        switch  (mode)
        {   case  MODIFY :  if (ask_init())
                                   init_cc();
                            display(MODIFY);
```

```
                          break;
            case INITIAL :  init_cc();
                            setup_cam();
                            printf(
"\n****   I-SIGHT camera card initialisation completed    ****");
                            break;
            case VIEWING :  setup_cam();
                            display(VIEWING);
        }
}

void display(mode)
int mode;
{       int i = 0;
        do
        {   if (mode == MODIFY)
            {       set_cam(0);
                    set_cam(1);
            }
            take_picture();
            set_screen();
            do
            {       read_cam();
                    view(0);
                    view(1);
                    calib(0);
                    calib(1);
                     curs_xy((int)20,(int)17);
                    if ( *(cam2_buf+150) > 0x80 )
                        printf("Yes");
                    else
                        printf("No ");
                    curs_xy((int)30,(int)17);
                    printf("%7d",i++);
            } while ( !kbhit() );
        } while (getch() != ESC);
}

int ask_init()
{       char  c;

        printf("\n Initialize Z80 ? (Y/N) : ");
        c = getchar() ;
        getchar() ;
        if (c == 'Y' || c == 'y')
                return(TRUE);
        return(FALSE);
}

void init()
{       long int i ;
                                                        /* set up pointers */
        init_pt(&cc_pt,(unsigned int)SEGMNT,(unsigned int)0);
        cccb_pt = cc_pt+CONTRLB;

        init_pt(&a_screen,(unsigned int)A_MEM,(unsigned int)0);
        init_pt(&b_screen,(unsigned int)B_MEM,(unsigned int)0);
```

```
        *cccb_pt = FREEZE;
        *cccb_pt = BUSFRZ;
                                                        /* init thresholds   */
        for (i=CAM1_DT; i < CAM1_DT + 8; i+=2)
                *(cc_pt+i) = 0;
        *cccb_pt = FREEZE;                              /* FREEZE normally up*/
}

void set_screen()
{       alpha();                                /* set screen up for graphics*/
        curs_xy((int)0,(int)20);
        printf(" STRIP  rows :- ");
        curs_xy((int)0,(int)21);
        printf(" SEAM   cols :- ");
        curs_xy((int)0,(int)22);
        printf("        freq :- ");
        curs_xy((int)0,(int)23);
        printf(" SLOT  width :- ");
        curs_xy((int)0,(int)24);
        printf("        freq :- ");
        curs_xy((int)30,(int)18);
        printf("enter in ESC to exit");
}

            /* The Z80 must be initialized only once after power-up */
void init_cc()
{       long  int i,d;

        printf("\n Initialising the I-SIGHT camera card");
        *cccb_pt = INITZ80;                             /* Initializing  Z80 */
        for (i=0;i<1000;i++)                    /* delay while Z80 resets */
                d = i*4;
        *cccb_pt = FREEZE;
        for (i=0;i<1000;i++)                    /* delay while Z80 resets */
                d = i*4;
}

void set_cam(icam)              /* routine to set up & init. camera */
short int icam;
{       short int dummy, dum;
        char  answ, error;

        error = FALSE;
        do                                              /* enter in threshold*/
        {       printf(
     "\n enter in threshold value for camera #%d :",icam+1);
                scanf("%d",&dummy);
                dum = getchar();                        /* remove extra char */
                error = FALSE;
                if (dummy > 0x7f || dummy < 0)
                {   printf(
                    "\n illegal threshold value = %d",dummy);
                    error = TRUE;
                }
        } while (error);
        thresh_b[icam] = dummy;

        printf("\n invert image ? (Y/N) : ");
```

```
        scanf("%c",&answ);
        dum = getchar();
        if (answ == 'y' || answ == 'Y')
                thresh_b[icam] |= 0x80;

        *cccb_pt = BUSFRZ;                              /* request access  */
                                                      /* instal thresh val.*/
        *(cc_pt+cam_dt[icam]) = thresh_b[icam];
        *cccb_pt = FREEZE;                            /* release Z80 bus    */
}

void setup_cam()
{       int i;

        *cccb_pt = BUSFRZ;
        *(cc_pt+cam_dt[0]) = (char)10;
        *(cc_pt+cam_dt[1]) = (char)10;
        *cccb_pt = FREEZE;
        take_picture();
        for (i=0; i < 3; i++)
        {       delay(100);
                read_cam();
        }
}

void read_cam()
{
        z80_wait();                         /* wait until picture taken  */
        *cccb_pt = BUSFRZ;                    /* transfer data to buffer */
        movmem(cc_pt+cam_ofs[0],cam1_buf,(unsigned)NPIXLS);
        movmem(cc_pt+cam_ofs[1],cam2_buf,(unsigned)NPIXLS);
        *cccb_pt = FREEZE;
        take_picture();
}

void take_picture()
{       *cccb_pt = TRIGGER;             /* release FREEZE to trigger Z80*/
        *cccb_pt = FREEZE;                              /* reset FREEZE  */
}

void z80_wait()                          /* routine to check Z80 flag */
{       short int i,j;
        char flag1, flag2, *tp1_pt, *tp2_pt;

        tp1_pt = cc_pt + cam_fl[0];                   /* pointer to flag */
        tp2_pt = cc_pt + cam_fl[1];

        for(j=0; j < 1000; j++)
        {                                  /* delay for Z80 proccessing */
                for(i=0; i< 100 ;i++)
                        ;

                *cccb_pt = BUSFRZ;
                flag1 = *tp1_pt;                        /* read flag byte */
                flag2 = *tp2_pt;
                *cccb_pt = FREEZE;

                if (flag1 != 0 && flag2 != 0)                /* test flag */
```

```
                        return;
        }
        printf("\n excessive waiting for Z80");
}


        /*  This routine displays the camera picture on the screen. */
        /*  Since  screen pixels are rectangular each row of camera */
        /*  pixels is repeated 4 times. */

void view(icam)
short int icam;
{       int ofs_1, ofs_2, start, n, m, ipix;
        char *cam_buf;

        cam_buf = icam ? cam2_buf : cam1_buf;
        start = icam ? 208 : 162;

        for ( n = ipix = 0; n < NROW; n++)
        {       ofs_1 = (2*n*L_SCREEN) + start;
                ofs_2 = ((2*n + 1) * L_SCREEN) + start;

                for (m=0;m < NCOL; m++,ofs_1++,ofs_2++,ipix++)
                {       if ( *(cam_buf+ipix) > 0x80 )
                        {       *(a_screen+ofs_1) = BLOCK;
                                *(a_screen+ofs_2) = BLOCK;
                                *(b_screen+ofs_1) = BLOCK;
                                *(b_screen+ofs_2) = BLOCK;
                        }
                        else
                        {       *(a_screen+ofs_1) = BLANK;
                                *(b_screen+ofs_1) = BLANK;
                                *(a_screen+ofs_2) = BLANK;
                                *(b_screen+ofs_2) = BLANK;
                        }
                }
        }
}


edge_find(cam_buf,irow,icol2)
char *cam_buf;
int irow, *icol2;                   /* no. of pixel row to be searched */
{
        int ipix, icol, icol1, phase;
        ipix = (irow * NCOL);               /* N.B. irow starts at zero  */
        phase = 1;
        *icol2 = 0;
        for (icol = 0; icol < NCOL; icol++,ipix++)
        {                             /* phase 1 - search for black edge */
             if (phase == 1)
             {
                   if ( *(cam_buf+ipix) < 0x80 )
                   {   phase = 2;
                       icol1 = icol;
                   }
             }
             else                                /* phase 2 - search for */
```

```
                {                                   /* following white edge */
                        if ( *(cam_buf+ipix) > 0x80 )
                        {       *icol2 = icol;
                                return(icol1);
                        }
                }
        }
        if (phase == 2)
                return(icol1);
        return(NCOL-1);
}

void calib(icam)
int icam;
{
        int start,irow,icol,i,ii,slot[NCOL], strip[NROW], istrip,
            seam[NCOL], icol2;
        char *cam_buf;
                                                /* reinitialise on entry */
        for (i = 0; i < NCOL; i++)
        {       slot[i] = 0;
                seam[i] = 0;
        }

        cam_buf = icam ? cam2_buf : cam1_buf;
        start = icam ? 44 : 18;

        for (istrip = irow = 0; irow < NROW; irow++)
        {
                icol = edge_find(cam_buf,irow, &icol2);
                if (icol > NCOL-2)
                        strip[istrip++] = irow;
                else
                        seam[icol]++;
                if (icol2 > icol)
                        slot[icol2-icol]++;
        }
        curs_xy(start,(int)20);
        for (i = 0; i < 7; i++)
                if (i < istrip)
                        printf("%3d",strip[i]);
                else
                        printf("   ");
        curs_xy(start,(int)21);
        for (i = 0, ii = 0; i < NCOL; i++)
                if (seam[i] > 3)
                {       ii++;
                        printf("%3d",i);
                }
        if (ii < 7)
                for (i = 0; i <= 7-ii; i++)
                        printf("   ");
        curs_xy(start,(int)22);
        for (i = 0, ii = 0; i < NCOL; i++)
                if (seam[i] > 3)
                {       ii++;
                        printf("%3d",seam[i]);
                }
```

```
        if (ii < 7)
                for (i = 0; i <= 7-ii; i++)
                        printf("   ");
        curs_xy(start,(int)23);
        for (i = 0, ii = 0; i < NCOL; i++)
                if (slot[i] > 2)
                {       ii++;
                        printf("%3d",i);
                }
        if (ii < 7)
                for (i = 0; i <= 7-ii; i++)
                        printf("   ");
        curs_xy(start,(int)24);
        for (i = 0, ii = 0; i < NCOL; i++)
                if (slot[i] > 2)
                {       ii++;
                        printf("%3d",slot[i]);
                }
        if (ii < 7)
                for (i = 0; i <= 7-ii; i++)
                        printf("   ");
}

void delay(times)
int times;
{
        int i,j;

        for (i=0; i < times; i++)
                for (j=0; j < 500; j++)
                        ;
}


        TITLE   ASSEMBLER ROUTINES FOR CAMERA PROGRAMS
        NAME    CAM_SUP
        INCLUDE LM8086.MAC

X       EQU     6               ; offset of arguments for L model

        PSEG                            ; code segment begins
;
;               module entry points
;
        PUBLIC  ALPHA
        PUBLIC  CURS_XY
        PUBLIC  INIT_PT
;
;               define stack structure for parameter access
;
CFSS    STRUC
;
CFBP    DW      ?
CFRA    DD      ?
CFPA    DW      ?
CFPB    DW      ?
CFPC    DW      ?
```

```
CFPD    DW      ?
;
CFSS    ENDS
;
ALPHA   PROC    FAR

        push bp
        mov bp,sp

        mov ah,0                  ; set 640 x 200 bw graphics mode
        mov al,6
        int 16

        mov dl,9            ; column postion of cursor
        call prcam          ; subroutine to print camera title
        mov al,49           ; camera number
        call dis            ; subroutine to display a character
        mov dl,61           ; column position of cursor
        call prcam
        mov al,50           ; camera number
        call dis

        pop bp
        ret
ALPHA   ENDP
                             ; subroutine to display camera title
prcam:  call cursor
        mov al,67
        call dis
        mov al,65
        call dis
        mov al,77
        call dis
        mov al,69
        call dis
        mov al,82
        call dis
        mov al,65
        call dis
        mov al,32
        call dis
        ret

                           ; subroutine to display a character
dis:    mov ah,10          ; write char at current cursor postion
        mov cx,1           ; count of characters to write
        int 16             ; video_IO BIOS routine
        call cursor
        ret
                           ; subroutine to increment cursor positn
cursor: mov ah,2
        mov dh,17          ; row postion of cursor
        add dl,1           ; increment column postion
        mov bh,0           ; page no.- must be 0 for graphics mode
        int 16
        ret
```

CLOTHWORKERS' LI
UNIVERSITY OF LE

```
;
;       name :   curs_xy(icol, irow)
;
;                char icol, irow;
;
;    purpose :   put screen cursor at specified column and row

;
CURS_XY PROC    FAR
;
        push bp                     ; save base pointer on stack
        mov bp,sp              ; base points to stack for parameters

        mov bx, [bp].CFPA           ; 1st parameter: column no
        mov dl, bl
        mov bx, [bp].CFPB           ; 2nd parameter: row no
        mov dh, bl

        mov ah,2                    ; select 'set cursor' function
        mov bh,0              ; page no.- must be 0 for graphics mode
        int 16

        pop bp                      ; replace old base pointer
        ret
;
CURS_XY ENDP
;

;
;       name :          init_pt(ptr, segment, offset)
;
;                       unsigned int segment, offset;
;                       char **ptr;
;
;       purpose :       set up pointer segment and offset

;
INIT_PT PROC    FAR
;
        push bp         ; save base pointer on stack
        mov bp,sp       ; base pointing to stack for parameters
        push es
;
        les bx,DWORD PTR [bp].CFPA      ; #1 - pointer to pointer
        mov ax,[bp].CFPC                ; #2 - segment
        mov es:[bx+2],ax                ; move segment
;
        mov ax,[bp].CFPD                ; 3rd parameter: offset
        mov es:[bx],ax                  ; move offset
;
        pop es
        pop bp                          ; replace old base pointer
        ret
;
INIT_PT ENDP
;
        ENDPS
        END
```

APPENDIX G

SIMULATION PROGRAM

```
PROGRAM simulate(input,output);
{
      This program simulates robotic sewing of a curved
        cloth contour using a visually servoed robot.
                                                              }

CONST
     npixels = 31 ;
     pixel_width = 0.5;
     cloth_length = 190;          {  dist. to sew in mm.            }
     timelimit   = 50 ;           {  time limit for program         }
     seam_width  = 13.0;          {  seam width request             }
     del_t     = 0.14 ;           {  servo loop time interval       }
     cam2_dist   = 23.0;          {  dist. between 2 cameras        }

     x_offset    = 0 ;            {  graphic output parameters      }
     y_offset    = 0 ;
     u_offset    = 0 ;
     v_offset    = 0 ;
     screenlimit = 199 ;
     scalefactor = 1.0 ;
     display_on  = TRUE ;
     printout_on = FALSE  ;
     curved_seam = TRUE ;

TYPE
     coord  =  RECORD
                     u : real;
                     v : real
               END ;

     regs   =  RECORD
                     ax,bx,cx,dx,bp,si,di,ds,es,flags : integer ;
               END;

     timestr = string[8];
     datestr = string[10];

VAR
    pixel, gain : ARRAY[0..npixels] OF real ;
    upper_pix, lower_pix  : integer ;
    pix_gain_fact,deriv_gain,alpha_init,
    error_init,rtn,cloth_feed,limit_total,
    accel limit, vel limit, prop gain   :  real;
    excessive : boolean;

PROCEDURE  InitData;
BEGIN
     prop_gain  := 0.07;
     deriv_gain := 1.6;
     alpha_init := 0.4;
     error_init := 0.50;
```

```
     rtn         := 300.;
     cloth_feed := 60.;
     limit_total := 200.;
     accel_limit := 3.0;
     vel_limit   := 8.0;
END;

PROCEDURE InputData;
BEGIN
  gotoxy(65,1);  write('prop, K1   ',prop gain:6:4);
  gotoxy(75,1);  read(prop_gain);          gotoxy(75,1);
  write(prop_gain:6:4);
  gotoxy(65,2);  write('deriv, K2  ',deriv_gain:5:2);
  gotoxy(76,2);  read(deriv_gain);         gotoxy(76,2);
  write(deriv_gain:5:2);
  gotoxy(65,3);  write('init alpha ',alpha_init:5:1);
  gotoxy(76,3);  read(alpha_init);         gotoxy(76,3);
  write(alpha init:5:1);
  gotoxy(65,4);  write('initial E  ',error_init:5:1);
  gotoxy(76,4);  read(error_init);         gotoxy(76,4);
  write(error_init:5:1);
  gotoxy(65,5);  write('dist, Xf   ',rtn:5:0);
  gotoxy(76,5);  read(rtn);                gotoxy(76,5);
  write(rtn:5:0);
  gotoxy(65,6);  write('max Yf     ',limit_total:5:0);
  gotoxy(76,6);  read(limit_total);        gotoxy(76,6);
  write(limit_total:5:0);
  gotoxy(65,7);  write('speed, Vc  ',cloth_feed:5:0);
  gotoxy(76,7);  read(cloth_feed);         gotoxy(76,7);
  write(cloth_feed:5:0);
  gotoxy(65,8);  write('max acceln ',accel_limit:5:1);
  gotoxy(76,8);  read(accel_limit);        gotoxy(76,8);
  write(accel limit:5:1);
  gotoxy(65,9);  write('max velcty ',vel_limit:5:1);
  gotoxy(76,9);  read(vel_limit);          gotoxy(76,9);
  write(vel_limit:5:1);
  gotoxy(65,10); write('no pixels  ',npixels:5);
  gotoxy(65,11); write('pix width  ',pixel_width:5:2);
  gotoxy(65,12); write('time step  ',del_t:5:3);
  gotoxy(65,13); write('dist, Xcam ',cam2_dist:5:1);
  gotoxy(65,14); write('seam width ',seam_width:5:1);
END;

PROCEDURE  draw line ;
  VAR
       seam_off1, seam_off2  : integer ;
  BEGIN
      seam_off1 := round(seam_width * 0.7071 * scalefactor) ;
      seam_off2 := round(seam_width / 0.7071 * scalefactor) ;
      draw (x_offset-seam_off1, y_offset+seam_off1,
      x_offset+screenlimit-seam_off2, y_offset+screenlimit, 3) ;
 END ;

PROCEDURE  draw_curve ;
    VAR
        temp, temp2, temp3  : real ;
        seam_x, seam_y, x1, x, y, x_o, x_n  :  integer ;
    BEGIN
```

```
        x_o := x_offset ;
        x_n := x_o + round(sqrt(200.0 * screenlimit) ) ;
        FOR  x1 := x_o  TO x_n  DO
          BEGIN
              x := x1 - x_offset ;
              temp := x ;
              temp2 := sqr(temp) ;
              y := round(temp2/200.0) ;
              temp3 := (seam_width*100.0)/(sqrt(temp2 + 10000.0)) ;
              seam_y := round(y + temp3) ;
              seam_x := round(x1 - (x * temp3 / 100.0) ) ;

              plot(x1, y, 3) ;
              plot(seam_x, seam_y,3) ;
          END ;
     END ;

PROCEDURE setup_screen ;
 BEGIN
   graphmode ;
   graphbackground(0) ;
                                  {  draw x and y axes  }
   draw (x_offset, y_offset, x_offset, y_offset+screenlimit, 3) ;
   draw (x_offset, y_offset, x_offset+screenlimit, y_offset, 3) ;

   IF  curved_seam  THEN  draw_curve
   ELSE  draw_line ;
 END ;


PROCEDURE setup_pixels ;
VAR
    nspaces,gain_sign,centre_pix,i,factor,gain_switch : integer;
    half_pix_gain  :  real ;

BEGIN
  pix_gain_fact := prop_gain*pixel_width;
  nspaces    := npixels + 1;
  centre_pix := nspaces DIV 2 ;
  factor     := 1 - centre_pix ;
  gain_switch := 0 ;
  gain_sign  := -1;
  half_pix_gain := (pixel_width / 2.0) * pix_gain_fact ;

  {   IF printout_on  THEN
  BEGIN
      writeln(lst,'                    Pixel arrangement') ;
      writeln(lst) ;  writeln(lst,
          ' pixel no.    factor        spacing         gain');
      writeln(lst) ;
  END ;                                                    }

  FOR  i := 0  TO  npixels -1   DO
  BEGIN
      pixel[i] := pixel_width * factor ;
      gain[i] := (pixel[i - gain_switch] * pix_gain_fact) +
                               (half_pix_gain * gain_sign) ;
      gain[i] := - gain[i] ;
```

```
{    IF printout_on  AND (gain_sign = -1) THEN
     BEGIN
          writeln(lst,'                                        ',
                              '                  ',gain[i]:10);
          writeln(lst,'   ',i:4,'    ',factor:8,'            ',
                              pixel[i]:10) ;
     END ;                                                  }

     IF  factor = 0  THEN  gain_switch := 1 ;
     IF  factor = 0  THEN  gain_sign   := 1 ;
     factor := factor + 1 ;
  END;
  gain[i+1] := (pixel[i] * pix_gain_fact) +
                                   (half_pix_gain * gain_sign) ;
  gain[i+1] := - gain[i+1] ;

  IF printout_on  THEN
  BEGIN
       writeln(lst) ;
       writeln(lst) ;
  END ;
END ;

FUNCTION  limit(qty, lim : real) : real;
       BEGIN
            IF (qty > lim) THEN
                limit := lim
            ELSE
            BEGIN
                 lim := lim * -1;
                 IF (qty < lim)  THEN
                     limit :=  lim
                 ELSE
                     limit := qty;
            END;
       END;


FUNCTION  np_measured (ndle,pos : coord; cosalpha : real) : real;
  VAR
       i  : integer ;
       half_pix_width,np_calculated,error_calc,error_meas : real;
  BEGIN
       i := 0 ;
       half pix width := pixel width / 2.0 ;
       np_calculated := (pos.u - ndle.u)/cosalpha ;
       error_calc := np_calculated - seam_width ;

       WHILE  (error_calc > pixel[i]) AND (i < npixels)
           DO   i := i + 1 ;

       IF  i = npixels   THEN
           error_meas := pixel[npixels-1] + half_pix_width
       ELSE
           error_meas := pixel[i] - half_pix_width ;

       np_measured := error_meas + seam_width ;
```

```
  END ;


FUNCTION calc_error (ndle,pos:coord; cosalpha,beta:real):real;
    VAR
          np : real ;
    BEGIN
          np := np_measured (ndle, pos, cosalpha) ;
          calc_error := (np * cos(beta)) - seam_width ;
    END ;


PROCEDURE rotate (tanalpha : real;ndle : coord; VAR pos : coord);
    VAR
         temp1, temp2, temp3  :  real ;
    BEGIN
         IF  curved_seam THEN
         BEGIN
             temp1 := 100.0 * sqr(tanalpha) ;
             temp2 := 2.0 * (ndle.v + (ndle.u*tanalpha)) ;
             temp3 := 100.0 * tanalpha ;
             IF (temp1 + temp2 < 0) THEN
                   pos.u := 0
             ELSE
                   pos.u := 10.0 * sqrt(temp1 + temp2) - temp3 ;
             pos.v := sqr(pos.u)/200.0 ;
         END
         ELSE
         BEGIN
             pos.v := (ndle.v + (ndle.u*tanalpha))/(1 + tanalpha);
             pos.u := pos.v;
         END ;
     END ;


 PROCEDURE translate(dist,alpha,cosalpha : real;ndle1 : coord;
                                                 VAR ndle2,pos : coord);

     VAR
          sinalpha : real;

     BEGIN
          sinalpha := sin(alpha) ;
          ndle2.u := ndle1.u - (dist * sinalpha) ;
          ndle2.v := ndle1.v - (dist * cosalpha) ;
          rotate ( (sinalpha/cosalpha), ndle2, pos )
     END ;


  FUNCTION   transferfunctn (error, beta : real) : real ;
     VAR
         i : integer ;
         transfer : real ;

     BEGIN
         i := 0 ;
         WHILE  ( error > pixel[i] ) AND ( i < npixels )
         DO        i := i + 1 ;
```

```
        transfer := gain[i] + (deriv_gain * beta) ;
        transferfunctn :=  transfer ;
     END ;

PROCEDURE  initial_pos (VAR pos, ndle : coord ) ;
VAR
     tanalpha, sinalpha, cosalpha : real ;

BEGIN
   sinalpha := sin(alpha init) ;
   cosalpha := cos(alpha_init) ;
   tanalpha := sinalpha/cosalpha ;

   IF curved_seam THEN
   BEGIN
        ndle.v := 199.0 ;
        pos.v := ndle.v - ((seam_width + error_init)*sinalpha);
        pos.u := sqrt( 200.0 * pos.v ) ;
        ndle.u := pos.u - ((ndle.v - pos.v) / tanalpha ) ;
   END
   ELSE
   BEGIN
        ndle.v := cloth_length ;          {arbitrary  needle postn}
        ndle.u := ndle.v - ((seam_width+error_init) *
                                    (sinalpha  + cosalpha)) ;
        pos.u  := (ndle.v + ndle.u*tanalpha)/(tanalpha + 1) ;
        pos.v   := pos.u ;
   END ;
END ;

PROCEDURE lineplot ( n, p  : coord ) ;
   VAR
      nu_i, nv_i, pu_i, pv_i : integer ;
   BEGIN
      nu_i := round(((n.u-u_offset) * scalefactor) + x_offset) ;
      nv_i := round(((n.v-v_offset) * scalefactor) + y_offset) ;
      pu_i := round(((p.u-u_offset) * scalefactor) + x_offset) ;
      pv_i := round(((p.v-v_offset) * scalefactor) + y_offset) ;

      draw (nu i,nv i,pu i,pv_i,3) ;
   END ;

 PROCEDURE curve_plot (n, p : coord) ;
     VAR
            nu_i, nv_i, pu_i, pv_i : integer ;
     BEGIN
            nu_i := round(n.u + x_offset ) ;
            nv_i := round(n.v) ;
            pu_i := round(p.u + x_offset) ;
            pv_i := round(p.v) ;

            draw (nu_i, nv_i, pu_i, pv_i, 3) ;
     END ;

 FUNCTION  time : timestr;
     VAR
            regpack        : regs;
            hour, min, sec : string[2];
```

```
      BEGIN
            WITH regpack DO
                 ax := $2c shl 8;
            MSDOS(regpack);
            WITH regpack DO
            BEGIN
                 str(cx shr 8, hour);
                 str(cx mod 256, min);
                 str(dx shr 8, sec);
            END;
            time := hour+':'+min+':'+sec;
      END;


FUNCTION  date  : datestr;
    VAR
            regpack          : regs;
            month, day       : string[2];
            year             : string[4];

    BEGIN
            WITH regpack DO
                  ax := $2a shl 8;
            MSDOS(regpack);
            WITH regpack DO
            BEGIN
                 str(cx, year);
                 str(dx mod 256, day);
                 str(dx shr 8, month);
            END;
            date := day+'/'+month+'/'+year;
     END;


PROCEDURE  print_heading ;
  BEGIN
    writeln(lst,#12,'                                                    ',
    '                           ',  date);
    writeln(lst,    '                                                    ',
    '                           ',  time,#10);
    writeln(lst,#27#69,'                   Simulation of Robotic ',
                             'Sewing of Curved Cloth',#10) ;
    write(lst,#27#70,'            version 1.8 :   cloth',
           ' contour ') ;
    IF  curved_seam THEN
            write (lst,'CURVED seam  v = sqr(u)/200')
    ELSE
            write (lst,'STRAIGHT seam   u=v') ;
    writeln (lst) ;
    IF  cam2_dist = 0  THEN
       writeln(lst,'                       one   camera   only')
    ELSE
       writeln(lst,'                       forward feedback',
                                ' from 2nd camera');
    writeln(lst,'                           Acceleration limiting');
    writeln(lst) ;
    writeln(lst,'                                    Input Data') ;
    writeln (lst) ;
```

```
   write(lst,'  no. pixels            = ',npixels:4) ;
   writeln(lst,'     derivative gain       = ',deriv_gain:8:4);
   write(lst,'  seam width            = ',seam_width:4) ;
   writeln(lst,'      proportional gain    = ',prop_gain:8:4);
   write(lst,'  feed speed            = ',cloth_feed:4) ;
   writeln(lst,'       servo loop time delay = ',del_t:8:4) ;
   write(lst,'  initial error         = ',error_init:4) ;
   writeln(lst,'     initial angle         = ',alpha_init:8:4);
   write(lst,'  cloth length          = ',cloth_length:4) ;
   writeln(lst,'     total limit           = ',limit_total:8:4);
   write(lst,'  inter camera distance = ',cam2_dist:4) ;
   writeln(lst,'     inter pixel distance  = ',pixel_width:8:4);
   writeln(lst,'  acceleration limit    = ',accel_limit:4);
   writeln(lst) ;
 END ;

PROCEDURE  print_table ;
 BEGIN

     writeln(lst,'                                  Simulation ',
                                      '     Results') ;

     writeln(lst) ;
     writeln(lst,' error         alpha       np         beta     ',
     ' gain       y_sc      y_displ');

{    writeln(lst,'           ndle.u          ndle.v          ',
                          ' pos.u                 pos.v') ;
                                                                 }

     writeln(lst) ;
 END ;


FUNCTION calc_beta(ndle1,pos1:coord;alpha,cosalpha:real):real;

{  This function returns the locally measured }
{  angle between cloth & sew m/c }

    VAR
       ndle2, pos2 : coord ;
       np_1, np_2 : real ;

    BEGIN
         translate (cam2_dist,alpha,cosalpha,ndle1,ndle2,pos2);
         np_1 := np_measured(ndle1,pos1,cosalpha) ;
         np_2 := np_measured(ndle2,pos2,cosalpha) ;

         calc_beta := arctan((np_1 - np_2)/cam2_dist) ;
    END;

PROCEDURE performance(error,dist : real);
   BEGIN
       IF excessive THEN exit;
       IF abs(error) < 1.0 THEN exit;
       gotoxy(68,16); write('P.I. = ',dist:5:2);
       excessive := TRUE;
   END;
```

```
                        { MAIN PROGRAM    }
VAR
   error, sew_dist, alpha, total_time, next_error, y_sc, y_displ,
     cosalpha, tanalpha, del_alpha, del_dist, yd_old, y_offst,
     dedt, np_old, acc lim, vel lim, old y, np, beta : real ;
   ndle1, ndle2, pos1, pos2, pos3 : coord ;
   result1 : regs;
   dummy : char;
   cloth_end : boolean;

BEGIN
   InitData;
   REPEAT  BEGIN
   IF display_on  THEN setup_screen ;
   InputData;
                                                { initialisations   }
   initial_pos (pos1,ndle1) ;
   sew_dist := 0;
   alpha   := alpha_init ;
   cosalpha := cos(alpha) ;
   tanalpha := sin(alpha)/cosalpha;
   np := np measured(ndle1,pos1,cosalpha) ;
   del_dist := cloth_feed*del_t;                { incr. feed distance }
   total_time := 0 ;
   excessive := FALSE;
   cloth_end := FALSE;
   y_sc := 0;    y_displ := 0;   yd_old := 0;   old_y :=  0;
   y_offst := tanalpha * rtn;

   { convert robot motion limits from handshakes to del_t units }
   vel_lim := vel_limit*del_t/0.028;
   acc_lim := accel_limit*del_t*del_t/0.028/0.028;
   IF printout_on THEN print_heading ;
   setup_pixels ;
   IF printout_on THEN print_table ;
   IF display_on  THEN
   BEGIN
        IF curved seam  THEN  curve_plot(ndle1,pos1)
        ELSE   lineplot(ndle1,pos1) ;
   END ;

   REPEAT
      np_old := np;
      np := np_measured(ndle1,pos1,cosalpha) ;

      IF  cam2_dist = 0  THEN
           beta := arctan((np_old - np)/del_dist)
      ELSE
           beta := calc_beta(ndle1, pos1, alpha,cosalpha);

      error := calc_error (ndle1,pos1,cosalpha,beta) ;
      del_alpha := transferfunctn (error, beta) ;

      translate (del_dist, alpha, cosalpha, ndle1, ndle2, pos2) ;
                                   { update alpha }
      alpha := alpha + del_alpha;
```

```
    cosalpha := cos(alpha) ;
    tanalpha := sin(alpha)/cosalpha;
                    {  calculate robot displ in mm & limit it }
    y_sc := tanalpha * rtn - y_offst;
    y_displ := y_sc - old_y;
    y_displ := limit(y_displ,vel_lim);
    y_displ := limit(y_displ-yd_old,acc_lim) + yd_old;
    y_sc := limit(old_y+y_displ,limit_total);
    y_displ := y_sc - old_y;
    yd_old := y_displ;
    old_y := y_sc;
    tanalpha := (y_sc + y_offst)/rtn;
    alpha := arctan(tanalpha);
    cosalpha := cos(alpha) ;

    rotate ( tanalpha, ndle2, pos3) ;
                                                   {  update parameters   }
    sew_dist := sew_dist + del_dist ;
    ndle1.u := ndle2.u ;
    ndle1.v := ndle2.v ;
    pos1.u  := pos3.u ;
    pos1.v  := pos3.v ;
    total_time := total_time + del_t ;

    performance(error,sew_dist);
    IF (pos1.u < 0) or (pos1.v < 0) or
         (ndle1.u < 0) or (ndle1.v < 0)
       THEN  cloth_end := TRUE;
    IF alpha < 0 THEN cloth_end := TRUE;

    IF printout_on and not cloth_end  THEN
    BEGIN
       writeln(lst, error:6:2,'    ',alpha:6:3,'   ', np:6:2,
               '     ',beta:6:3,'    ',del_alpha:6:3, '   ',
                 y_sc:6:1,'   ',y_displ:6:1) ;
   {   writeln(lst,'         ',ndle1.u:10,'         ', ndle1.v:10,
                 '      ',pos1.u:10,'        ', pos1.v:10);  }
    END ;

    IF (display_on and (not cloth_end)) THEN
    BEGIN
         IF curved_seam  THEN  curve_plot(ndle1,pos1)
         ELSE   lineplot(ndle1,pos1) ;
    END ;

  UNTIL (total_time > timelimit) or cloth_end;

  gotoxy(68,18); write('final = ',sew_dist:5:2);

  IF (display_on) AND (printout_on)  THEN
  BEGIN
       print_heading ;
       writeln(lst,#10#10#10);
       intr(5,result1);
  END;
  readln(dummy);
  END;  UNTIL NOT curved_seam;
END.
```

APPENDIX H

INTERFACE CIRCUITS

H.1. IBM AT Interface Card

In addition to the RS232C serial ports which were required for the ALTER and Uplink facilities, several other interfaces were necessary between the IBM AT and other components of the FIGARO system. These interfaces were implemented on an IBM AT prototype card.

H.1.1. General Purpose Ports

Three 373 tri-state latches and two 8255 PIO controllers were installed on the card. Two of the 373 latches were configured as output ports, and are referred to as PORTA and PORTB in the software. The third latch, PORTC, was configured for input.

The PIO controllers provided 6 ports, PORTE through to PORTJ, which could be configured under program control. The address of the control port of each PIO is listed in the header file, under CB IO 1 and CB IO 2.

H.1.2. Sewing Machine Interface

An AD558JN DAC was incorporated on the card, and configured to provide an analog output of 0 to 10 VDC. The address of the DAC was referred to as SPEED_P. The DAC's output was connected to the sewing machine's speed control pin.

The interfacing of the sewing machine's functions to the IBM AT is described in table H-1. The lines to the sewing machine were buffered to accommodate the higher CMOS voltages in the sewing machine controller.

H.1.3. Counter for Encoder Signal

The shaft encoder signal was connected to a counter circuit which is shown in fig H-1. The two 373 latches were referred to as LO_COUNT and HI_COUNT in the header file. The MASTER RESET and the ENABLE LATCHES lines were taken from pins 1 and 2 of PORTJ (or CB_COUNTR).

PORT & pin no.	Address	Buffer	Description
A 1	772 lo	7406	middle speed
2		"	thread trimming
3	(0x304)	"	needle up
4	output	"	compensation
5		"	low speed
6		"	high speed
7		"	presser foot
8	hi	"	back tack
B 1	773	7406	needle up stop
2	(0x305)	"	needle down stop
C 1	774	4049	needle up signal
3	(0x306)	"	encoder signal

Table H-1: Interface to Sewing Machine Functions

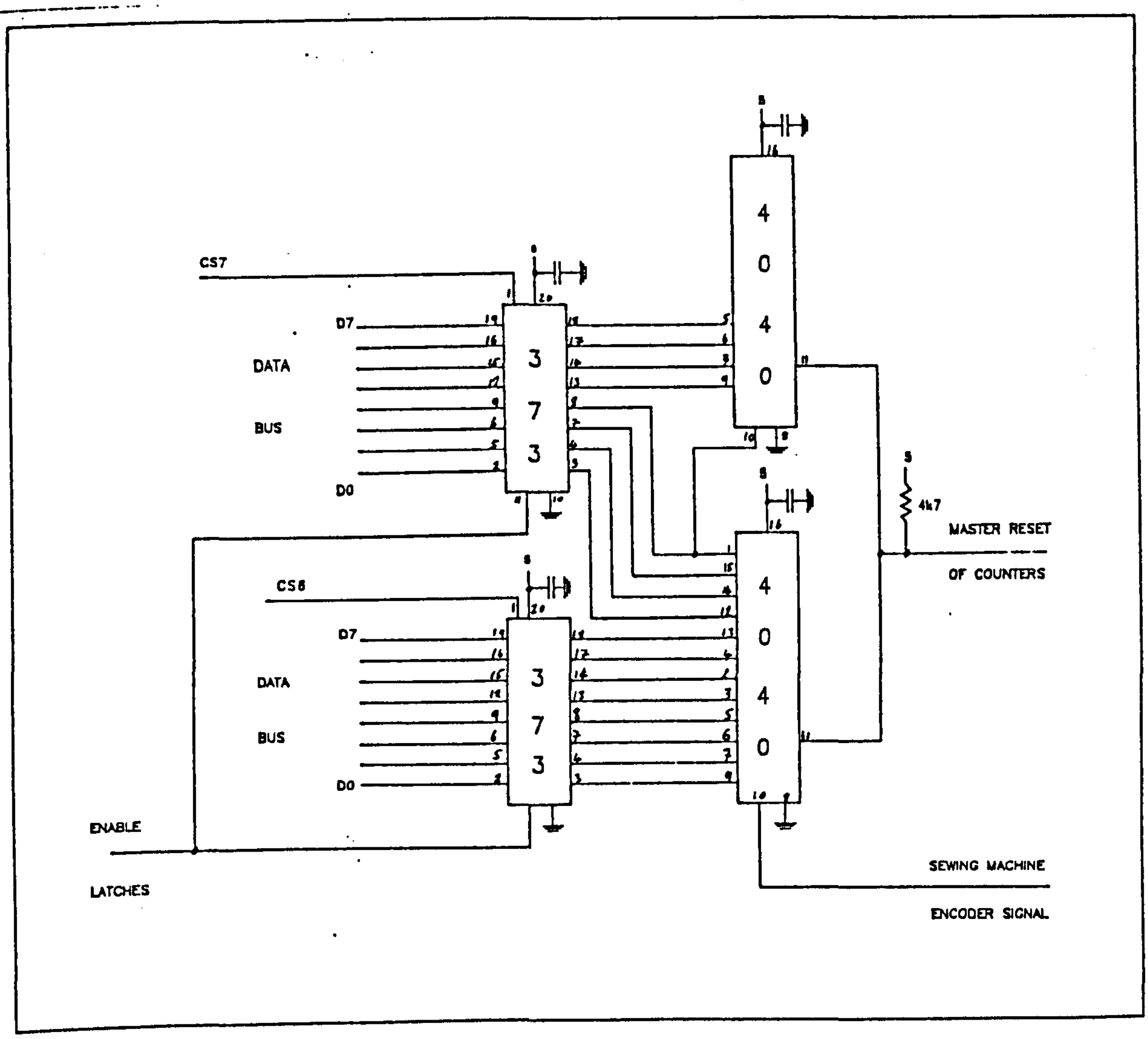


Fig. H-1: Counter Circuit for Shaft Encoder Signal

H.1.4. GPC Interface

The implementation of the GPC link on the IBM AT is detailed in table H-2. (PUR is an abbreviation for pull-up resistor).

PORT & pin no.	Address	Buffer/ interfc	Destin Address	Description
E 1	776	7406	WX9G	Output Data bus
2	(0x308)	"	WX10G	to Unimation
3		"	WX11G	
4		"	WX12G	
5		"	WX13G	
6		"	WX14G	
7		"	WX15G	
8		"	WX16G	
F 1	777	PUR	0X9S	Input Data Bus
2	(0x309)	"	0X10S	from Unimation
3		"	0X11S	
4		"	0X12S	
5		"	0X13S	
6		"	0X14S	
7		"	0X15S	
8		"	0X16S	
G 1	778		IRQ3	interrupt - input
2	(0x30A)	7406	WX7G	INPUT BUFF FULL
3		PUR	0X7G	STROBE - input
4			IRQ5	interrupt - output
5		PUR	0X6S	CONTROL SIGNAL in
6				
7		PUR	0X8S	ACKNOWLEDGE - out
8		7406	WX8G	OUTPUT BUFF FULL

Table H-2: IBM AT Implementation of the GPC Link

H.2. Tension Sensor

The cloth tension sensor consisted of a bridge of four strain gauges. The bridge was supplied with ±5 VDC regulated supplies. The bridge output was amplified 1000 times by an AD524 instrumentation amplifier, which operated with ±12 VDC regulated supplies. The regulated power supplies were housed in a separate box to improve noise insulation. The circuits for the power supply unit is shown in fiq. H-2. The strain gauge bridge and amplifier circuit is shown in fig. H-3. The overload protection circuit, descibed in section 4.3.5.2, is shown in fig. H-4.

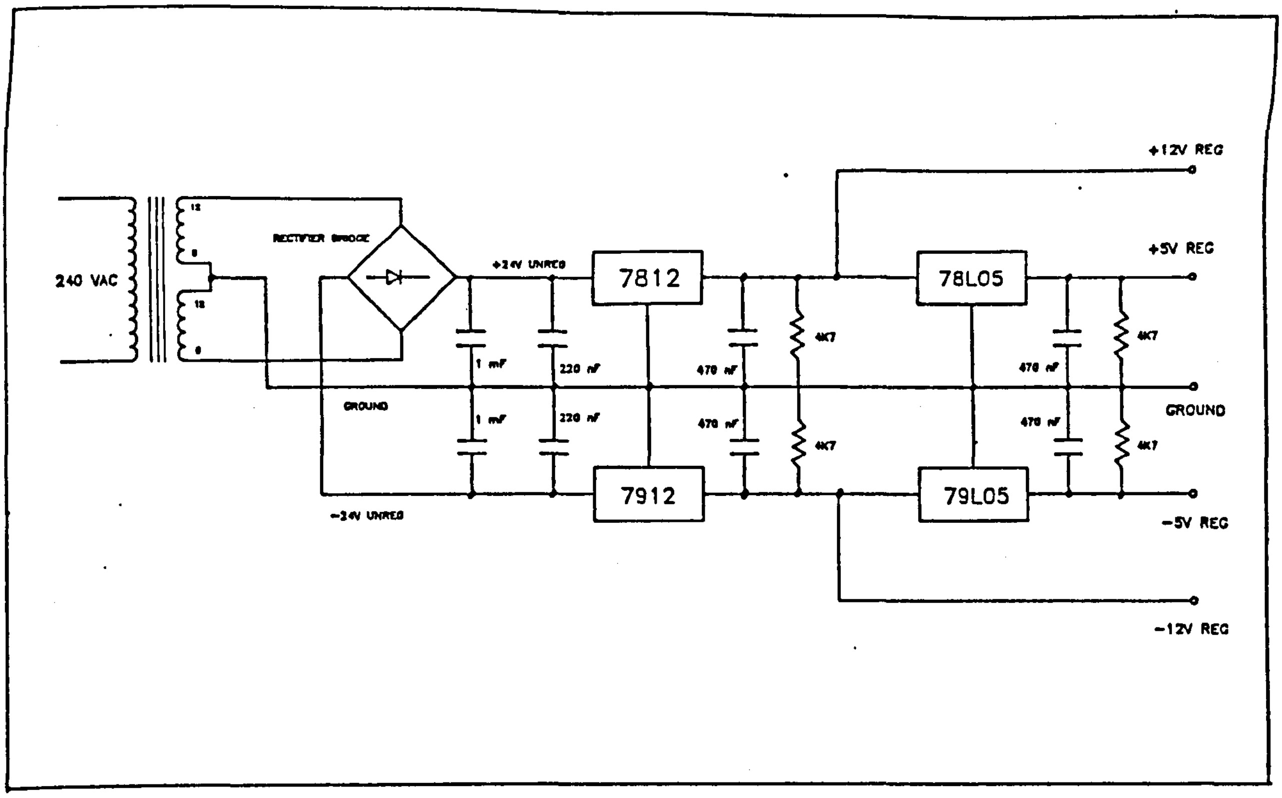


Fig. H-2: Power Supply Unit

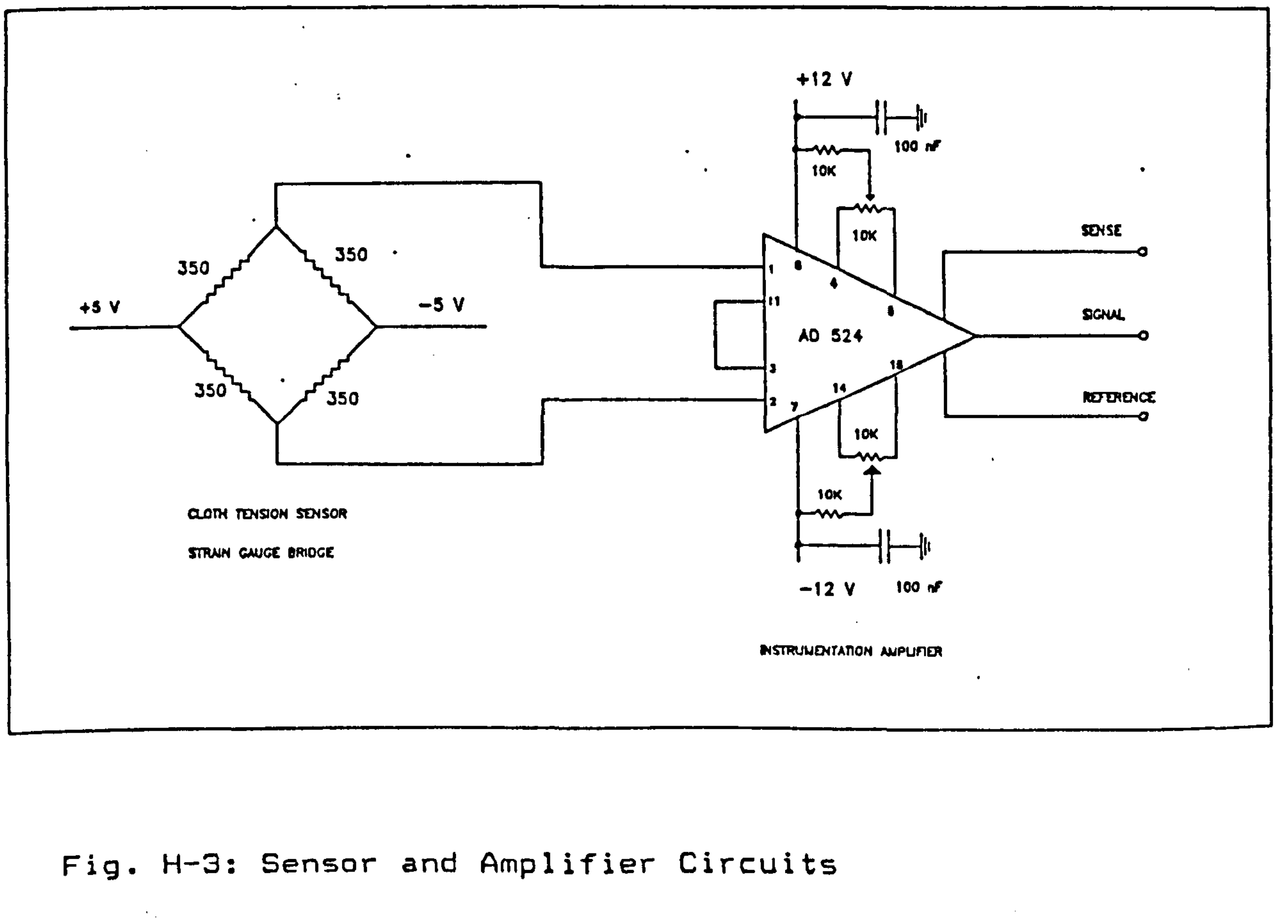


Fig. H-3: Sensor and Amplifier Circuits

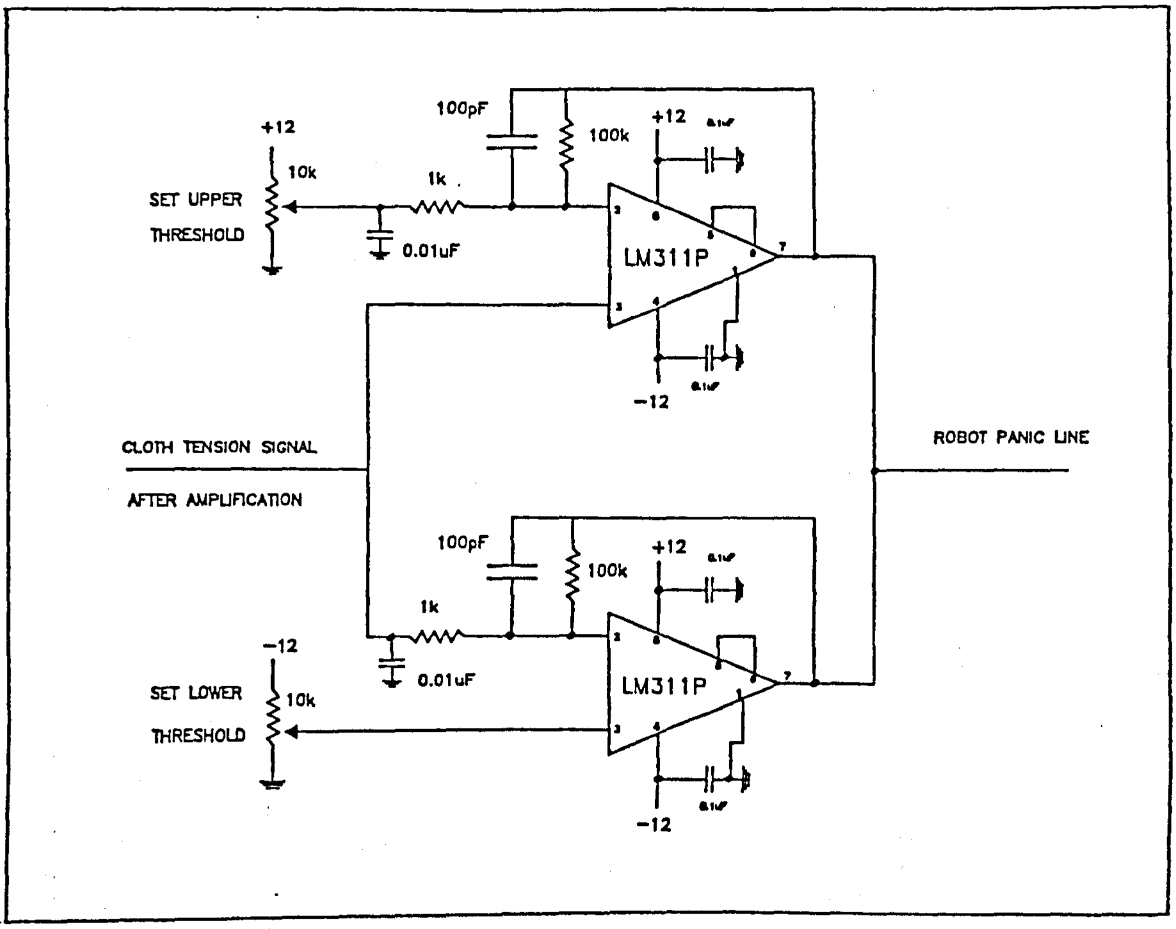


Fig. H-4: Overload Protection Circuit

APPENDIX I

PAPER PRESENTED AT THE 16th ISIR, BRUSSELS, 1986.

ROBOTIC SEWING USING MULTI-SENSORY FEEDBACK

D. Gershon and I. Porat, University of Leeds, England

I.1. ABSTRACT

To date, little has been published on the development of robotic automation for the garment industry. The major distinction between automating garment assembly and other manufacturing processes is the extensive sensory capabilities required to perform the simplest of operations on cloth.

This paper describes the development of a robotic cell to sew a contoured seam on cloth. The system was designed and analysed using a simulation program which accounted for control transfer function, and non-linearities such as camera pixel resolution, time delays and arm movement limitations.

The cell comprises a PUMA robot under real-time path control with feedback loops for edge tracking, cloth tension and cloth feed tracking. Cameras, a cloth tension sensor and the sewing machine shaft encoder provide the sensory input.

I.2. INTRODUCTION

The clothing industry is starved of flexible automation equipment such as has been available in other manufacturing industries, despite growing demands for this technology [1,2,3]. Although dedicated semi-automatic devices have been developed, the application of flexible automation systems based on robotics to garment assembly and handling operations has been hindered by the unpredictable and awkward nature of limp fabric [4,5].

The Clothing Automation Group at the University of Leeds has a comprehensive research programme aimed at the development of techniques and devices which will pave the way for the implementation of Flexible Manufacturing Systems in the clothing industry. One long-term project, (named FIGARO - Flexible Intelligent Garment Assembly Robot), investigates robotic fabric handling and sewing skills.

This paper describes the development of a robotic sewing capability of a contoured seam without the use of mechanical guides.

I.3. SYSTEM OVERVIEW

I.3.1. Concept (fig. 1)

The robot holds the end of the cloth against a smooth table using two rubber-tipped fingers. The fingers are spring-mounted onto the end-effector. The cloth is fed into the sewing machine by the conventional feed mechanism of the sewing machine. The robot's path is generated in real-time by two sensory servo systems :-

a) a seam tracking servo that controls the sideways and rotational movements of the end-effector, based on visual tracking of the cloth edge.

b) a cloth feed tracking servo that controls the forward motion of the end-effector, based on the sewing machine shaft encoder signal and and on the cloth tension measured by an instrumented finger.

I.3.2. Development System (fig. 2)

The development system is organized around a master controller with the robot controller and the sensors as slaves.

The master processor is an IBM AT operating under the AMX-86 real-time, multi-tasking executive. The interrupt service procedures and high speed communication routines are written in 8086 assembler, and the rest of the routines are written in C.

The robot is a Unimation PUMA 560 with VAL II. A major advantage of the VAL II system is the ALTER facility which permits real-time path control by an external computer. Full descriptions of the VAL II system may be found in references [6,7,8].

There are two communications channels between the IBM AT and VAL II :-

a) The ALTER channel is a high speed (19.2 kbaud) serial communication line dedicated to transferring real-time path control data from the IBM to VAL II. The ALTER protocol permits robot position data to be updated every 28 ms.

b) A general purpose 8 bit parallel communication channel was developed by the Leeds University Clothing Automation Group, which is used for process control, task synchronization and parameter passing. The channel combines the I/O binary signals from VAL II with an 8255 PPI chip in the IBM.

The sewing machine is a conventional Mitsubishi LS2-190 lockstitch machine with drop feed, underbed thread trimmer and a microprocessor controlled needle-positioning motor with a non-contact clutch. The sewing machine controller was interfaced to the IBM AT permitting central control of all sewing machine functions.

The seam tracking and cloth feed tracking servo systems are described in the following sections.

I.4. SEAM TRACKING SERVO SYSTEM

I.4.1. Simulation Program (fig. 3)

The seam tracking servo was developed with the aid of a simulation program. The program had the following input variables :-

- pixel resolution of linear array camera in line with needle
- pixel resolution of optional second linear array camera
- distance between cameras
- servo transfer function and gain parameters
- system time delay
- cloth feed speed
- initial seam error
- seam width
- limits on ALTER increments (to ensure smooth robot motion)

The system time delay was a single parameter which accounted for camera sampling rate, processor delays and actuation delays. The program assumed that sideways motion of the robot produced perfect pivoting of the cloth about the needle, without buckling. Figure 4 shows two typical simulation runs.

The simulation program demonstrated that stable control depended on applying the transfer function to the actual seam error and not to the measured error (fig. 3c). The actual seam error was calculated from the measured error and from a calculated incidence angle.

Furthermore the servo was always unstable when a single camera was used. The servo was well controlled when a second camera was specified at a distance of 20 mm in front of the first, and when a large derivative gain was combined with a small proportional gain. A linear array of thirty pixels with a resolution of 0.5 mm gave a satisfactory performance.

Stability was strongly dependent on the system time delay and seam tracking became more difficult as the sewing speed was increased. As would be expected, the maximum speed at which satisfactory performance could be obtained increased as system time delay decreased.

I.4.2. Vision System

Two I-SIGHT cameras were selected because of their small size and low cost. This 30 by 32 pixel CCD camera is decribed in reference [9]. The camera's low resolution permits high frame rates which is so essential in real-time control. The resolution was satisfactory since only a small field of view was required, and because the cameras could be placed close to the table. The extra pixel dimension, provided by the camera's two dimensional array, was utilised to attenuate signal noise by averaging the edge position measurement over three rows.

The cameras were interfaced to the IBM via a circuit board installed in the IBM bus. The camera board consists of individual frame stores for each camera and a Z80 processor which is responsible for picture grabbing, exposure timing and thresholding. The IBM AT read the frame stores using a DMA block move. Typically the time taken from triggering the cameras, to reading both frame stores and finally calculating the seam error was 11 ms.

The lighting arrangement consists of a projection lamp directed at the table's mirror surface, and was found to be effective for all types and colours of fabric.

I.4.3. End-Effector Rotation

In order to prevent buckling of the cloth and to encourage correct pivoting of the cloth about the needle, it was necessary to combine all sideways movements of the end-effector with a simultaneous pivoting of the end-effector about the instrumented finger. The auxiliary finger was rotated about the instrumented one so that both fingers were at all times equidistant from the needle (fig. 5).

With the VAL II system this rotation was easily achieved by defining the TOOL transformation so that the WORLD Z axis was colinear with the finger's centre-line.

I.5. CLOTH FEED TRACKING SERVO

I.5.1. Sewing Machine Encoder Signal

The encoder signal was read into a counter to track the sewing machine revolutions. The counter was set to zero at the start of a seam so that the robot's position update in the forward direction was given by :-

$$x = c * s / b \qquad (1)$$

where x = robot position demand
c = instantaneous count

b = no. of counts per revolution
s = stitch length

Although the robot could track the sewing machine's feed-dog speed accurately by using the counter, in practice it could not track the cloth speed accurately. The discrepancy between feed-dog speed and cloth speed was due to slipping between the cloth and the feed mechanism. This discrepancy could not be compensated for because the slipping was unpredictable and varied for different fabrics. Evidently a cloth tension sensor was necessary for correct cloth feed tracking.

I.5.2. Cloth Tension Sensor (fig. 6)

The cloth tension sensor was designed for minimum hysteresis and maximum mechanical decoupling. The two slender parallel beams were machined from a solid block of Al 2024. Similar force sensors are described in references [10,11,12]. A foil strain gauge was bonded to each beam face. The sensor sensitivity obtained was 2.6 mV/N before amplification in the x direction. Good decoupling was achieved with a cross-sensitivity of 0.2 mV/N. Thus, the ideal cloth tension during sewing, which is 0.5N, represented a signal of 1.3V after amplification.

When the sensor signal was viewed on an oscilloscope, it showed a regular rise and fall of cloth tension per stitch due to the intermittent nature of the feed mechanism. Since the feedback control requires an instantaneous reading of cloth tension, sampling the raw sensor signal would be unsatisfactory. The signal was interfaced to a digital peak detector so that at each sample the processor would read the peak tension since the previous sample. The sampling rate was such that the reading obtained was the peak tension over several stitches.

I.5.3. Cloth Feed Tracking Control

The feedback control based on the cloth tension sensor was complicated by the effect of friction between the table surface and the finger (fig. 7). When the robot moves forward the measured cloth tension is less than the actual tension because of table friction. However, when the robot moves backwards, the table friction changes direction and the measured tension is larger than the actual tension. Clearly, control would be impossible if end-effector displacements were permitted in both directions. Consequently, the robot was limited to forward displacements only, and the small offset due to the table friction was easily compensated.

Satisfactory cloth feed tracking was achieved by combining integral and proportional control on the cloth tension signal, with the displacement calculated from the shaft encoder signal (fig. 8).

I.6. SYSTEM PERFORMANCE

I.6.1. Seam Tracking

Figure 9 shows areas of gain values in which satisfactory seams could be obtained for 2 different sewing speeds. The solid contour line is the boundary within which satisfactory seams were obtained, and the dotted line shows the region within which good seams were obtained.

At 1600 rpm, satisfactory seams were obtained with increasing derivative gain for increased proportional gains. But when the proportional and derivative gains were further increased, unsatisfactory seams were obtained. However, at higher speeds, large proportional and derivative gains had to be applied to obtain satisfactory seams. These large gains when applied at the lower speed produced unsatisfactory seams. An adaptive control technique is possibly indicated.

The system stability can be readily improved by minimising the total time delay between measurement and actuation. The time delay comprises the following main components :-

- actuation delay (i.e. robot speed)
- VAL II transformation calculations
- ALTER update rate (every 28 ms)
- IBM communication overhead (8.7 ms per 28 ms)
- camera exposure and capture time (10 ms)

The IBM communication overhead could be reduced if a separate processor was used for managing the ALTER high speed communications. The use of a four axis SCARA robot would reduce the VAL II transformation calculations. A faster robot and a higher ALTER update rate would also benefit performance.

I.6.2. Tension control

The cloth feed tracking servo limited excessive tension variations, sufficiently to sew satisfactory seams. However, tension variations had an amplitude of up to 0.7N. More work is required to control the tension within closer limits.

I.6.3. Seam quality

Excessive tension variations and buckling of the cloth produced seam puckering. The tendency to buckle was reduced by using a highly polished smooth stainless steel table top and by limiting robot displacements to ensure a smooth sliding motion. Seam quality varied considerably for different fabrics; open structure fabrics were very tolerant of tension variation, heavy fabrics were resistant to buckling forces, but light and tightly structured fabrics were more sensitive.

The sewing machine's presser foot, which holds the cloth against the feed dogs, hinders pivoting of the cloth about the needle. This effect becomes more severe as the robot approaches the needle. Consequently, this method of robotic sewing is at present only effective for finger to needle distances between 1000 mm and 250 mm. A refinement of this technique to enable satisfactory sewing close to the needle is being developed.

I.7. CONCLUSION

An adaptive robotic sewing system has been developed that uses multi-sensory inputs to manipulate the cloth in real-time. The system is stable within a narrow margin. The stability margin can be improved by reducing the system time delay. Seam quality can be improved by a more precise tension control.

I.8. ACKNOWLEDGEMENTS

I would like to acknowledge the financial assistance provided by the Textile and Other Manufactures Requirements Board of the Department of Trade and Industry.

REFERENCES

[1] Lower, J. M., "Automation Heard Around the World", Bobbin, Vol 26, No. 8, April 1985.

[2] Tredwin, P., "Computerised Garment Manufacture", Proc. Annual World Conference, The Textile Institute, London, May 1985.

[3] "Automation in Apparel", Bobbin, Vol 23, No 5, Jan 1982.

[4] Nilsson, N., "The Apparel Crisis in Sweden - Countermeasures and Developments", Bobbin, Dec 1982.

[5] Taylor, G.E., Kemp, D.R., Taylor, P.M., Pugh, A., "Vision Applied to the Orientation of Embroidered Motifs in the Textile Industry", 2nd ROVISEC, England, 1982.

[6] Shimano, B.E., Geshke, G.G., Spalding, G.H., Smith, P.G., "A Robot Programming System Incorporating Real-Time and Supervisory Control, VAL II", Proc. of Robots 9, Vol 2, Detroit, June 1984.

[7] Loughlin, C., *Morris, J., Rovetta, A., Franchetti, I.,* "Line, edge and contour following with eye-in-hand vision system", 14th ISIR, Gothenburg, Sweden, Oct. 1984.

[8] Van der Heijden, F.J.M., "Assembly of small components by a vision-controlled robot", 5th ROVISEC, Amsterdam, Oct. 1985.

[9] Loughlin, C. and Morris, J., "Application of eye in hand vision", 7th B.R.A., Cambridge, England, May 1984.

[10] Van Brussel, H., Belien, H., Thielemans, H., "Force Sensing for Advanced Robot Control", 5th ROVISEC, Amsterdam, Oct. 1985.

[11] Lestelle, D., "Gripper with finger built-in force/torque sensors", 5th ROVISEC, Amsterdam, Oct. 1985.

[12] Rosen, C., et al, "Exploratory Research in Advanced Automation", Second Report, Stanford Research Institute, Aug. 1974.

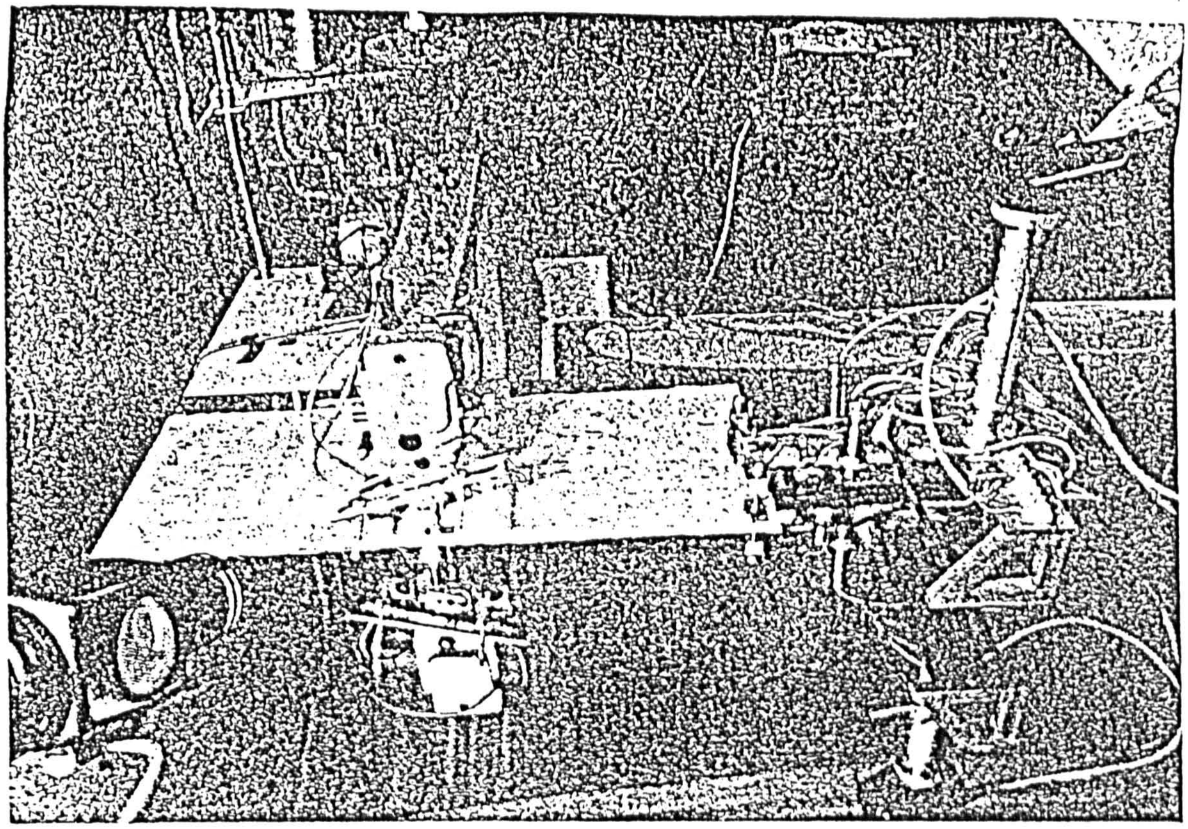


Figure 1. Robotic Sewing System

The cameras are more clearly seen in the reflection of the sewing table's mirror surface

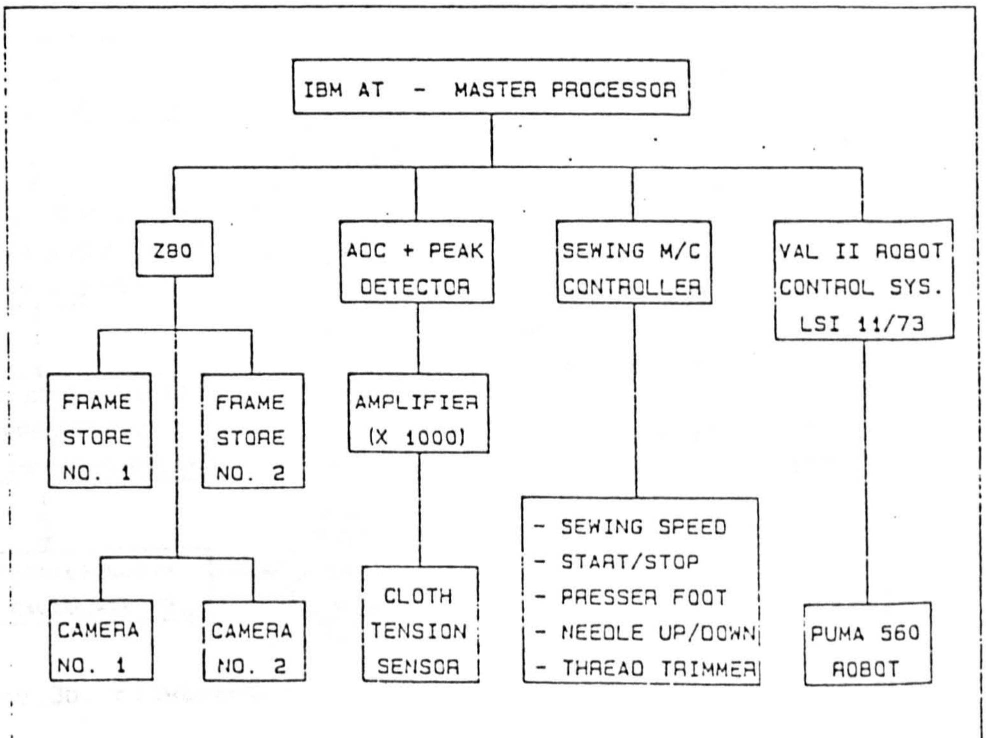
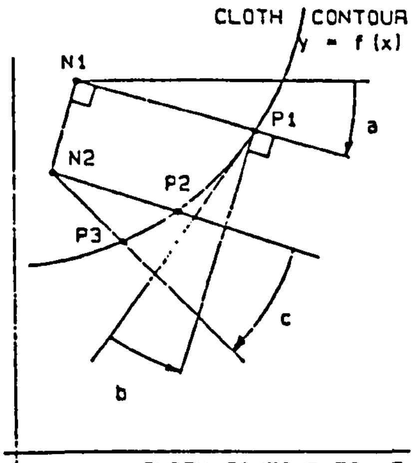


Figure 2. Block Diagram of Development System



a - ANGLE BETWEEN SEWING M/C AND CLOTH CONTOUR AXES

b - CLOTH INCIDENCE ANGLE INTO SEWING M/C

c - CLOTH ROTATION ANGLE TO CORRECT FOR SEAM ERROR

N1 - NEEDLE POSITION AT TIME t1

N2 - NEEDLE POSITION AT TIME t2

N1P1 - MEASURED SEAM WIDTH AT t1

N2P2 - MEASURED SEAM WIDTH AT t2

N2P3 - MEASURED SEAM WIDTH AFTER CLOTH ROTATED

Figure 3a Parameter Definitions

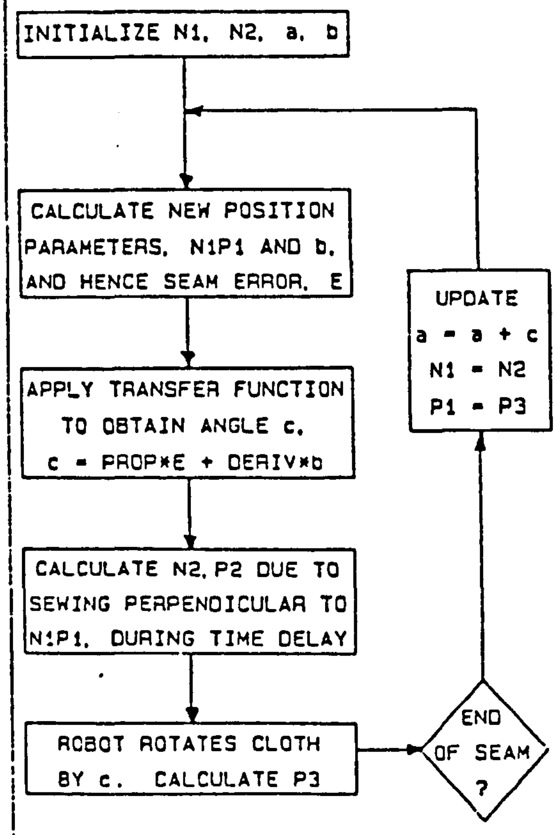
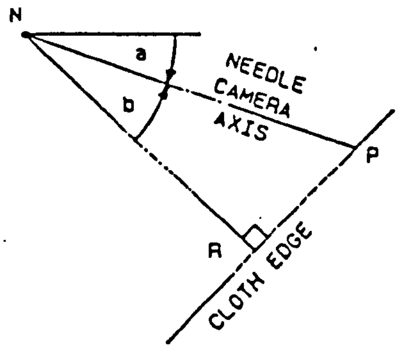


Figure 3b. Flowchart



NP - MEASURED SEAM WIDTH

NR - ACTUAL SEAM WIDTH

Figure 3c. Seam Width Calculation

Figure 3. Simulation Program

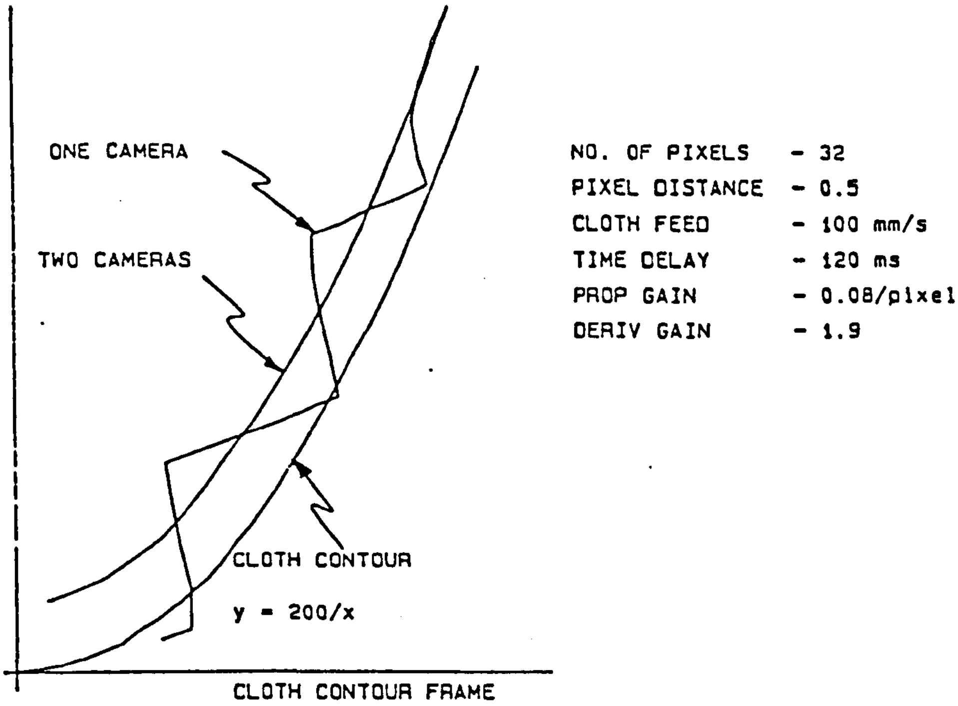


Figure 4. Simulation Results

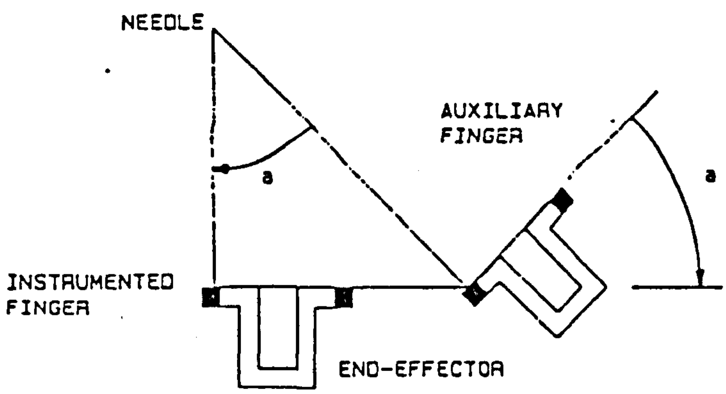


Figure 5. Relationship between Rotation and Sideways Motion of End-Effector

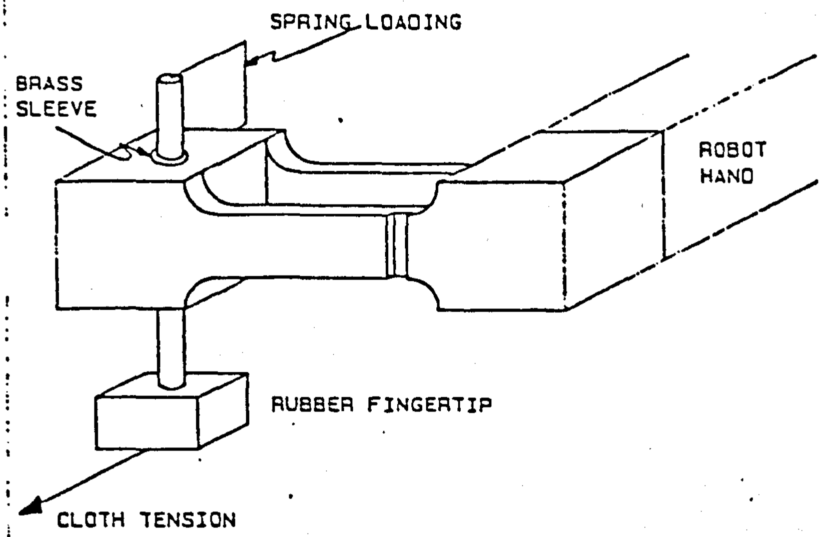


Figure 6. Cloth Tension Sensor

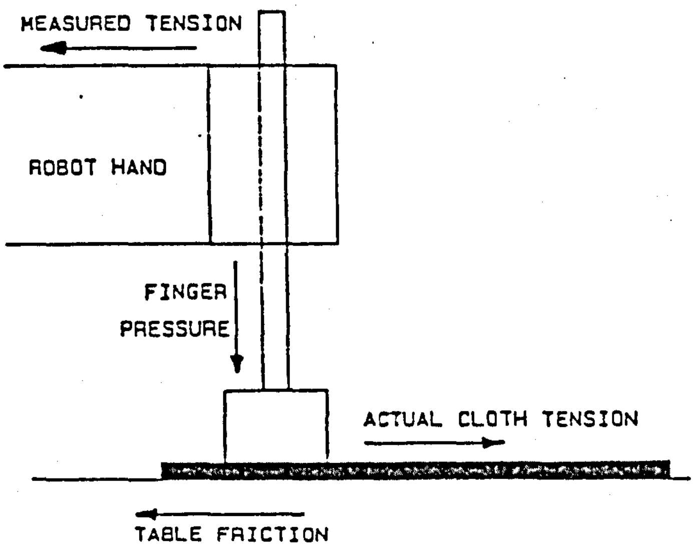


Figure 7. Table Friction and Cloth Tension Measurement

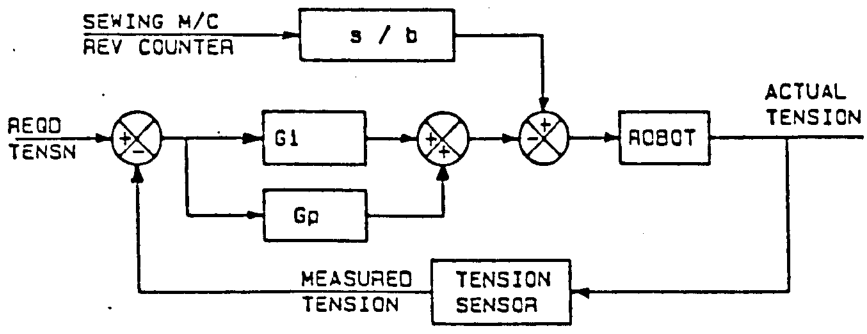


Figure 8. Cloth Feed Tracking Control System

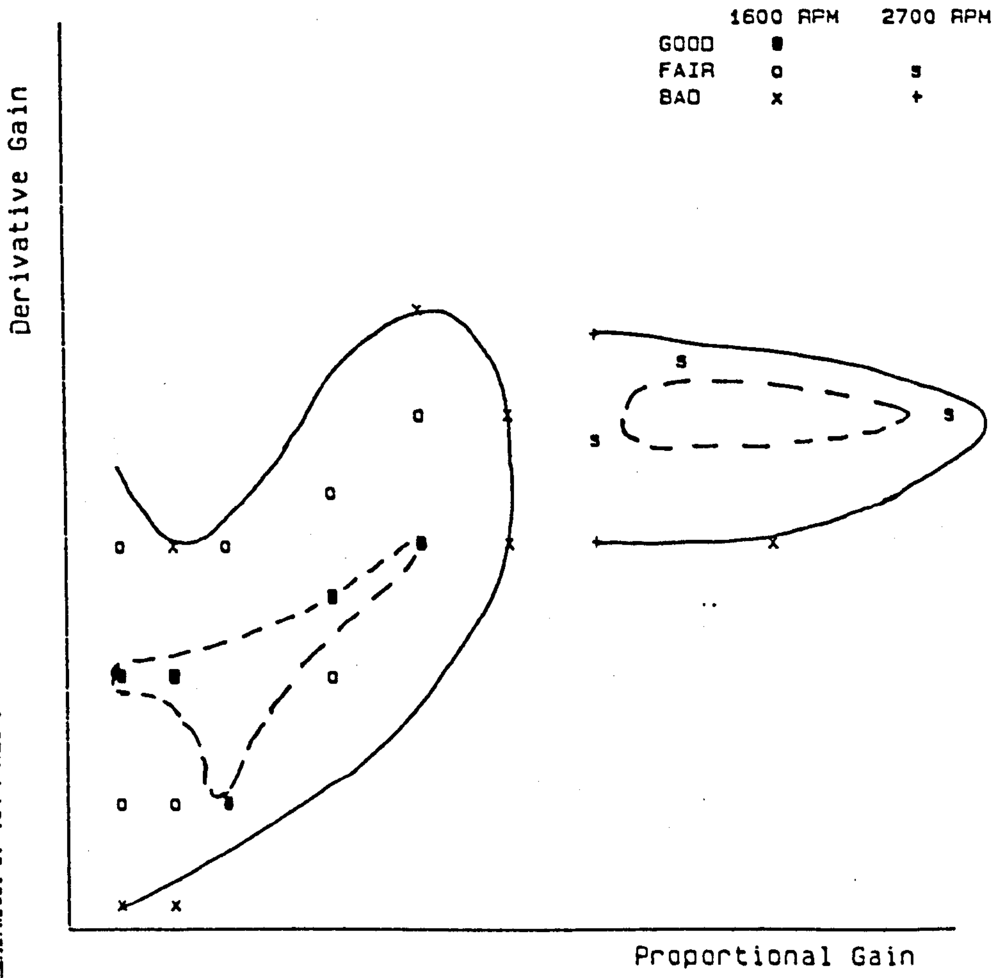


Figure 9. Seam Tracking Performance