THESIS CONTAINS CD/DVD

The Design and Evaluation of an Interface and Control System for a Scariculated Rehabilitation Robot Arm

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Bernard Neil Parsons

June 2001

School of Engineering Systems
Middlesex University
London (United Kingdom)

Abstract

This thesis is concerned with the design and development of a prototype implementation of a Rehabilitation Robotic manipulator based on a novel kinematic configuration. The initial aim of the research was to identify appropriate design criteria for the design of a user interface and control system, and for the subsequent evaluation of the manipulator prototype. This led to a review of the field of rehabilitation robotics, focusing on user evaluations of existing systems. The review showed that the design objectives of individual projects were often contradictory, and that a requirement existed for a more general and complete set of design criteria. These were identified through an analysis of the strengths and weaknesses of existing systems, including an assessment of manipulator performances, commercial success and user feedback.

The resulting criteria were used for the design and development of a novel interface and control system for the Middlesex Manipulator - the novel scariculated robotic system. A highly modular architecture was adopted, allowing the manipulator to provide a level of adaptability not approached by existing rehabilitation robotic systems. This allowed the interface to be configured to match the controlling ability and input device selections of individual users.

A range of input devices was employed, offering variation in communication mode and bandwidth. These included a commercial voice recognition system, and a novel gesture recognition device. The later was designed using electrolytic tilt sensors, the outputs of which were encoded by artificial neural networks. These allowed for control of the manipulator through head or hand gestures.

An individual with spinal-cord injury undertook a single-subject user evaluation of the Middlesex Manipulator over a period of four months. The evaluation provided evidence for the value of adaptability presented by the user interface. It was also shown that the prototype did not

currently conform to all the design criteria, but allowed for the identification of areas for design improvements.

This work led to a second research objective, concerned with the problem of configuring an adaptable user interface for a specific individual. A novel form of task analysis is presented within the thesis, that allows the relative usability of interface configurations to be predicted based upon individual user and input device characteristics. An experiment was undertaken with 6 subjects performing 72 tasks runs with 2 interface configurations controlled by user gestures. Task completion times fell within the range predicted, where the range was generated using confidence intervals ($\alpha = 0.05$) on point estimates of user and device characteristics. This allowed successful prediction over all task runs of the relative task completion times of interface configurations for a given user.

Acknowledgements

My director of studies, Peter Warner, succeeded in generating in me some of the strong motivation he holds for developing a rehabilitation robot arm. Throughout the project, his wealth of experience has very quickly placed the problems I have encountered into an appropriate context, and guided me towards a solution. Along with my supervisors Anthony White and Raj Gill, I have been provided continual support and guidance, without which very little would have been achieved.

Many thanks to Jeremy Lewis who has provided a degree of help far above and beyond the call of friendship : from technical and spiritual advisor, to proof reader extraordinaire. Steve Prior's encouragement and support at the beginning of the project provided a guiding light that was sorely missed when he disappeared to sunnier climes. I am grateful to Roger Delbourgo for helping out with the trigonometric analysis in chapter 4. I would like to thank members of the Computing Science department at Middlesex University, for supporting this research within Engineering Systems. In particular Harold Thimbleby, Ann Blandford, Christopher Kindberg, Mathew Jones and Mat Smith - their expertise within the field of human-computer interaction has been invaluable.

The technical staff within Engineering Systems have suffered my continual badgering, and in spite of this, still provided me with help. Thanks to Imtiaz Bhaiji, Issam Siman, Ken Bone, Goodwin Griffiths, and Neil Matticks.

I owe a great debt of gratitude to Paul Rocca for first agreeing to participate in the user evaluation, and then for investing such energy and commitment over a prolonged period of time. His understanding of the issues involved in assistive technology have formed an essential part of this work. The same is true of the experience and understanding of the field held by Robin Platts

of the Royal National Orthopaedic Hospital, whose association with this project was also a vital component.

I am grateful to the postgraduate and undergraduate students who have undertaken projects in support of the work reported here. In particular Lothar Gellrich, for his long hours and hard work during his six month visit to London.

My friends and colleagues at Middlesex University have provided a supportive and friendly working environment, particularly Jaime Valles-Miro, Jagpal Surdah and Aboubaker Lasebe of Advanced Manufacturing and Mechatronic Systems.

Finally, thanks to my family for their unwavering support and encouragement.

In loving memory of

Clive Arnold Parsons

Who taught me all about travelling.

To travel hopefully is a better thing than to arrive, and the true success is to labour.

Robert Louis Stevenson.

Table of Contents

Section II

Manipulator and motor control system design

Section III

USER INTERFACE DESIGN

Section IV

Evaluation

Section V

Adapting the User Interface

Section VI

Discussion and Conclusions

Section VII

Supporting materials

List of abbreviations and acronyms

A/D	Analogue/Digital
ADL	Activities of daily living
ANN	Artificial neural network
BP	Back propagation
C	The C programming language
C++	The C++ programming language
CLG	Command language grammar
DC	Direct current
DOF	Degrees of freedom
DDE	Dynamic Data Exchange
DDEML	Dynamic data exchange management library
DLL	Dynamic link library
DM	Dialogue Manager
DPA	Dynamic programming algorithm
FDM	Feedback device manager
FET	Field effect transistor
GOMS	Goals, operators, methods and selection rules
HCI	Human – computer interaction
HTA	Hierarchical Task Analysis
IC	Integrated circuit
IDM	Input device manager
IO	Input/output
ISO	International standards organisation
JMCL	Juvo motor control language
JUCL	Juvo user command language
JUVO	Meaning 'to assist' in Latin - is a term used to refer to the Middlesex Manipulator
LCD	Liquid crystal display
LTM	Long term memory
MHP	Model human processor
MLP	Multi-layer perceptron
MLU	Modal logic unit
NGOMSL	Natural GOMS language
OOA	Object oriented analysis
PC	Personal computer
PID	Proportional Integral Derivative control
PPI	Programmable peripheral interface
PS	Problem severity
PUMA	Programmable universal machine for assembly

PWM	Pulse width modulation
RBF	Radial basis function
RC	Resistor/Capacitor
SBUF	Serial buffer (on 8051 microcontroller)
SC	Solution cost
SCARA	Selective compliance assembly robot arm
SCON	Serial control register (on 8051 microcontroller)
s.d.	Standard deviation
SLP	Single layer perceptron
TAG	Task Action Grammar
TAL	Task Action Language
TIDE	Technology Initiative for Disabled and Elderly people - a European Union funding initiative.
UART	Universal asynchronous transmitter receiver
UML	Universal modelling Language
UIS	User interface system
VA	Veterans Association (USA).
VB	Visual basic programming language
VDU	Visual display unit
WM	Working memory

List of Figures and Tables

Section I

Introduction and literature survey

Chapter 1

Introduction

This chapter provides an outline of the research objectives, and describes the work that was undertaken to achieve these. Two questions are addressed within this thesis, the first being :

Does the Middlesex Manipulator – a prototype implementation of the Scariculated robot design - conform to a set of design criteria that are essential for the design of successful rehabilitation robotic systems ?

To answer this question, a number of steps were required, including : defining appropriate design criteria, building a control system and user interface, supervising the construction of the Middlesex Manipulator prototype, and evaluating the prototype.

The project led to an examination of the relationship of Rehabilitation Robotics to related fields, such as human computer interaction (HCI) and artificial intelligence. This resulted in the novel application of design and evaluation methodologies from these fields. In particular, this work addressed the question :

How may existing HCI evaluation methodologies be used to quantify the effect of adaptability on the usability of an interface designed for a rehabilitation robot arm ?

1.1 Motivation & background

Rehabilitation Robotics provides an area of research with a unique combination of challenges, rewards and fascinations. The principal challenge is to increase the independence and enrich the lives of people with physical disabilities. This challenge has been taken up by a number of mainly academic or government-assisted institutions over the past three decades. This is in contrast to 'mainstream' robotics, where development has occurred primarily in the private sector. Why should it be the case that this field is so different ? A cynical view may be that there is a perceived lack of high-profit on offer. But it is certainly also the case, that there exists far greater diversity, both in the expertise required, and within the potential user-group, who necessarily form part of a lengthy iterative design-cycle.

The design-cycle of the field as a whole has now passed the proof-of-concept stage, and has produced commercially available systems, (for example, Kwee 1989 and Topping 1996). However, these systems have achieved only a limited amount of success if measured by the degree of user-acceptance and commercialisation that they have attained. As discussed in Chapter 2, their advent has provided new challenges, and underlined the importance of some of the aims of the pioneering projects, which remain only partially achieved.

The research reported in this thesis has been motivated and guided by these challenges, building upon some initial research initiated at Middlesex University in 1988 (see Prior et. al., 1992). The research objective was to develop a robotic arm capable of assisting people with physical disabilities in activities of daily living (ADL). An extensive survey of potential users was undertaken to identify user requirements. The survey produced a set of user tasks that were assessed in terms of the cost, complexity, accuracy and the payload that they would require. This work resulted in the novel 'Scariculated' kinematic configuration – a combination of the SCARA robot design, and the vertically articulated design. The work reported in this thesis relates to the development of a control system and user-interface for a prototype implementation of the Scariculated design, referred to as the Middlesex Manipulator.

1.2 Objectives and deliverables

The initial objective of the work reported in this thesis was to test whether a prototype implementation of the Middlesex Manipulator conformed to design criteria that are essential for the development of a successful rehabilitation robotic arm. The work undertaken towards this objective was as follows :

Identification of design criteria

A Criterion is defined within the Oxford Concise English Dictionary as 'a principle or standard by which a thing is judged'. In the case of a rehabilitation robotic arm, judgment is ultimately performed by the users or potential users of the arm. Design criteria were therefore identified by undertaking a review of the field of rehabilitation robotics, with a particular focus on the results of the user evaluations of exiting systems. This exercise allowed for the identification of general criteria, the conformance or violation of which could be used to explain the successes and failures of existing systems. Design criteria differ from the design requirements of specific projects in their level of generality, thus design criteria provide a measure against which different systems may be compared.

Design and implementation of a motor control system and user interface

Following a review of the field of rehabilitation robotics, design requirements appropriate for the Middlesex Manipulator's user interface and control system were specified. These were then used to develop a motor control system for the manipulator prototype. Implementation of a user interface included an investigation into novel forms of user interaction. This necessitated the design of an appropriate software architecture, and an investigation into novel input devices. This occurred in parallel with the supervision of the construction of the manipulator prototype based upon the Scariculated design.

Evaluation of the Middlesex Manipulator

Once a functioning Manipulator had been realized, a process of evaluation was undertaken. Initially this consisted of assessing the manipulator's construction and performance against design requirements. This was followed by an extensive user evaluation of the prototype by an individual with spinal-cord injury. The results were measured against the design criteria allowing for an assessment of whether the current prototype could reasonably be evolved into a manipulator that was likely to attain wide user acceptance.

The three components described above were necessary stages in achieving the stated objective, and were used to form a process analogous to an iterative design-cycle. For example, the user evaluation was undertaken in four separate phases. The outputs of each phase allowed modifications to be made to the user interface and control system, and for the design requirements to be modified or verified.

Adapting the user interface

Work towards the design of an appropriate user interface included a review of the field of Human-Computer Interaction (HCI). The objectives of the field of HCI are common to those of rehabilitation robotics, and to the field of assistive technology in general. However, past overlap between the two fields has been limited. Typical HCI evaluation methodologies are biased towards the use of graphical user interfaces, with conventional input devices and the notion of 'typical' users. Consequently, the design requirements of the Middlesex Manipulator presented issues that are not typically addressed by the HCI evaluation methodologies reviewed. Principal amongst these requirements, was the project's need for the interface to be configurable to match the controlling ability of a specific individual. This problem led to the second objective of the work reported in this thesis : to test whether HCI evaluation methodologies may be used to quantify the effect of adaptability on the usability of an interface designed for a rehabilitation robot arm.

A method of Task Analysis was identified that may be used to make predictions of the relative usability of different interface configurations. Possible measures of usability include the time required to undertake tasks, error frequencies, error recovery times, interface complexity and interface consistency. In its standard form, this approach incorporates the concept of a typical user, which is represented by the 'Model Human Processor'. A form of task analysis is proposed within this thesis that replaces this model with estimates of users' controlling ability. Controlling ability is defined as the combination of the user's functional ability, and device characteristics. An experiment was undertaken as part of this research, to test whether this novel form of task analysis could be used to quantify the effect of adaptability on the usability of an interface designed for a rehabilitation robotic arm.

1.3 Contribution to rehabilitation robotics research

This thesis presents a set of general design criteria for the development of rehabilitation robotic arms. These criteria are likely to be refined and developed through future technological advances, however their existence is necessary to consolidate the lessons learnt from existing systems. This thesis argues that systems currently prominent within the field do not adequately conform to these criteria, and that this has significantly limited their user acceptance. Nevertheless, an examination of the various combinations of attributes previously achieved, demonstrates the feasibility of greater success.

The work reported in this thesis produced a novel Rehabilitation Robot arm : the Middlesex Manipulator. Similar systems currently exist (for example Hillman and Jepson 1997, and Sheredos 1996), but none have conformed to the same design criteria, and each offers unique solutions to design requirements. Consequently, continual evaluation and comparison of these systems, as undertaken within this thesis, is required to progress the field of rehabilitation robotics towards its aims.

This thesis attempts to contribute towards encouraging an overlap between rehabilitation robotics and related fields. Whilst work within assistive technology is necessarily multi-disciplinary, this thesis has formally applied techniques from the field of artificial intelligence and human-computer interaction, providing novel ways of implementing and analysing user interaction. Specifically, a novel form of Task Analysis was developed and tested. Results demonstrated the technique's unique applicability to the assessment of the relative usability of configurable user interfaces, where user's controlling ability is a significant determining factor.

1.4 A brief history of the work

Initial work on the content of this thesis began in January 1995 with an up-to-date review of the field, allowing appropriate design criteria for a control system and user interface for the Middlesex Manipulator prototype to be identified.

A motor control system employing low-cost embedded microcontrollers was developed through the course of 1995. Motor control software was developed for the embedded system, using the C programming language, in the last half of 1995.

This took place in parallel with the supervision of a number of postgraduate Mechanical Engineering students, undertaking the construction of a Scariculated prototype.

Systems analysis techniques were used to model and define typical user tasks, leading to the design of a modular software architecture for the user interface and manipulator controller, which commenced in the last half of 1995. By April 1996, implementation of this system began, involving the development of a number of Windows applications using C++.

Investigations into the development of novel communication devices resulted in the identification of an electrolytic tilt-sensor that proved capable of encoding simple hand-gestures and head-gestures. The first half of 1996 involved the development of a gesture recognition system, employing an artificial neural network, that may be used to classify gestures issued by either the sensor, or by a standard trackball.

By the end of 1996 a working prototype was ready, and an initial evaluation of the system was undertaken, by an individual with a C4[1] incomplete spinal cord injury. Results of the evaluation allowed for the refinement and improvement of the manipulator system, which continued throughout 1997. A further two user evaluations were undertaken, each more extensive than the last. These involved semi-structured interviews and user observations, addressing aspects of the interface and manipulator. Typical user tasks such as pick-and-place, feeding and drinking were undertaken.

A review of the field of human-computer interaction had been undertaken to assist with the development of the user interface. In May 1997 this work focused on the application of HCI evaluation methodologies to assist configuring adaptable systems. These ideas were developed through 1997, resulting in a modified form of task analysis. An experiment was designed and undertaken in February 1998 to test the suitability of task analysis for interface configuration.

1.5 Structure of the thesis

This thesis is divided into the following major sections. Section (I) *Introduction and literature survey* : provides an introduction to the work undertaken in this thesis, and an

[1] C4 refers to the level break within the spinal column. Ranging from 1 to 10, with lower numbers referring to the top of the spine.

overview of the field of rehabilitation robotics. Section (II) *Manipulator and motor control system design* : outlines the background of the Middlesex Manipulator project, and describes the approach taken for the design of a motor control system.. Section (III) *User interface design* : provides a review of the field of human-computer interaction, and presents the design of an adaptable user interface system. Section (IV) *Evaluation* ; presents the results of an initial evaluation of the manipulator. Section (V) *Adapting the user interface :* provides a discussion of how task analysis may be used to configure an adaptable user interface, and describes an experiment undertaken to test this approach. Section (VI) *Conclusion* : provides a summary, a discussion of possible future work, and concluding remarks. Section (VII) S*upporting Materials* : Contains references, a list of acronyms, and appendices including notations used in design specifications, source listings, circuit diagrams and published papers. A video of sections of the user evaluation is included with the thesis, a transcription of which exists as an appendix.

Chapter 2

Rehabilitation Robotics

This chapter provides an overview of the field of Rehabilitation Robotics, by providing a comparison of the characteristics and relative successes of projects that are representative of the field. A number of conclusions are drawn, principally :

- In contradiction to a prevalent cost/benefit argument, if a system is marketed at too high a cost, then user-uptake will be severely restricted, *irrespective* of the system's functionality.
- To maximize user acceptance, a range of user tasks should be addressed, with the minimum performance characteristics defined as those required to undertake these tasks.
- To minimize costs, 'base-line' performance and functionality should be identified, but should be extendable, such that systems may evolve to meet changing user needs and attitudes.
- The system should be mobile, aesthetically acceptable and safe.
- Flexibility should be inherent to the user interface and control system.

These conclusions were used to define a set of design criteria. This thesis argues that whilst the design criteria should evolve, they form a coherent picture of the field as a whole, and should be used as general guidelines for the development of rehabilitation robotic devices. Consequently they were used in the design of a user interface and control system for the Middlesex Manipulator, and formed the criteria against which the system was evaluated.

2.1 Introduction

Assistive Technology may be defined as a field of research which furthers the development of devices that can be used by people with physical disabilities to improve their quality of life. This goal may be achieved by reducing either the severity of a physical impairment, or its effect; i.e. by providing therapy, assistance or both. As a part of this field, Rehabilitation Robotics adopts the same objectives, and attempts to achieve them through the application of robotic technology. The field has developed over the past four decades, with many of its original pioneers active in the development of orthotic and prosthetic devices (for example, the Texas Institute for Rehabilitation and Research, and the VA Palo Alto Research Centre). Rapid progress in robotic technology during the late 1960s and early 1970s, particularly for the automotive industry, led to a widespread interest in its application to Assistive Technology throughout Europe, America and Japan.

This thesis focuses on Rehabilitation Robotic systems that aim primarily to provide forms of assistance, as opposed to therapy, though as discussed below, one is often a by-product of the other. The systems have typically been designed to address either vocational tasks or activities of daily living, and have employed either industrial robots, educational robots or purpose-built arms. However, the most common form of classification has been based on how the robot arms are mounted. The typical categories being : fixed workstations, wheelchair-mounted, or mobile systems. Perhaps inevitably, researchers have disagreed as to where the boundaries between these categories lie. For example, relatively light systems mounted on easily movable platforms are regarded by some as being mobile, and by others as being fixed workstations (c.f. Prior 1993, and Hillman, 1992). Furthermore, systems developed originally to be wheelchair-mounted have been employed as workstations (for example, Driessen, 1997), and technologies developed as workstations are evolving into a mobile form (for example, Dario et. al., 1995). Attempting to relax these forms of classification, the following chapter is structured around a discussion of the prominent examples of rehabilitation robotic projects. No attempt has been made to reference every project ever undertaken, but to focus on a number that collectively represent the culmination of more than thirty years of research and development. For a broader survey of the field see Hillman (1992), Kassler (1993), Dallaway (1995) or Mahoney (1997).

This chapter highlights a number of common and significant issues that have emerged from this research, and from system evaluations in particular. The chapter summarises these issues, and argues that a coherent picture results, that can be used to provide guidelines and design criteria for the development of successful rehabilitation robotic devices.

2.2 The Manus Arm

The Manus arm, illustrated in figure 2.1, was developed primarily as a wheelchair-mounted system to assist with daily living tasks. It employs a sophisticated kinematic structure consisting of eight axes, allowing a wide range of tasks to be addressed. The designers' attention to aesthetics has resulted in a more slender and lighter design than industrial systems with comparable functionality. From a commercial standpoint, only a handful of rehabilitation robotic systems can claim to have had any real success (see Mahoney, 1997), and in terms of the number of units sold, the Manus arm comes second. The system has been evaluated extensively within Europe and the US, and in many ways has acted as an impressive flagship for the field in general. The Manus project was initiated in 1984 by the Dutch Organisation for Applied Physics, though expertise was employed from an earlier French initiative named Spartacus (Guittet et al 1979).

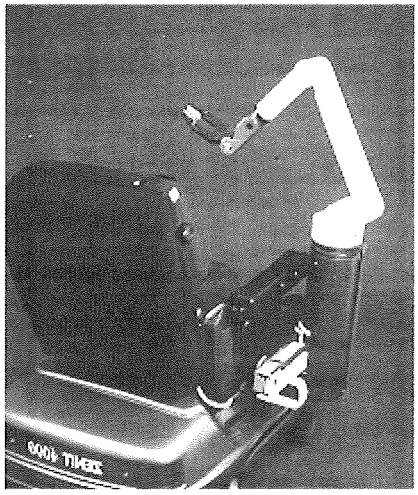


FIGURE 2.1. THE MANUS MANIPULATOR

The design employs an articulated arm on a telescoping base with a combined mass of 20 kg, providing a reach of 88 cm and a payload of 1.5 kg. Rounded appearance and light weight is

achieved by using aluminium castings and carbon fibre tubing, these house the DC motors with associated gears and belt drives. Slip couplings are employed on a number of the joints, limiting the torque that may be exerted, and hence increasing system safety. The control of the system is managed by a control box mounted to a wheelchair. This houses an 80186 processor, transducer interfaces, a power supply and communication interfaces. The standard input devices for the Manus arm are a keypad and a joystick, with feedback being provided by a small LED display. The arm is operated by moving the end-effector through Cartesian space, with pitch, yaw, and roll also possible. The cost of the basic system is approximately $30,000. For additional technical details see Kwee and Duimel (1989).

Preliminary evaluations of the Manus arm were undertaken in Canada (Milner et. al., 1992), France (Brelivet, 1992), and Norway (Oderud and Bastiansen, 1992), by users with a range of disabilities including muscular dystrophy, cerebral palsy and spinal injury. A large number of activities were successfully undertaken with the arm, including feeding and drinking tasks; manipulating objects such as knobs and dials; and picking and placing objects such as books and video tapes. A summary of these evaluations (Verburg et. al. 1996), describes the users as unanimously finding the use of the arm enjoyable. However, a number of difficulties were encountered, which significantly restricted the number of users willing to participate in more extensive tests that would involve the Manus arm being attached to their wheelchairs for up to four week periods.

Reasons identified for this included :

- The size and bulk of the arm effecting the mobility of the wheelchair.
- Incompatibilities existing between the Manus control requirements and the wheelchair control system.
- Insufficient margin between effort to control the arm, and the return for that amount of effort
- The strength and fine control of finger movement that the standard input devices required.

These results led to further work being undertaken to improve the mounting system, the interface, and methods for integrating the Manus arm with wheelchair control systems. Development was undertaken by a Dutch company, Exact Dynamics, receiving funding from the Dutch government via the public health insurance company. Currently over fifty units have been sold, creating a large Dutch user group of about forty individuals, mainly with

muscular dystrophy, but also spinal cord injury, and multiple sclerosis. Feedback from the user group has resulted in findings similar to those reported above (see Stuyt 1997). The users reacted to the system positively, though still encountered some difficulties with wheelchair mobility. Frustration was expressed regarding the time required to complete tasks with the arm. The time required to learn to use the arm effectively can range from one hour to three months depending on the individual's ability and motivation. A desire was expressed by the users for the existence of pre-programmed routines, and the ability to lift heavier loads. An evaluation of the system by 14 individuals reported by Oderud (1997), reiterates the size problems, as well as lack of programmable routines. The study also raises the issue that the usability of the device should be improved. Nevertheless, as with all evaluations, a positive response was encountered, and users stressed the fact that a positive psychological effect results from being able to perform tasks independently.

A more recent evaluation of MANUS was performed at Lund University Hospital (Eftring & Boschian, 1999). User trials involved eight users for 3-4 hours per day for 1-2 days undertaking tasks such as pick and place and drinking tasks. User feedback suggested that the arm was too large, too heavy and difficult to control. One of the 8 was keen to obtain a MANUS for use at home, with 4 more stating they would be interested if improvements were made. The main improvements being smaller and lighter design, possibly mounting on the back of the wheelchair, and simplified control.

2.3 HANDY-1

The HANDY-1 system, illustrated in figure 2.2, was developed as a dedicated feeding aid, by modifying a low-cost educational robot, the Cyber 310. The system has had more commercial success than any other rehabilitation robotic device, with over 140 units sold. However, the success of the project should also be judged by the numerous accounts of the valuable independence users have gained at meal-time, and the therapeutic effects the device offers, that are now coming to light.

FIGURE 2.2. THE HANDY 1 FEEDING AID

The project was initiated by Mike Topping, a student at Keele University in 1988, to enable a 12 year old boy with cerebral palsy to eat unaided. By 1992, the system had evolved into a commercially available product, marketed by a company based at Keele science park, namely Rehabilitation Robotics Ltd. The Cyber robot is a 5-axis arm, weighing 15 kg, and offering a repeatability of 1.5 mm. The arm is fairly compact at 51 cm height, with a length when fully extended of 90 cm. The principal modifications made to the arm were the replacement of the gripper with a spoon holder, and doubling the payload to 500 g. The arm has been provided with a suitable cover, and mounted on a small, portable base stand. A tray is provided that can contain the prepared food in seven separate sections. A simple LED scanning system is used to allow the user to select food from one of the sections by slight movement of a control switch. This 'no frills' approach has resulted in a system cost of £4750, including assessment for suitability, delivery, training, and a 1 year call-out service contract (Topping 1996).

Initial evaluations of the system (Topping, 1993), report an extremely positive response by users. A level of independence, often never previously experienced, is gained by the user being in control of the pace of a meal, and the choice of separate items of food. A more recent study (Smith and Topping 1997), supports these early findings with a questionnaire completed by a random selection of 22 Handy 1 users. Reference is made to the dignity that the system allows the user, in what previously had been regarded as a humiliating situation. A positive response is also elicited from carers, who enjoy the fact that they are now able to eat at the same time as those they care for. From a therapeutic point of view, a number of users have improved control of their head positioning, since the arm is consistent with its delivery of the food (carers are able to compensate for poor head positioning). An

improvement in hand-eye coordination is also claimed, resulting from operation of the control switch. Improvement of oral motor control has also been reported, and verified in a study at Wayne State University (Erlandson et. al., 1995).

Partly in response to requests from the existing user group, work is currently being undertaken to extend the functionality of Handy 1. A project referred to as RAIL (Robotic Aid to Independent Living), is being undertaken as part of a TIDE initiative (Technology Initiative for Disabled and Elderly people - a European Union funding initiative). As described by Topping et. al. (1997), additional tasks being addressed include shaving, grooming, and make-up application. The RAIL project has added positional feedback sensors and more sophisticated control algorithms, allowing more accurate control of both trajectory and position (see Bolmsjo et al., 1997). However, no fundamental modifications are being made to the kinematic configuration of the arm, which remains based on the fairly restrictive Cyber 310 design. Instead of attempting to evolve the system into a general-purpose arm comparable to the Manus arm, researchers are developing a number of light-weight interchangeable attachments, such as, a washing attachment that may hold a sponge, a toothbrush, or a shaver. Clinical evaluations of the RAIL system have yet to be reported.

2.4 The Wolfson, Wessex, and Weston systems.

Research into rehabilitation robotics has been active at Bath Institute for Medical Engineering since the mid 1980's, and provides a clear example of evolving technology. Initially a commercially available robot was employed in a fixed workstation, this was replaced by a purpose-built arm, which was eventually mounted on a mobile platform. Current investigations include the development of a wheelchair-mounted system. At each stage of project development potential users have been involved, either by questionnaire, or system evaluation. Unlike the Manus and Handy 1 projects, this research has not progressed to a commercial product, but has made a significant contribution to defining and understanding user requirements.

The initial workstation system employed a commercially available Atlas manipulator from LJ Electronics, Norwich, UK. System evaluations within a Spinal Injuries Unit allowed appropriate design specifications for a purpose built arm to be determined, these included : 0.5 mm resolution, 1 kg payload, and the ability to traverse the workspace in 5 s (Hillman and Jepson 1992).

It was also concluded that the size and appearance of the Atlas arm were deemed unacceptable. The resulting system, referred to as the Wolfson workstation system, was based on a SCARA design, employing a jointed cylindrical configuration. This was mounted on a desk unit that also contained a cassette tape player, tape storage, computer disk drive, and book storage, around which a number of tasks could be planned. Both direct control of the arm, and the use of pre-programmed routines, were possible. User interaction was via a scanning system and a single or double switch joystick.

The evaluations reported by Hillman and Jepson (1992), elicited a generally positive reaction by users and Occupational Therapists, and better than had been achieved with the Atlas system.

A number of conclusions were drawn :

- better aesthetics would be achievable by either more slender or rounded design;
- ease of use was limited by the scanning system, and most users are capable of using some form of analogue input device;
- the use of pre-programmed routines is important in facilitating robot control, and
- systems would be easier to learn to use if initial configurations offer simple instructions and limited options, which may later progress to more advanced facilities.

There was also the suggestion by Occupational therapists that the system would not be of use as a feeding aid, since the meal-time constitutes an important social occasion. This contradicts the Handy 1 evaluations, perhaps highlighting the diversity of the potential user population, and therefore the diversity of user needs.

Hillman and Jepson (1997), conclude that there were two main reasons why a workstation is impractical for everyday use :

- a desk mounted manipulator is too large for an average home setting, and
- many of the tasks undertaken by individuals were personal care functions, and they wished to perform these tasks in an appropriate place.

Consequently, a project was undertaken to transfer the experience and technology developed to a more mobile, trolley mounted system - referred to as the Wessex system illustrated in figure 2.3.

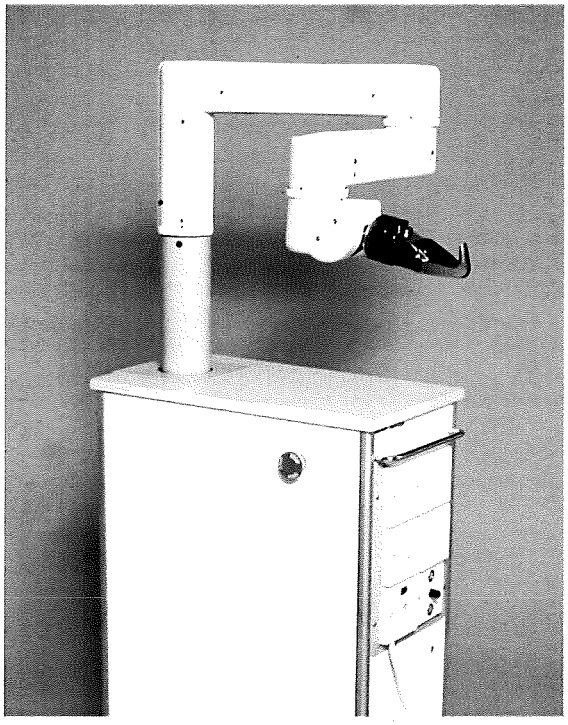


FIGURE 2.3. THE WESSEX ARM

Various improvements to the arm were made, including: appearance, payload (2 kg), and efficiency of the motor drive system. A case study evaluation was undertaken, and is described by Hillman and Jepson (1997). The system was evaluated over a three month period by an individual with spinal cord injury in his home. A wide range of tasks are reported as being successfully undertaken with the arm. The user was able to take advantage of the system's mobility, using it in a number of different rooms, usually placed adjacent to the wheelchair. The user quickly became proficient at using the system, and was soon to be requesting functionality that could not be provided. This seems to reiterate the comment previously made by Occupational Therapists, that systems should be able to be adapted to increase functionality over time. A phenomenon was also reported that occurred in a number of user evaluations with the Manus system : the patience and creativity of the user will result in the system being applied to a number of tasks not originally envisaged by the system's designers.

Current research at Bath includes the development of a wheelchair-mounted system, the Weston arm (Hagen et. al., 1997). The project employs a design similar to the Wessex arm, mounted on a vertical mast attached to a wheelchair as shown in Figure 2.4.

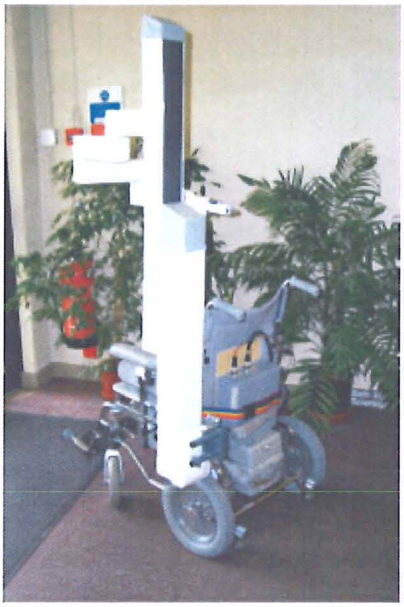


FIGURE 2.4. THE WESTON ARM

The arm's design and positioning attempts to minimize the weight and size impact on the controllability of the wheelchair. Care has been taken with respect to aesthetics, and ease of maintenance. At time of writing, user evaluations of the system are planned.

2.5 The Neil Squire Foundation

The Neil Squire Foundation is a non-profit organisation based in Canada, involved in service delivery and research which addresses the needs of people with severe disabilities. The centre has been involved in the development of robotic technology since the late 1980s, developing a fixed workstation system that operates in a structured vocational environment. The system is referred to as the Neil Squire Foundation Robotic Assistive Appliance (RAA), and is also known by its commercial name : Regenesis. Research at the Foundation has been distinguished by an extensive evaluation of the RAA, which has attempted to quantify the effectiveness of vocational systems. This is in contrast to the majority of evaluations of previous workstation systems, that have tended to focus on subjective issues.

Most of the work discussed so far in this chapter has resulted in systems with some form of mobility. The RAA is unequivocally a fixed workstation system, and builds upon the

experience of a number of similar projects that were the focus of rehabilitation robotics in North America throughout the 1980s and into the 1990s. The most prominent of these is the DeVar system (Van der Loos and Hammel, 1990), which employs an industrial robot arm, the PUMA 260, mounted on an overhead track. Although evaluations of DeVar have been successful (Hammel et. al. 1992), migration of systems into the real world has been hampered by high cost. Only three systems have been built to date, at an estimated cost of $100,000 per system. The Neil Squire Foundation has attempted to meet user needs at low cost, by developing a purpose built robot arm. The initial requirements analysis resulted in the following objectives (Birch, 1993) :

- low cost;
- ease of use;
- functionality based on user needs;
- full programmability;
- portability;
- safety;
- flexibility in configuration, and
- reliability.

The resulting system has 4 rotary and 2 linear axes, a payload of 2.2 kg, and a mass of 8 kg. Potentiometer feedback is used for closed loop PID control, providing a resolution of 0.73 mm for the linear axes, and 0.33° for rotational axes. The motor control system employs a Motorola 6809 CPU, communicating to a PC based user interface. User interaction has been via a standard keyboard, with the assistance of a handstick, mouthstick, or headstick. An expanded keyboard has been used by those with poorer motor control. The estimated cost of the robotic system is $23, 000. However, the total cost of the workstation system including a special desk, computer adaptations, and architectural modifications, is estimated at $35,000 (Birch et al., 1996).

A formal evaluation of the system was undertaken by seven severely disabled individuals, and a number of able-bodied attendants, as described by Birch (1993). An experiment was designed that allowed the subjects to undertake a word-processing based task using two similar workstations, only one of which contained the RAA. The nature and length of interventions required by the attendants was measured, as was the overall productivity, in order to gauge the effectiveness of the RAA. Results showed that the workstation with the

RAA required significantly fewer attendant interventions, however, this was offset by significantly lower productivity of the RAA based system.

A discussion of the results (Birch et al., 1996), states that the number of interventions was fewer with the RAA based workstation, as filing and printing tasks could be handled without the attendant's assistance. The suggested cause of lower productivity, was that operation of the RAA is slower than waiting for and receiving assistance from an attendant. In a real working environment, it is assumed that attendants would be co-workers. Therefore, in order to estimate the effectiveness of the RAA it is necessary to estimate the loss of productivity caused by disturbing a co-worker, and the loss of productivity of the disabled worker whilst waiting for attendance. If the cost of disturbance and waiting were high, then the RAA based workstation could be argued as being more cost effective, even though the overall productivity is lower. It is not difficult to see some fundamental problems in this approach. Office-based work that requires a significant amount of word-processing can usually be arranged such that productivity does not grind to a halt when a worker is waiting for a printout. Additionally, many office environments are such that trips to a shared printer by somebody in the office is frequent. There is also a drive by some companies to create 'paper-less' offices where possible. Each of these issues may serve to dissuade an employer (or government) from purchasing the systems - a fact that seems to have been borne out over the last decade.

2.6 Other Rehabilitation Robotic Systems

As the projects and system evaluations described above are fairly representative of the field as a whole, the remaining active or recent projects that are particularly relevant to this thesis are covered below with a little less detail.

A project referred to as RAID (Robot to Assist the Integration of Disabled people) has had objectives and an approach similar to the RAA described above. A workstation system was developed to address vocational tasks, employing an arm that was developed for rehabilitation or light industrial applications - the RTX by Universal Machine Intelligence Ltd, UK as shown if figure 2.5.

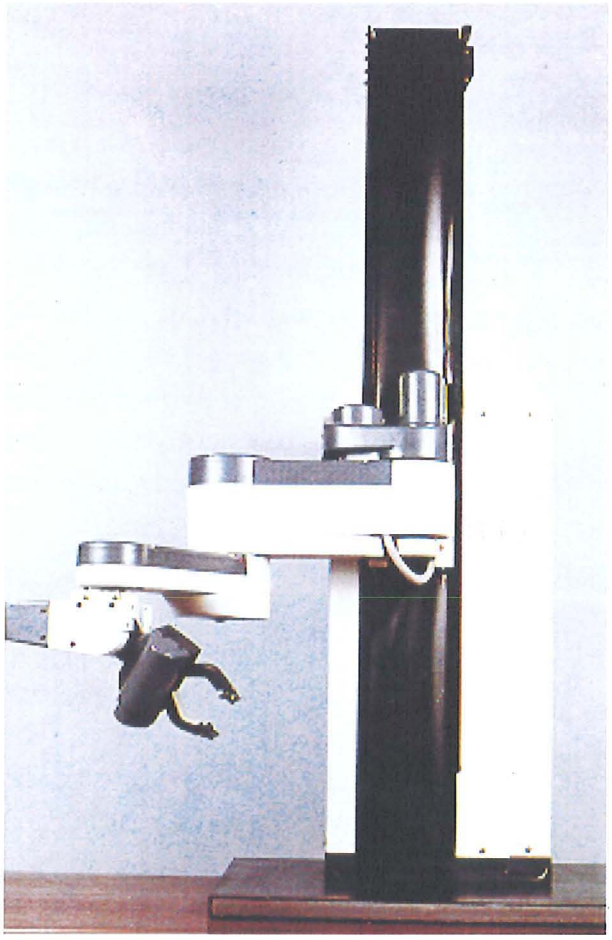


FIGURE 2.5 THE RTX ROBOT ARM

Like the RAA, and DeVar, RAID has undertaken successful evaluations (see Danielsson and Holmberg (1994)), but its high cost at $55,000 per workstation, has been central in restricting its deployment to clinical evaluations. As with the RAA, user feedback was generally positive, though tasks were regarded as being fairly slow. Common to most system evaluations, improvements to the user interface were suggested - particularly in terms of available input devices. The project is currently being progressed under the EPI-RAID acronym as part of a TIDE initiative. This employs a more recent robot based on the RTX (RT200, Oxford Intelligent Machines Ltd, UK), a new programming language CURL (see Mahoney et. al. (1992)), and a modified workstation offering greater reliability than the first. Reports of evaluations of the current system are expected in 1998 - greater system usability is anticipated, but lower cost is not.

A company called Kinetic Rehabilitation Instruments in the USA has developed the Helping Hand - a wheelchair-mounted robotic arm. Simplicity has been central to system design, resulting in an arm significantly cheaper than the Manus arm at $9,500. The 5 degree of

freedom arm has also addressed the size and weight problems of the Manus arm : at 11 kg the arm adds just 1 inch to the width of the wheelchair. Simplicity has restricted how the arm may be controlled : the arm is operated one joint at a time by a joystick. Closed-loop control is not implemented, and cartesian movement or pre-programmed routines are not available. However, evaluations have demonstrated that a large number of daily living tasks could be undertaken with the system (Sheredos et. al., 1996; Sheredos et. al. 1997).

A number of projects have investigated the use of pneumatic actuators for robot control, as they provide a relatively low-cost and safe form of actuation. However, trials of devices by Prior et. al. (1992), and Mattie and Hannah (1994), identified a number of difficulties including : the control of excessive sway and drift, the bulkiness of the actuators affecting aesthetics, and an unacceptable level of noise from the air compressor. More recently, work at the Queen Alexandra Centre for children's health in Canada has improved on one of the original pneumatic devices - the Inventaid Arm developed by Jim Hennequin of the Papworth Group UK. This new device, the QA manipulator (Mattie J., Hannah R., (1995)), has an improved control system, but has retained unacceptable appearance and noise levels.

2.7 The cost/benefit argument

Mahoney (1997), has estimated the cost and number of units sold, of several rehabilitation robotic systems. The major purchasers have been identified, as summarised in table 2.1 below. The Handy 1 system has clearly been the most successful commercial venture to date. This is demonstrated not only by the significantly greater number of units sold, but also by the fact that systems have been purchased and are owned by individual users (though purchase was often with the assistance of local councils and charities).

Product	R&D Support	Approx Cost	Approx # sold	where sold
DeVar	VA Palo Alto Stanford Univ.	$100,000	3	Clinical evaluation
Manus	IRV, TPD	$35,000	50	Various
Handy 1	Keele Univ.	$6,000	140	Individuals
Helping Hand	KRI	$9,500	10	Clinical evaluation
Inventaid	Papworth Grp.	$8,000	5	Not known
RAID	TIDE	$55,000	9	Clinical evaluation
RAA	Neil Squire Foundation	$23,000	7	Clinical evaluation

Table 2.1 Rehabilitation Robotics - commercial endeavours

(Reproduced from Mahoney, 1997).

The Manus arm, which comes second in terms of number of units sold, has typically been bought either by research centres, or via a funding scheme involving the Dutch government - a scheme which has since been replaced (Verburg et. al., 1996). Given that Handy 1 has less functionality than its competitors, it would seem to be the case that the cost of systems must be kept low if they are to be successful. However, as discussed in section 2.5 above, researchers have attempted to justify the high costs of vocational systems by using a cost/benefit analysis, relying on assumptions that would be unlikely to convince potential funding bodies. A return-on-investment analysis has also been made in support of the Manus system (Styuyt, 1997). Manus evaluations have shown that the system was used for 2 hours per day on average. Styuyt equates this 2 hours to a reduction in care requirements of 2 hours per day, and argues that this would lead to a return-on-investment in one year. However, 2 hours of system use per day is not equivalent to 2 hours reduction in care. The number of interventions required by a carer is dependent on the nature of the user's disability, what they are using the system for, and how experienced they are at using it. Additionally, in many cases, carers are unpaid members of the family. Ultimately, the success of the Handy 1 system has been the improvement in the quality of life of its users, not any financial savings, and the shape of the field as a whole suggests that future systems will need to be at a comparable price to emulate or improve on this success.

2.8 System Mobility

The failure to win the cost/benefit argument in the eyes of potential employers has meant that more success has been achieved with systems that address activities of daily living, as opposed to vocational systems. This is the domain of the mobile (or at least portable) system, and trade-offs exist when determining how this mobility is achieved. Attaching an arm to an electric wheelchair provides an immediate and potentially extensive degree of mobility. However, this imposes severe restrictions on the weight and size of the manipulator - a problem as yet unresolved with the Manus system, though successfully addressed by the less sophisticated Helping Hand. Evaluations have also suggested that aesthetic design is more of an issue for wheelchair-based systems : an arm attached to a wheelchair is very closely associated with its occupant, as evident from feedback concerning the Weston arm (Hagen et. al, 1997). Additionally, designing a system for an electric wheelchair restricts the potential user group to those who posses an appropriate electric wheelchair. As pointed out by Verburg et. al. (1995), some wheelchair designs do not allow mounting of the Manus arm. One of the principal disadvantages of systems such as the trolley-mounted Wessex Arm and Handy 1, are that a carer is required to position the system appropriately for any activities undertaken. While the field awaits the maturity of fully autonomous systems, evidence suggests that a valuable area of research is the development of systems that could be mounted either on a wheelchair *or* a mobile platform, thereby combining the benefits of both approaches.

2.9 System performance

A direct comparison of performance characteristics of different robotic systems provides limited information as to the impact of their relative performance levels. Certainly there is no correlation between accuracy or payload and user acceptance. The HANDY 1, with a repeatability of 1.5 mm and a payload of 0.5 kg, provides lower performance than the remaining systems, but has achieved greater success. It is more informative to consider performance in terms of the tasks that specific systems are designed to address. The MANUS system has a payload of 1.5 kg, but has received user feedback suggesting that this should be increased. This request results from the fact that a general-purpose robotic system requires a greater payload than a dedicated feeding aide.

The minimum performance characteristics required of a robotic system may therefore be defined as those required to successfully undertake the tasks that the system is designed to address. This allows for the development of a base-line product that has the potential for user acceptance, providing that it can be marketed at an appropriate price. Evaluations of the MANUS and Wessex systems, suggest that increased performance will always be requested, even if the original tasks identified can be completed. However, this is typical of most consumer products. For example, wheelchair design has made significant ergonomic advances since the invention of the wheelchair, but these improvements were not a pre-requisite of user acceptance or commercial success.

2.10 System functionality

Following its commercial success, user feedback from the HANDY 1 project quickly highlighted the need for general-purpose robotic systems. This echoed a prediction made by Finlay (1988), stating that the projected UK sales for a proposed 'fetch and carry' robot priced at £10, 000 could be 170 units per year. Due to its limited functionality, the HANDY 1 project has been unable to repeat its success as a feeding aide when applied to any other task, and has therefore only scratched the surface of a potentially large assistive technology market.

The concept expressed above of developing a base-line product, the performance of which may be progressed through time to meet a greater number of user's needs, is also applicable to system functionality. An example of evolving technology towards a solution is provided by the projects undertaken at BATH University as described above. Each project phase builds on the lessons learnt from the previous, and where designs incorporate limitations, these are abandoned or modified accordingly. As the field of rehabilitation robotics is maturing, a significant measure of any system's value is not its current level of user acceptance, but its potential for being evolved into a system with greater user acceptance. This requires not designing limitations into the system at early stages in the project (or inheriting limitations as is the case with HANDY 1).

The situation for the MANUS arm may be regarded in some respects as being the reverse of that of HANDY 1. The MANUS is a general-purpose device, and as mentioned above, its expense has limited its success. This suggests that a sensible course of progress for MANUS should be back towards a base-line product, with research investigating whether aspects of

the system have been over-engineered, and whether costs can be reduced. However, this is not the natural course of evolution for any product, and has not as yet been pursued for MANUS.

2.11 The User Interface and Control System

The extreme diversity that exists within the potential user population of rehabilitation robotic systems, has led evaluations to call for wider ranges of available input devices. In response, the M3S TIDE initiative is attempting to develop an interface standard, which includes a Bus system to which any M3S compliant device may be attached (see Overboom et al., 1997). Problems can result where the original design of a system does not anticipate such adaptability. As reported by Kwee (1994), the functionality and flexibility of the Manus system was severely restricted when incorporated within an M3S system. User diversity has also been a factor in highlighting the need for systems that can be configured to present an appropriate level of functionality. As discussed in the study by Hillman and Jepson (1992), systems should be capable of being re-configured as the requirements and experience of specific users change over time. The evaluations outlined above have also indicated that the usability of systems is enhanced when a number of different control modes are available, i.e. joint, cartesian, pre-programmed positions, and pre-programmed routines.

2.12 Design Criteria

The above examination of the strengths and weaknesses of extant rehabilitation robotic systems is used here to define guidelines for the development of future systems. This chapter has discussed how the Handy 1 system has successfully addressed a single task at low-cost, but notes that it is restricted from becoming a general purpose manipulator due the limited functionality of the robot arm employed. Manus has demonstrated the value of general-purpose manipulators, but has been restricted in its success due to its high-cost, limited interface and control options, and its physical size. Helping Hand provides an example of engineering specifications that address the size problems, and the Wessex system has

demonstrated the value of mobility and the need for control options that include pre-programmed routines.

The points argued in sections 2.7 to 2.11 may be summarised as follows. If a system is marketed at too high a cost, then user-uptake will be severely restricted, irrespective of the system's functionality. Systems should be designed to address a range of user tasks, and the minimum performance characteristics required of any system may be defined as those required to successfully undertake the specific tasks addressed. The base-line performance and functionality of systems should be modifiable, such that systems may evolve to meet changing user needs and attitudes. A degree of system mobility should be provided, and flexibility should be inherent to the user interface and control system. Finally, the appearance of the resulting system has to be acceptable to potential users.

The design guidelines are formulated as follows :

- low-cost should be prioritised;
- the system should be of general purpose, providing functionality that addresses a range of user needs;
- base-line performance characteristics should be derived from the requirements of the user tasks that are addressed;
- The design should facilitate future modifications to improve system performance and functionality;
- a form of system mobility/portability should be provided;
- operation should be possible with a wide range of user input devices;
- a variety of control modes should be available;
- ease of use should be enhanced by allowing systems to be configured to match individual user needs;

- the system should have an acceptable appearance, and

- the system should allow for safe operation.

The following chapter examines the design of the Middlesex Manipulator prototype, and illustrates the application of the guidelines outlined above to the design of a user interface and control system for the prototype.

Section II

Manipulator and motor control system design

Chapter 3

The Middlesex Manipulator

This chapter discusses the background to the development of the Middlesex Manipulator. This may be regarded as consisting of three phases :

Phase I **User Requirements & Consequent Mechanical Design Specification**

The user requirements analysis and system design specification included a user survey (Prior, 1990), and a novel manipulator design. This work was undertaken at Middlesex University by Dr Steve Prior (Prior, 1993), and is reported in this thesis as background material.

Phase II **Construction**

An implementation of Prior's design was undertaken at Middlesex University by graduate students under the supervision of Peter Warner, and later under the supervision of the author. This resulted in a prototype employing DC servo-motors, replacing an earlier pneumatic prototype developed by Prior.

Phase III **User interface design, Control system design & System Evaluation.**

The design and implementation of a control system and user interface for the Middlesex Manipulator prototype, and its subsequent evaluation, was undertaken by the author. This is introduced in this chapter, and described throughout the remainder of this thesis.

3.1 Defining User Requirements (Phase I)

An analysis of user requirements was performed by Prior (1993), as the first stage of product development. This consisted of a review of world rehabilitation robotics research (Prior, 1989), and a survey of potential users (Prior, 1990), expanding on and updating a similar survey performed by Clay and others (Clay et. al., 1987). The survey of 50 individuals with various disabilities, identified the activities that were either difficult or impossible to perform, and established a number of tasks that people would wish to undertake with a robotic device.

Personal Hygiene Tasks	**Leisure and Recreational Tasks**
(% with Difficulty + %Not at all)	**(% with Difficulty + %Not at all)**
88% Washing Hair	58% Pick-up and Throw Objects
80% Rearranging Clothes After Toilet	54% Opening a Wine Bottle
68% Cleaning After Toilet	52% Gardening
54% Combing Hair	46% Shooting
54% Shaving/Makeup	44% Playing Snooker/Pool
Domestic Tasks	**Working Environment Tasks**
(% with Difficulty + %Not at all)	**(% with Difficulty + %Not at all)**
84% Cooking	48% Opening a Letter
82% Preparing Food	48% Using a Stapler
78% Filling the Kettle	46% Posting a Letter
78% Opening/Closing Windows	44% Pick and Place Objects
70% Pouring Water/Milk	44% Filing Documents

Table 3.1 Most Important Task Lists

Prior employed a weighted matrix method (Middendorf, 1986), to order the tasks dependent upon the cost, control complexity, accuracy and payload that they would be likely to require. This was achieved by assigning each of these criteria a weight corresponding to an estimate of its importance relative to the other criteria. Each of the tasks were judged against the criteria, and awarded a score. The tasks with the highest scores should in theory be the easiest to incorporate

into the design of the manipulator. The results acted as a prioritized task list, on which the design specification was based. These are shown in table 3.2 below. The tasks are then listed in table 3.3 in order of score. Estimates are also shown of the number of degrees of freedom (D.O.F) the manipulator is likely to require in order to undertake the tasks.

Criteria (weight)	Cost (0.4)	Complexity (0.3)	Accuracy (0.2)	Payload (0.1)	
					Score
Personal Hygiene					
Washing Hair	0.1	0.1	0.2	0.15	0.125
Re-arranging Clothes	0.25	0.2	0.2	0.2	0.22
Cleaning after the Toilet	0.1	0.15	0.15	0.2	0.135
Combing Hair	0.3	0.3	0.2	0.2	0.27
Shaving/Makeup	0.25	0.25	0.25	0.25	0.25
Domestic Tasks					
Cooking	0.2	0.2	0.2	0.15	0.195
Preparing Food	0.1	0.1	0.1	0.25	0.115
Filling the Kettle	0.2	0.2	0.2	0.15	0.195
Opening/Closing Windows	0.2	0.2	0.25	0.2	0.21
Pouring Liquid	0.3	0.3	0.25	0.25	0.285
Leisure & Recreational					
Pick-up & Throw Objects	0.1	0.2	0.3	0.15	0.175
Gardening	0.3	0.2	0.25	0.2	0.25
Opening a Wine Bottle	0.2	0.2	0.15	0.2	0.19
Playing Pool/Snooker	0.2	0.25	0.15	0.25	0.21
Shooting	0.2	0.15	0.15	0.2	0.175
Working Environment					
Opening a Letter	0.15	0.15	0.15	0.2	0.155
Using a Stapler	0.15	0.2	0.2	0.15	0.175
Posting a Letter	0.3	0.25	0.25	0.15	0.27
Pick & Place Objects	0.2	0.2	0.2	0.2	0.2
Filing Documents	0.2	0.2	0.2	0.2	0.2
OtherTasks					
Drinking	0.3	0.3	0.25	0.15	0.275
Painting	0.3	0.3	0.25	0.25	0.285
Writing/Typing	0.15	0.15	0.15	0.2	0.155
Showering	0.1	0.1	0.2	0.15	0.125
Creaming	0.15	0.15	0.15	0.25	0.16
Top Five Tasks					
Reaching & Gripping	0.2	0.25	0.25	0.2	0.225
Pick & Place from Floor	0.2	0.25	0.2	0.2	0.215
Eating/Feeding	0.3	0.2	0.15	0.3	0.24
Dressing	0.15	0.1	0.2	0.2	0.15
Pickup Large/Heavy Object	0.15	0.2	0.2	0.1	0.17

Table 3.2 Weighted Matrix Results

Task	Score	D.O.F.
Pouring liquid	0.285	4
Painting	0.285	5
Drinking	0.275	4
Posting a letter	0.270	4
Combing hair	0.270	5
Gardening	0.250	5
Shaving/makeup	0.250	5
Eating/feeding	0.240	5
Reach & grip.	0.225	6
Re-arranging clothes	0.220	6
Pick from floor	0.215	5
Open/close windows	0.210	5
Playing pool/snooker	0.210	4
Pick & place objects	0.200	5
Filing documents	0.200	5
Cooking	0.195	5
Filling the kettle	0.195	5
Pick & throw objects	0.175	6

Table 3.3 Highest Scoring Tasks (from weighted matrix results)

The results of the survey, and a process of consultation with disabled people and care professionals, led to the development of the following design specifications. These are grouped into : i) general requirements; ii) design requirements; iii) environmental conditions; iv) ergonomics and aesthetics; v) safety; vi) cost; and vii) life expectancy and servicing.

3.1.1 General Requirements

- The system shall be capable of use by the majority of wheelchair users via several modular user interface options.
- The system shall have either a versatile end effector capable of picking up a large number of differently shaped objects or a tool changing end effector with an on-board selection of different end effectors.
- The operation of the system shall require minimal specialist training.
- The system shall be capable of being mounted to as large a range of wheelchairs as possible without substantial modifications.
- The system shall be able to be fitted on either side of the wheelchair with minimal modifications to the system.
- The system shall be capable of direct control by the operator through visual feedback together with re-programmable memory locations for use with pre-programmed routines.
- The system shall be capable of connection to a personal computer for workstation use.
- The system shall be capable of being easily detached from the wheelchair for either transportation or servicing.
- The operation of the system should not unduly fatigue the operator.
- The system shall be designed to be easy to manufacture, simple to assemble and accessible for repair and servicing.

3.1.2 Design Requirements

- The system shall be capable of lifting at least 1 kg anywhere within its working envelope.
- The system shall have a reach characteristic, r, of ($0.7\ m << r << 0.9\ m$).
- The system shall have an absolute positional accuracy of 15 mm.
- The system shall have a repeatability of 10 mm.
- The system shall have a coarse control speed of 0.2 m/s and a fine control speed of 0.05 m/s for the end point velocity.
- The system shall be able to reach to a zone on the floor, to the front and side of the wheelchair.
- The system shall be capable of reaching to a maximum height of 1.7 m above the floor.

- The system shall be capable of reaching to a zone in front of the operator from head to thigh (normal operating mode).
- The system shall be designed to have a kinematic configuration which under normal use is stiff in the vertical plane and compliant in the horizontal plane.
- The system shall have a total weight of less than 8 kg.
- The system shall be designed to comply with ISO 7176 : Part 1: Determination of Static Stability, and ISO 7176 : Part 2 : Determination of Dynamic Stability of Electric Wheelchairs.
- The system shall be designed and programmed with reference to the top eighteen tasks (listed in table 3.3).

3.1.3 Environmental Conditions

- The system shall be capable of operation within a temperature range of 0-40°C.
- The system shall be designed to prevent the ingress of dust and dirt.
- The system shall be constructed of materials able to withstand contact with chemicals and substances, which it might reasonably encounter during it's working life.
- System noise levels are to be limited to 40 dB at 1 m.
- The system shall be designed for both indoor and outdoor use.
- The system shall be designed to comply with ISO 7176 : Part 9 : Climatic Tests for Electric Wheelchairs.

3.1.4 Ergonomics and Aesthetics

- The system shall have a parked or home position which does not substantially increase the overall size of the wheelchair's width or length.
- The system's height when parked shall be below the height of the wheelchair's armrest.
- The system shall not prevent the wheelchair from passing through a normal doorway.
- The system's power supply shall come from the wheelchair's batteries.
- The system shall be capable of continuous operation for at least 4 hr/day.
- The system shall be designed to conserve energy when static.
- The system shall be aesthetically designed, in terms of form, size, colour, texture and movement.

3.1.5 Safety

- When in operation the system shall be prevented from causing injury to the operator by employing slow speed of operation, low inertia of moving parts, system monitoring and hard stops.
- An emergency stop switch and system reset switch should be provided.
- All external surfaces shall be free from sharp corners and projections.
- The system shall not unbalance the wheelchair when operating at maximum reach.

3.1.6 Cost

- The system shall have a maximum component cost of £1,500 - excluding the cost of interface mechanisms.

3.1.7 Life Expectancy and Servicing

- The system shall not require maintenance for at least the first 500 hours use, with an annual service thereafter.
- The system shall have a total life of at least 6,000 hours.

3.2 Kinematic Design (Phase I)

As described by Prior (1993), the initial conceptual designs for the kinematic arrangement were based on the following five standard industrial robot geometries:

- Articulated (PUMA : Programmable Universal Machine for Assembly);
- Horizontally articulated (SCARA : Selective Compliance Assembly Robot Arm);
- Cartesian;
- Spherical and
- Cylindrical.

Prior notes that the SCARA geometry has increased rapidly in popularity for industrial applications over the passed two decades, having demonstrated significant performance advantages over other

industrial robot designs (see Makino and Furuya, 1982). The concept has also been employed by successful rehabilitation robot designs such as the RTX and Wessex systems. Prior outlines a number of the design's advantages, for example, the major joints do not oppose gravitational forces, and can therefore be of small torque ratings. The arrangement of jointed planar linkages allow the actuators to be either direct-drive, or mounted in-board and driven through belts or chains. This lowers the moment of inertia of the links and the bending moment of the arm about the base joint. The compliant nature of the SCARA robot in the horizontal plane is also an important safety feature when in close proximity to the user. The workspace of the SCARA robot is in the form of a heart shape, which would suit the wheelchair application where there is a need to reach to the user as well operate at the front and side of the wheelchair.

The industrial SCARA robot is mainly designed to perform tasks involving pick, place and insertion operations. The vertical travel is small compared to the large horizontal workspace, and is usually achieved by placing a prismatic joint directly on the axis of the end effector. For rehabilitation applications, there is a similar need for a large horizontal workspace, but there is also a need for a large vertical stroke. Prior argues that using the industrial SCARA geometry and making the vertical stroke at the end effector larger is impractical, due to the related negative effects that the extra size and mass would cause.

In the wheelchair application, the space criteria dictates that the whole of the arm may park in a position that is beneath the armrest and which does not make the wheelchair substantially wider or longer. The high reach characteristic (reach up to 1.7 m) could be achieved with a fixed pillar arrangement, upon which the whole arm was raised, as in the RTX design. However, Prior notes that this would prevent the arm being parked, cause visibility problems for the wheelchair user and would be unlikely to be accepted; and therefore rejected the concept.

An alternative design solution was suggested, combining one or more of the basic kinematic arrangements. Combining the advantages of the SCARA configuration with the vertically articulated arm seemed to give an optimum solution to the twin problems of reach and suitable workspace.

3.2.1 The Scariculated Arm Design

The design solution proposed by Prior (1993), combines the advantage of large vertical stroke from the vertically articulated geometry with the advantage of large horizontal stroke from the SCARA geometry. This was achieved by inserting a 0° ± 90° joint at the beginning of the first link of a standard SCARA design. The arm is thus enabled to reach to the floor (-90° position) in the vertically articulated mode and up to a high reach (+90° position) also in the vertically articulated mode by the use of this extra joint; with the 0° position being the normal SCARA mode. The design consists of seven joints and the end effector grasp (five rotary and two linear). The kinematic arrangement selected for the prototype design is therefore a hybrid combination of the SCARA geometry and the vertically articulated geometry, and is referred to as the SCARICULATED arm geometry, illustrated in figure 3.1.

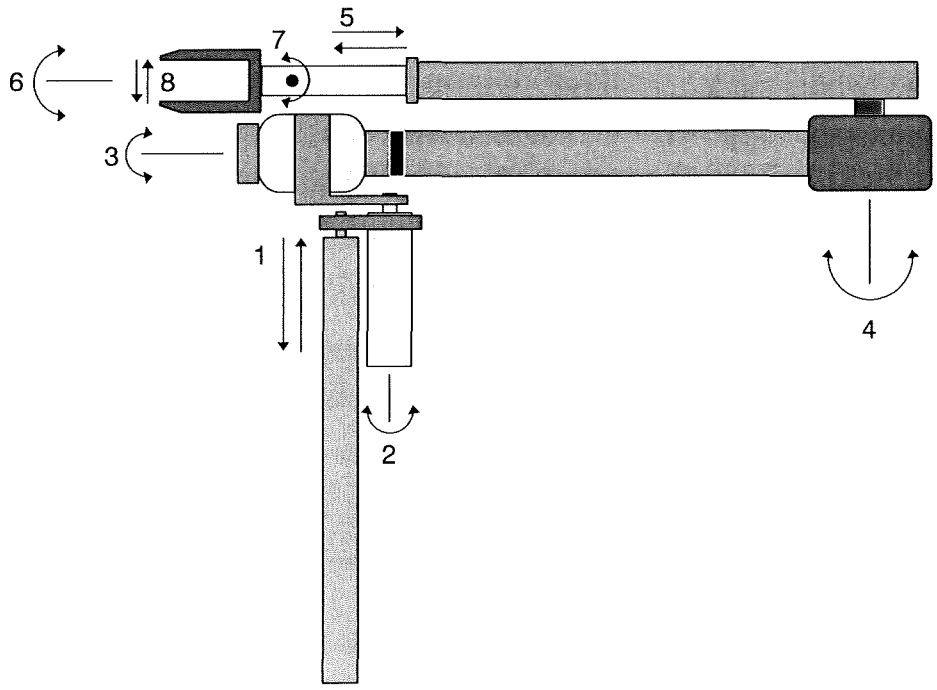


FIGURE 3.1 - THE SCARICULATED DESIGN

3.3 The Middlesex Manipulator Prototype (Phase II)

An early prototype of the Middlesex Manipulator employed pneumatic 'flexator' actuators. Research in the application of these actuators to the field of rehabilitation robotics was motivated by the safety offered by their natural compliance, their low-cost, and their favorable power to weight ratio. As anticipated, the actuators presented a more challenging control problem than DC motors, partly due to friction and hysteresis. However, Prior (1993) reports a number of techniques that can be used to reduce hysteresis. Prior also notes that flexator actuators will be of most use where the miniaturization of actuators is not a requirement. As discussed in Chapter 2, user evaluations have indicated that the bulkiness of pneumatic actuators results in unacceptable appearance. Consequently, the decision was made to employ DC servomotors for the current version of the Middlesex Manipulator.

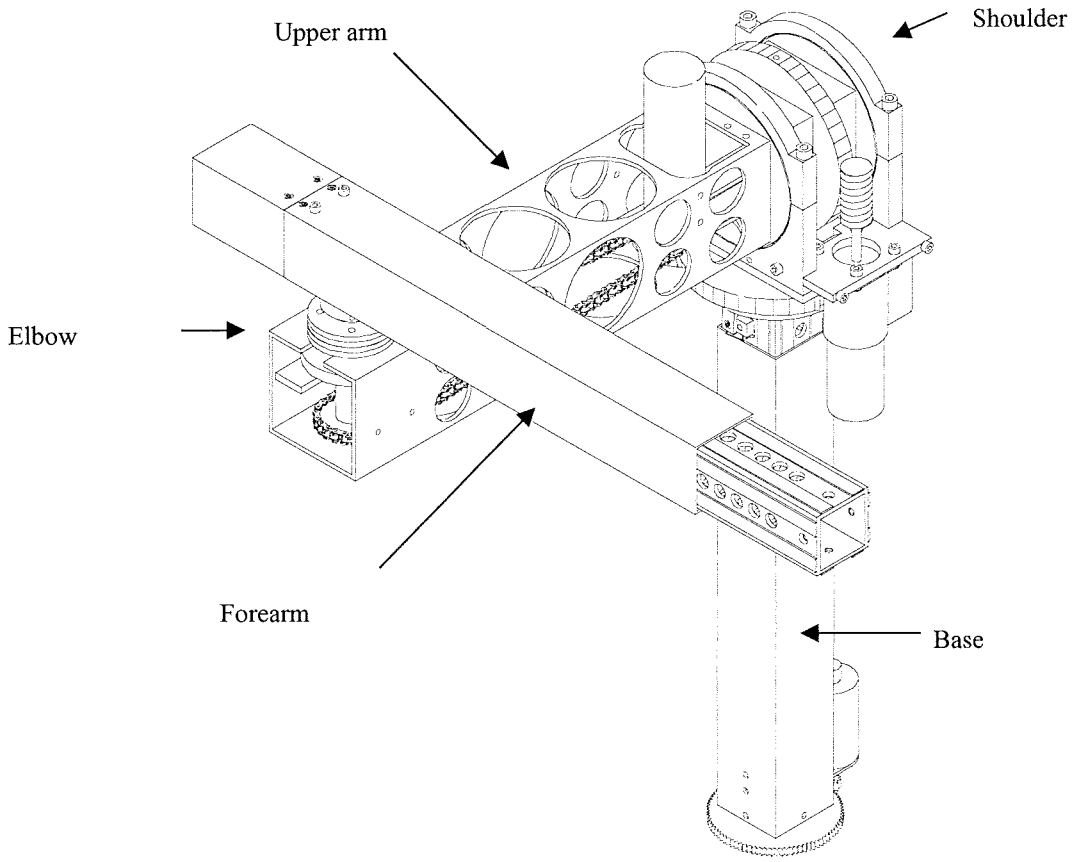


FIGURE 3.2 MIDDLESEX MANIPULATOR - ENGINEERING DRAWING

The mechanical construction of the prototype was initiated by two undergraduate students (Heide and Roorda, 1993), and continued by Dijkstra and Fennema (1994), all under the supervision of Peter Warner, Principal Lecturer, Middlesex University. Assembly and modification of the design was performed by the author and undergraduate students (Buter and Veltman, 1996) under the supervision of the author. The manipulator without end-effector is shown in figure 3.2, with DC motors replacing the original flexators.

The five axes shown include two prismatic axes (base and forearm), and three rotational axes (elbow, and two degrees of freedom at the shoulder). The Upper arm is 360 mm in length, and the forearm is 330 mm, extendible to 530 mm. The overall height of the manipulator varies from 620 mm to 900 mm. The shoulder joint can rotate through 200° in the horizontal plane, and 360° in the vertical plane. The elbow joint can rotate through 315° .

To reduce weight, holes have been drilled in the manipulator's aluminum tubing. Lightweight plastics are employed for the cover, and where possible for gears, and high density polyethylene strips form linear bearings for the prismatic joints. The resulting overall weight is 7 kg (excluding end effector).

A three degree of freedom end effector with detachable fingers is currently under development, and is shown with the manipulator on a temporary trolley mounting in Figure 3.3.

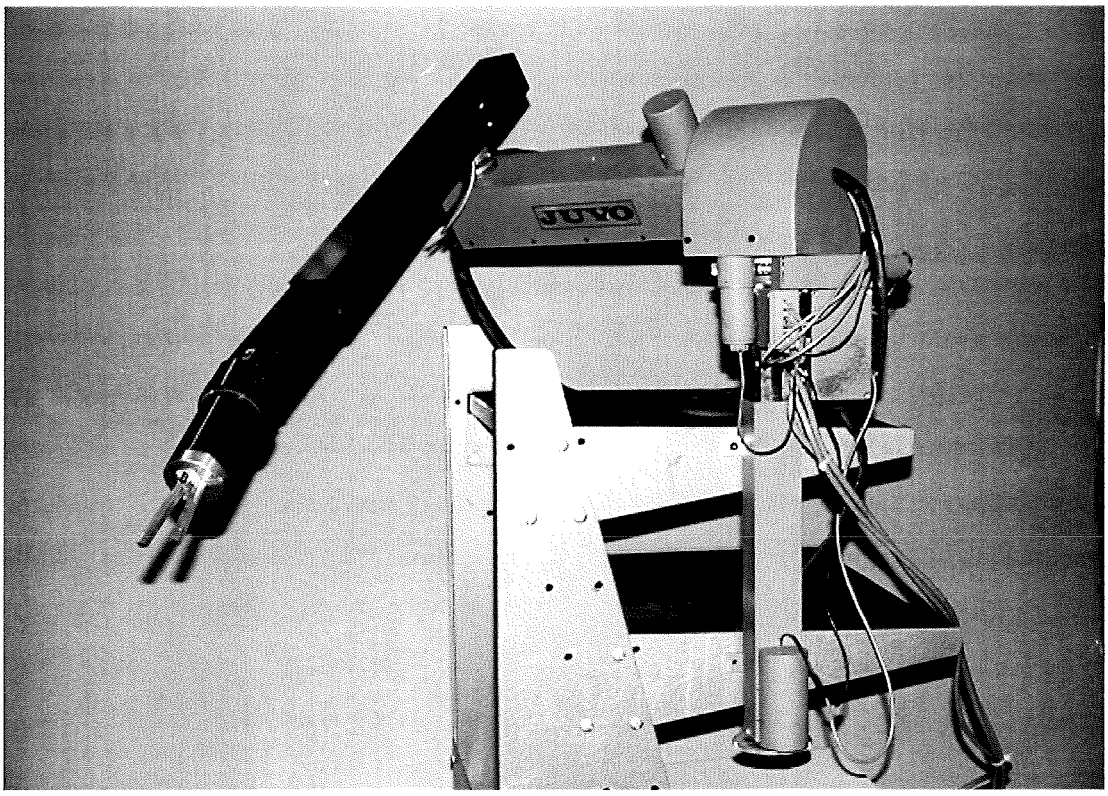


FIGURE 3.3 MIDDLESEX MANIPULATOR WITH END EFFECTOR

3.4 Controller and Interface Requirements Specification (Phase III)

As discussed in Chapter 1, the development of a control system and user interface for the current Middlesex Manipulator prototype is central to the work reported in this thesis. The inputs to the process of generating a requirements specification were :

- the general design criteria derived in Chapter 2;
- the initial Manipulator design specification generated by Prior (1993) and outlined above;
- a review of the field of HCI as discussed in Chapter 5, and
- consultation with researchers, care professionals and potential end-users.

The items that are pertinent to the control system and user interface were extracted from the initial design specification, with the following modifications and additions included. Firstly, the repeatability requirements were tightened, as the original estimates had been based on levels

deemed achievable with pneumatic actuators. The reference to modular interface options in the original design specification is expanded to explicitly refer to the provision of different input and feedback devices, and the possibility of adapting the system in terms of the functionality provided, and the form of user interaction employed. The issue of compatibility is also addressed, particularly with reference to compliance with developing interface standards. A target cost for the manipulator including the control system and user interface is provided, based on the approximate cost of the HANDY - 1 system. Finally, as Chapter 2 discussed the advantages of systems that can be mounted on a wheelchair or a mobile platform, the current requirements specification regards the potential user group as being people with physical disabilities, as opposed to only electric-wheelchair users.

3.4.1 Requirements

G1. System design should address the user tasks outlined in table 3.3

G2. The system should be safe to operate.

G3. The operation of the system should not unduly fatigue the operator (design should address ease of interface navigation, intuitive operation, and minimized likelihood of errors);.

G4. The operation of the system shall require minimal specialist training.

G5. The system should be subjectively pleasing;

G6. The cost of the Manipulator, control systems, and user interface should be low, at approximately £5000, and;

G7. The system should be easy to repair and maintain.

3.4.2 User Interface

U1. The system should allow for operation with a range of different input and feedback devices.

U2. The system should be adaptable, allowing the functionality and interface complexity to be configured to match user requirements.

U3. The system shall be capable of direct control by the operator through visual feedback together with programmable memory locations, and routines.

U4. The system shall be capable of connection to a personal computer for workstation use.

U5. The system should allow for connection to other assistive technology devices through common interface standards.

3.4.3 The Control System

C1. The system shall have a repeatability of 5 mm.

C2. The system shall have a coarse control speed of 0.1 m/s and a fine control speed of 0.05 m/s for the end point velocity.

C3. The system's power supply shall come from the wheelchair's batteries.

C4. The system shall be capable of continuous operation for at least 4 hr/day.

C5. The system shall be designed to conserve energy when static.

C6. When in operation the system shall be prevented from causing injury to the operator by employing slow speed of operation, low inertia of moving parts, system monitoring and hard stops.

C7. An emergency stop switch and system reset switch should be provided.

3.5 User Interface and Control System Overview (Phase III)

Initial design considerations resulted in the proposal of a system architecture as depicted in figure 3.4. A Personal Computer provides the platform for the user interface. This was chosen to provide greater flexibility than an embedded system for interface device development, as required by items G2, U1, U2, U4 and U5 of the requirements specification. However, both power consumption and cost may be increased as a result of not using an embedded system (items G5, C3 &C4). A solution would therefore be to port the system developed on a PC to an embedded PC at an appropriate stage of system development.

The User Interface system communicates with a separate motor control system implemented on dedicated embedded micro-controllers. A dedicated embedded control system with built-in redundancy increases system safety (G2 &C6), and reduces the performance requirements of the PC. With the appropriate choice of micro-controller, this approach would not substantially increase cost. Drive circuitry for the DC servo motors is purpose built, implementing closed-loop position control (C1), and open-loop speed control (C2). Input and feedback devices may be purpose-built and/or commercial dependent on system configuration (U1).

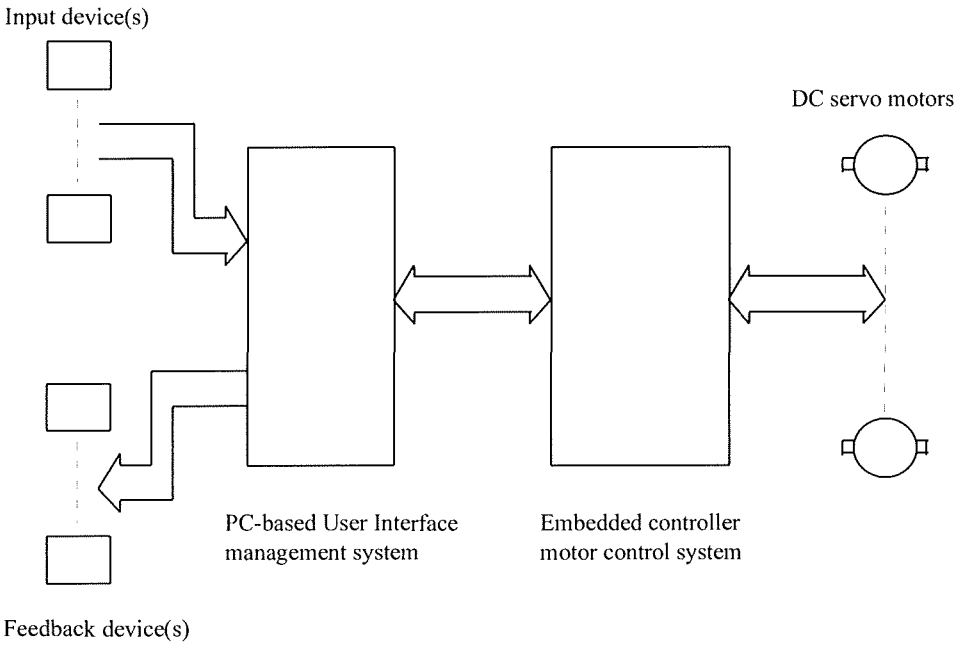


FIGURE 3.4 USER INTERFACE AND MOTOR CONTROL SYSTEM ARCHITECTURE.

3.6 Summary

This chapter described the background to the mechanical design of the Middlesex Manipulator, presenting a discussion of the initial design specification, and a design solution provided by a novel kinematic configuration. A requirements specification for the development of a control system and user interface was generated from consideration of : the Manipulator's initial design specification, a review of rehabilitation robotics, general HCI design issues, and user requirements. A modular architecture is proposed for system realization. The following chapters provide a detailed design description of the hardware and software for each of the system components.

Chapter 4

The Motor Control System

This chapter outlines the design of the Middlesex Manipulator's motor control system. Given the cost constraints and moderate performance requirements, an embedded micro-controller is used to provide closed-loop position control and open-loop speed control for the 8 axes of the Middlesex Manipulator. The system is designed to accept commands from a PC-based User Interface system as described in Chapter 3.

Purpose-built shaft encoders were developed for positional control, reducing the cost of peripheral components. Separate opto-isolated motor control modules were developed with motor control ICs generating Pulse Width Modulated outputs.

A method was devised to allow the micro controller to approximate Cartesian movement without performing inverse kinematic calculations. The resulting performance characteristics are summarised.

4.1 Hardware design

4.1.1 System overview

The following section provides an overview of the motor control system for closed-loop positional control and open-loop speed control.[1] As discussed in Chapter 3, the decision was taken to implement the motor control system using embedded microcontrollers. The 8051 family of microcontrollers was selected, due to the availability of support tools within the University (an emulator and a C compiler), and the previous design experience of the author. The cost of the 8051 is low (at around £5), and its 8-bit architecture results in lower design costs than 16-bit alternatives.

The option was available to implement a motor control module containing an 8051 for each of the Manipulator's axes. However, the cheaper option was selected, of having a single microcontroller for all axes. As described in section 4.2.2 below, it was estimated that an 8051 operating at 12 MHz with an appropriate selection of peripheral components, would provide adequate processing power to achieve the moderate performance required. This could be achieved through the use of programmable timer ICs generating Pulse Width Modulated (PWM) drive signals. A second embedded microcontroller could be included in a separate and simpler module, to provide system redundancy and enhance system safety.

Suitable low-cost motor drive ICs were identified, capable of accepting PWM control signals. These also contained a system-brake input that could be triggered by a motor-current sense facility as a safety option. The brake input also allows for power consumption reduction when the Manipulator is not in motion.

During the construction of the manipulator prototype, multi-turn potentiometers had been mounted for positional feedback for all rotational axes. Sensors had not been implemented for the prismatic joints. To maintain the low-cost approach, the decision was made to develop purpose-built shaft encoders.

[1] Open-loop control of speed was selected as there was no requirement for accuracy in controlling speed, only in providing appropriate limits of speed.

Evaluations of rehabilitation robotic systems have highlighted the need for carers to be able to control or move the manipulator. As carers can not always use the input devices provided, systems such as the MANUS and Helping Hand employ slip clutches that allow the arm to simply be pushed out of the way. However, the current design of the Middlesex Manipulator employs self-locking joints that are cheaper to manufacture, and offer safety when the power to the system is cut. The design option was therefore taken to include a manual control system that can override the embedded microcontroller, operated by pressing buttons mounted on each of the Manipulator's axes. Provision for this mode of operation has been included within the system design, but is not currently implemented.

A power supply module is included, to generate the various voltage levels required from a 12V battery. Power for the motor drive modules, is provided by a 24 V supply, electrically isolated from the remainder of the system. Figure 4.1 illustrates the interconnection of these system components.

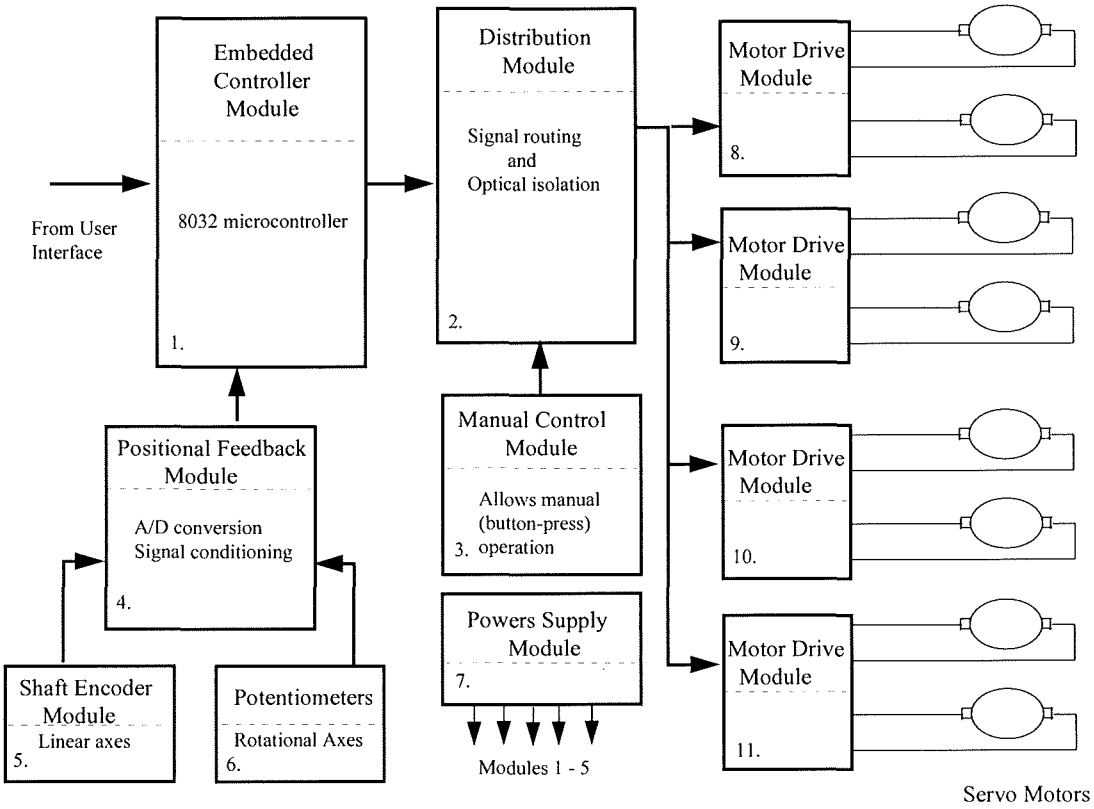


FIGURE 4.1 MOTOR CONTROL SYSTEM OVERVIEW

The following section provides a functional description of each of the system modules.

4.1.2 The embedded microcontroller module

Figure 4.2 shows the main components of the embedded microcontroller module. The 8032 microcontroller was chosen, as opposed to the 8051 which uses internal program memory, and the 8031 which has only 128 bytes of on-chip data memory (the 8032 has 256 bytes).

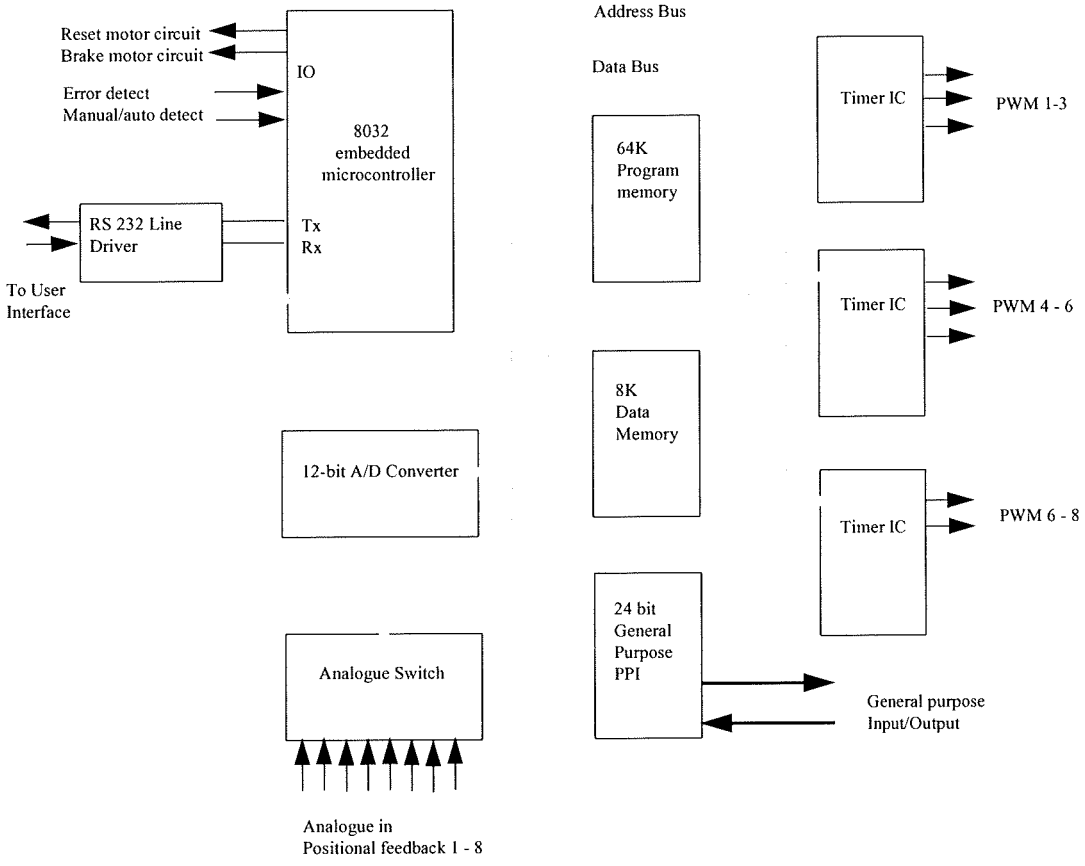


FIGURE 4.2 EMBEDDED MICROCONTROLLER MODULE

Peripheral components were address-mapped, and these include :

- 8254 programmable timer ICs for PWM signal generation. These may be operated by writing a data word to the IC's register. This determines the mark-space ratio of the output pulse at a constant frequency determined by a clock pulse. The pulse train remains unchanged until a new data word is written.

- An 8255 programmable peripheral interface IC for general purpose IO. Twenty four bits of configurable IO are available, 8 of which may be used to set motor brakes for each manipulator axis. Currently a single brake signal is employed for all axes, operated from a single output bit of one of the 8032's ports.

- A 12 bit A/D converter, the HI 5812, allows for conversion of the positional feedback signals. This is incorporated into an 8-bit system by having 2 internal registers corresponding to the lower 8 bits and upper 4 bits, both registers may be individually addressed.

- An analogue multiplexer, the MAX 378, allows the processor to select 1 of 8 analogue input channels. The multiplexer is IO mapped, using 3 bits of an 8032 output port.

- An RS 232 line driver, the MAX- 202, allows for serial communication with the PC-based user interface system. The TTL output of the 8032's serial port is converted to RS 232 voltage levels.

Two of the remaining available bits of the microcontroller's IO ports are used as outputs to brake and reset the motor drive ICs. A further two are used as inputs to detect for auto/manual mode, and the occurrence of a motor drive error caused by a current limit being exceeded. A system interrupt was not employed for error input, as logic circuitry ensures that generation of an error signal would automatically disable all motor drive ICs.

4.1.3 The distribution module

As shown in figure 4.3, a circuit was designed to provide electrical isolation between the motor drive modules and the remainder of the system. One of the project objectives was to develop a modular motor control system to facilitate system repair and servicing, i.e. faulty modules should be easily located, and simply un-plugged for replacement. The distribution module provides some of the signal routing to allow this modularity. The module also allows for the source of the PWM signals to be either the microcontroller, or the manual control module, depending upon mode of operation selected. Finally, a circuit is included to detect low battery power.

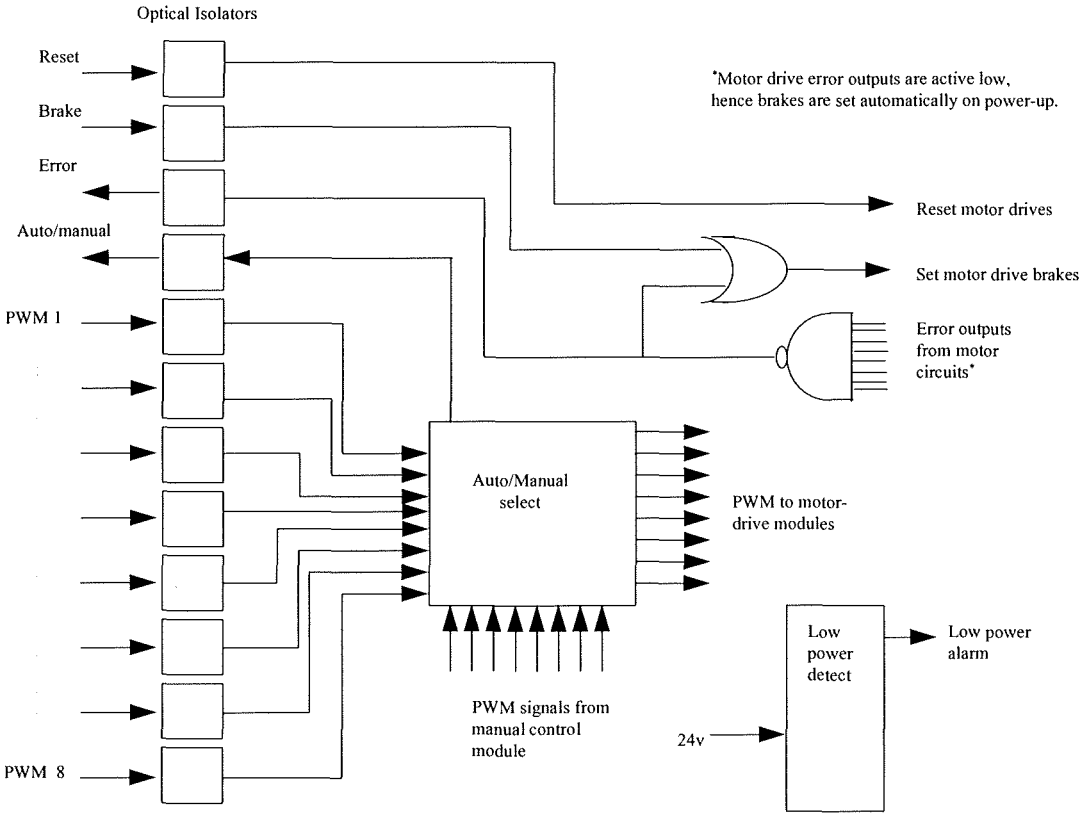


FIGURE 4.3 DISTRIBUTION MODULE

4.1.4 The shaft encoder

To act as shaft encoders two plastic disks were made, on which small reflective strips were mounted. These were used with optical switches for pulse generation.

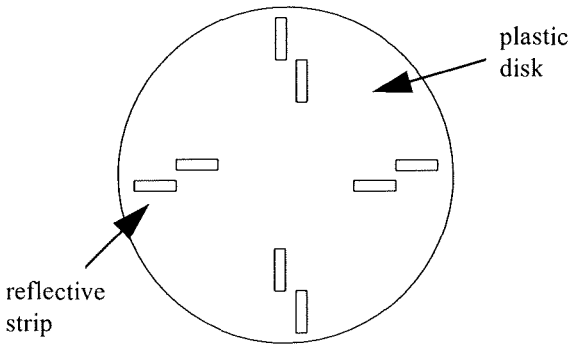


FIGURE 4.4 SHAFT ENCODER DISK

The pitch of the lead screw for the prismatic axes is 2 mm, consequently with four pairs of reflective strips the resolution for control of each of the prismatic joints is ±0.5 mm. The 2 reflective strips of each pair are positioned such that they are detected by two separate optical switches. The order in which the switches detect the strips depends on the direction in which the disk is rotating. The pulse trains generated by the optical switches provide inputs to the shaft encoder circuit. The circuit consists of a 12 bit counter, made up from 3 cascaded 4-bit counters. Figure 4.5 below shows in simplified form, how the up/down and clock signals are generated for the counter circuit.

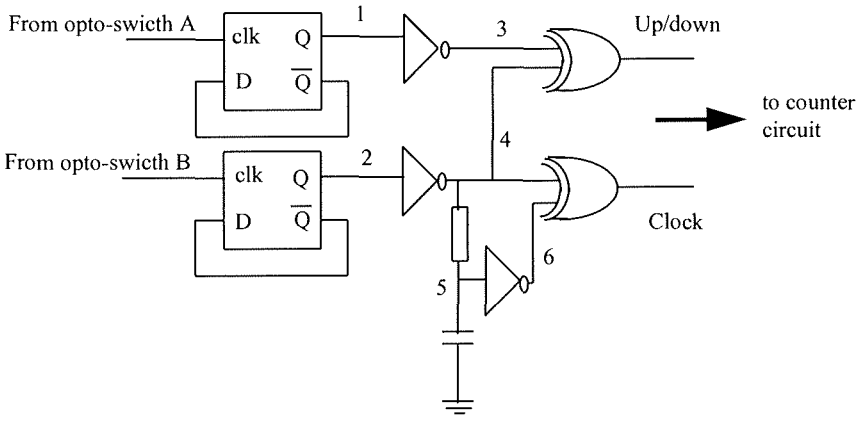


FIGURE 4.5 SIGNAL GENERATION FOR SHAFT ENCODER COUNTER CIRCUIT

The pulse train input to the two D-type flip-flops will be slightly out of phase, with the signal that is leading being dependent on the direction of the encoder disk.

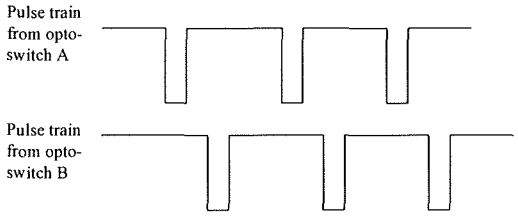


FIGURE 4.6 OPTO SWITCH PULSE-TRAIN

The outputs of the inverters at nodes 3 and 4 in figure 4.5 are toggled by the flip-flops. The up/down signal is normally low and the clock is normally high. Nodes 3 and 4 are toggled slightly out of phase. A short clock pulse is generated at node 6, determined by the RC circuit at node 4. The counter circuits are clocked by a rising edge, so the clock pulse generated occurs when the up/down signal is low if node 4 changed state before node 3, or high if the opposite is true.

The output of the shaft encoder circuit is an analogue voltage produced by a 12 bit D/A converter. This provides compatibility with the signals from the feedback potentiometers of the rotational axis, and therefore aids modularity. The alternative to this approach would have been to use 12 bits of the PPI (programmable peripheral interface) on the embedded controller module to read the counter output. As it was envisaged that a number of uses for the PPI may arise, for example as sensors are added to the system, the current approach was preferred.

All positional feedback signals are routed through the positional feedback module. The module provides simple signal conditioning by way of an amplifier for gain, and a summing amplifier for offset. Low-pass filtering is implemented to reduce the noise pick-up from the DC motors.

4.1.5 The motor drive module

Motor drive is achieved with a motor drive IC, the LMD 18200, which can supply up to 3A to a motor, and accepts a PWM signal as input. The IC may be configured to allow bi-directional control, with a unipolar PWM signal varying from 0 to 100 % mark-space ratio. A current sense output is available. Figure 4.7 shows how this may be applied to a comparator circuit to provide a current-limit facility. The IC also has a brake input that results in the generation of a PWM signal with equal mark-to-space ratio, thus removing motor drive current. Brake is connected to either

+5V or ground, as controlled by a relay circuit. A latch circuit causes the brake to be set if the current limit is exceeded, or if a brake signal is received from the embedded controller. A reset signal is used to toggle the latch output.

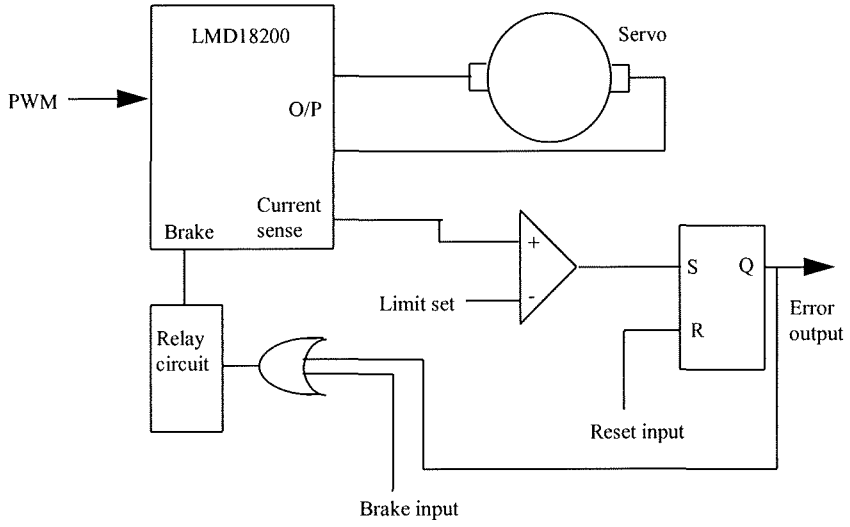


FIGURE 4.7 MOTOR DRIVE CIRCUIT

4.1.6 Motor control system implementation

Printed circuit boards were designed for each of the hardware modules described above, with the exception of the embedded controller which was wire-wrapped. Each module is currently of Eurocard size, and mounted in a Eurocard rack. This form was convenient for system development and testing. However, with the system now functional, a significant degree of miniaturization may be achieved. Some of the fabrication and assembly of the circuits was carried out by an undergraduate student (Gellrich, 1995), under the supervision of the author. All circuits were then tested and integrated into the system by the author. The total cost of components and materials for the motor control system was £440. This figure would be reduced if components were sourced more competitively, and purchases were made in bulk.

4.2 Microcontroller software development

The microcontroller is responsible for lower-level control concepts, such as setting a speed, or moving a joint to a specific position. The algorithms for higher-level control, such as task execution, are implemented on the PC-based User Interface System (UIS). By focusing at this lower level, and analysing both the system requirements specification, and the functionality of the

hardware described above, the following microcontroller software requirements specification was evolved.

4.2.1 Microcontroller software requirements specification

The microcontroller should respond to requests from the UIS to :

- set motor brakes for all axes;
- reset motor brake circuits;
- set the maximum speed for all axes;
- move a joint in a specific direction;
- move a joint to a specific absolute position; and,
- stop movement of all axes.

The microcontroller should be able to communicate to the UIS :

- the occurrence of motor brake set through current limiting;
- the position of each of the axes; and,
- the occurrence of the software limit of an axis being encountered.

A protocol was developed to allow this communication between the microcontroller and the UIS. This is referred to Juvo Motor Control Language (JMCL). JUVO, meaning to assist in Latin, was used as a simpler name for the Middlesex Manipulator during project development. JMCL consists of a set of instructions defined in both mnemonic and op-code form. An instruction exists for each of the requirements listed above, and the following that are specific to communication:

- an 'acknowledge' instruction is sent to acknowledge receipt of an instruction;
- a 'cancel' instruction issued by the UIS to cancel a dialogue (sequence of instructions);
- an 'error-in-transmission' instruction for violation of protocol (i.e. incomplete dialogue);
- a 'next' instruction elicits the next component of a dialogue.

Each instruction is represented by a single byte, and may be accompanied by one or two operands. The JMCL protocol is defined fully in Appendices A and B, however, a listing of the instructions is provided in Table 4.1 , to allow for their use in the pseudo-code contained in subsequent sections.

BRK	-	sets motor brake for all axes
ERM	-	indicates motor brake set
ACK	-	acknowledge
CAN	-	cancel dialogue
ERT	-	error in transmission
HLT	-	stop all axes
Hn	-	stop axis n
Sk	-	set max speed for axis k
Vn	-	set speed of axis n to value passed in next byte
Mnd	-	move axis n in direction d
Pn	-	move axis n to absolute position specified by next 2 bytes
WIn	-	transmit 2 bytes containing position of axis n
RST	-	reset motor brakes
NXT	-	request next byte
Lnd	-	limit of axis n in direction d encountered.

where : $0 \leq n \leq 7$, $0 \leq d \leq 1$, and $0 \leq k \leq 31$.

Table 4.1 JMCL instruction set.

4.2.2 Determining control constants and sampling frequency

With the interface to the UIS defined, the microcontroller code could be developed. The approach taken was to implement proportional control for closed-loop positional control[2]. This is achieved by polling the positional error for each of the 8 axes, and writing a byte to a timer IC proportional to the magnitude of the error. The proportional control constants were determined empirically. This approach was taken as a high degree of friction existed for each axis, and varied significantly throughout the range of movement for the axis, complicating the development of an adequate mathematical model of the system. The constants were determined by increasing their values until the positional error for each axis was minimized. This was performed for each axis at slow speed, and then repeated at increased speed levels until positional accuracy was compromised. This allowed a maximum operating speed for each axis to be determined. The actual speed levels and corresponding accuracy measurements are summarised in section 4.3.

The manipulator's axes were modeled as first-order systems, as the time response of each motor was negligible compared to that of the corresponding axes, subjected to significant frictional components through gearing. Figure 4.8 below, shows the response of one of the Manipulator's axes to a step input (the axis with the fastest response), as approximating a first order step response given by :

$$\frac{V_{out}}{V_{in}} = K(1 - e^{-\frac{t}{\tau}})$$

4.1

Where V_{out} and V_{in} are the output and input signals, K is a constant (in this case normalised to 1), τ is the time constant, and t is the measurement of time.

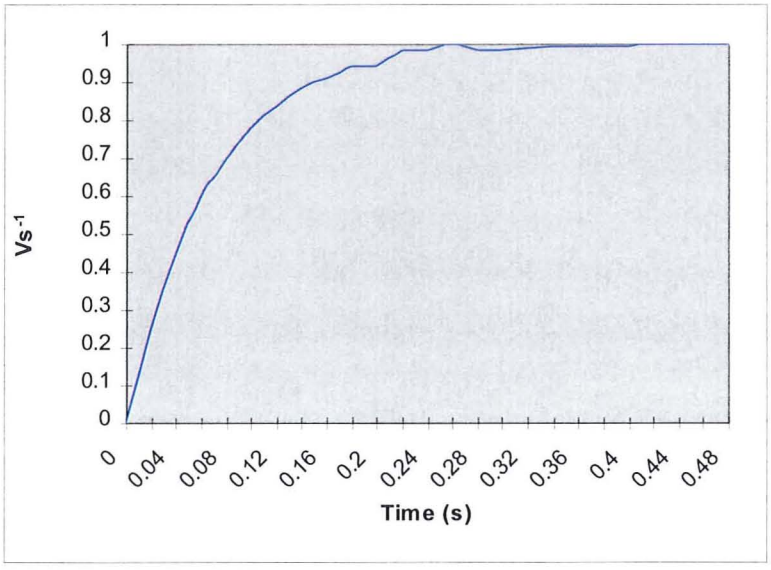


FIGURE 4.8 MANIPULATOR AXIS RESPONSE TO A STEP INPUT

[2] More sophisticated control algorithms such as PID were not investigated due to the system's moderate performance requirements.

When t = τ , the output signal is given as :

$$V_{out} = 1 - e^{-1} = 0.632v$$

4.2

This corresponds to the point t = 0.065s.

The response of the system in the s - domain is given as :

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{(1+s\tau)} = G(s)$$

4.3

And in the frequency domain as :

$$G(jw) = \frac{K}{(1+jwt)}$$

4.4

The bandwidth of the system is estimated as corresponding to the point at which the gain is reduced by 3 dB where :

$$w = \frac{1}{\tau} = \frac{1}{0.065}$$

4.5

As :

$$w = 2\pi f$$

4.6

the bandwidth is estimated at 2.5 Hz. This may be used to estimate an appropriate sampling period. One criterion that has been used successfully is to set the sampling frequency at ten times the bandwidth (Franklin & Powell, 1981). This corresponds to 25 Hz, the reciprocal of which gives a sampling period of 40 ms.

The target sampling period may be achieved through efficient structuring of code, and limiting the complexity of the control algorithms. Initial code tests indicated that a sampling rate of 30 ms would be achievable if the code were to be written in the C programming language.

The presence of noise on the feedback signals necessitated the definition of an acceptable positional error signal. As the axes were designed to be self-locking, this would allow the motor brakes to be set (and hence power cut) once each axis had reached its target position ± the acceptable error. The magnitude of the acceptable error for each axis was limited to ensure that the accuracy requirements of the system were achieved. The values were fixed just below this limit, to allow motor brakes to be set as frequently as possible.

4.2.3 Implementing Cartesian Control

The use of an embedded micro-controller for the motor control system meant that implementing cartesian control (straight-line movement through the x,y or z planes) would be problematic. The kinematic computations required to achieve this in real-time would be beyond the capabilities of the processor, particularly as the processor had to perform other tasks, such as maintain a 40ms sampling period. The approach taken was to limit cartesian movement as being available only when the manipulator is operating in SCARA mode. The following trigonometric analysis was performed to provide a method of controlling the manipulator in the x and y plane.

A SCARA robot achieves straight-line motion in the horizontal plane through the simultaneous adjustment of 2 angular joints. With reference to figure 4.9, the x and y coordinates of the end of link C may be calculated as :

$$y = B \sin \theta + C \sin (\theta + \phi) \qquad 4.5$$

and

$$x = B \cos \theta + C \cos (\theta + \phi) \qquad 4.6$$

Where B and C are the link lengths.

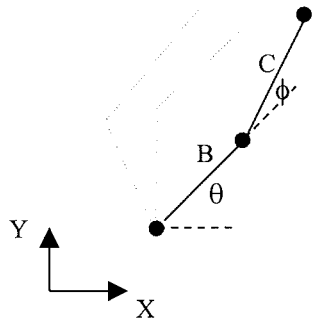


FIGURE 4.9 PLAN VIEW OF LINKS B AND C

For movement through the X plane, x is held constant, and speed may be calculated as the sum of the partial derivatives $\frac{d\theta}{dt}$ and $\frac{d\phi}{dt}$ derived as follows :

$$\frac{dy}{dt} = B\frac{d\theta}{dt}\cos\theta + C\left(\frac{d\theta}{dt} + \frac{d\phi}{dt}\right)\cos(\theta+\phi) \tag{4.7}$$

$$= B\frac{d\theta}{dt}\cos\theta + [x - B\cos\theta]\left(\frac{d\theta}{dt} + \frac{d\phi}{dt}\right) \tag{4.8}$$

But as $\frac{dx}{dt} = 0$:

$$-B\frac{d\theta}{dt}\sin\theta - C\left(\frac{d\theta}{dt} + \frac{d\phi}{dt}\right)\sin(\theta+\phi) = 0 \tag{4.9}$$

therefore :

$$\frac{dy}{dt} = B\frac{d\theta}{dt}\cos\theta + [x - B\cos\theta]\frac{\left(-B\frac{d\theta}{dt}\sin\theta\right)}{C\sin(\theta+\phi)} \tag{4.10}$$

$$= B\frac{d\theta}{dt}\left\{\cos\theta - \frac{[x - B\cos\theta]\sin\theta}{C\sin(\theta+\phi)}\right\} \tag{4.11}$$

and, by Pythagoras :

$$\frac{dy}{dt} = B\frac{d\theta}{dt}\left\{\cos\theta - \frac{[x - B\cos\theta]\sin\theta}{\sqrt{C^2 - (x - B\cos\theta)^2}}\right\} \tag{4.12}$$

Additionally, $\frac{d\phi}{dt}$ may be computed $\frac{d\theta}{dt}$, as :

$$\frac{d\phi}{dt} = \frac{-B\frac{d\theta}{dt}\sin\theta - C\frac{d\theta}{dt}\sin(\theta+\phi)}{C\sin(\theta+\phi)} \tag{4.13}$$

hence :

$$\frac{d\phi}{dt} = -\frac{d\theta}{dt}\frac{B\sin\theta + \sqrt{C^2 - (x - B\cos\theta)^2}}{\sqrt{C^2 - (x - B\cos\theta)^2}} \tag{4.14}$$

Hence, for constant velocity through the X plane, i.e. for a given value of $\frac{dy}{dt}$, values for $\frac{d\theta}{dt}$ and $\frac{d\phi}{dt}$ may be computed. As discussed above, the current implementation uses open-loop speed control, thus errors would occur and not be corrected. However, more significantly than this, performance measurements reported in section 4.3, show that the maximum value for $\frac{d\phi}{dt}$ is 0.1 rad s^{-1}. Consequently, the required linear velocity of 100 mm s^{-1} would be unattainable.

An alternative approach to approximating straight line motion would be to hold $\frac{d\phi}{dt}$ at its maximum, and for each new θ, compute a value of ϕ to satisfy (from 4.6) :

$$\phi = \left(\sin^{-1} \frac{x - B\sin\theta}{C} \right) - \theta \qquad 4.15$$

This approach results in far lower computational complexity, than solving $\frac{d\theta}{dt}$ and $\frac{d\phi}{dt}$ for a given $\frac{dy}{dt}$. The major disadvantage would be the resulting variation in $\frac{dy}{dt}$. This was quantified by using equation 4.15 to calculate typical values of $\frac{dy}{dt}$, and typical levels of variation. The results are shown in figures 4.10 and 4.11.

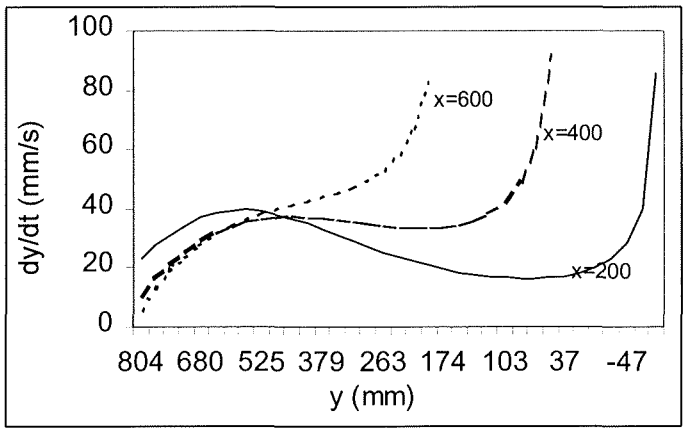


FIGURE 4.10 LINEAR VELOCITY V DISPLACEMENT FOR LINK C = 500 MM

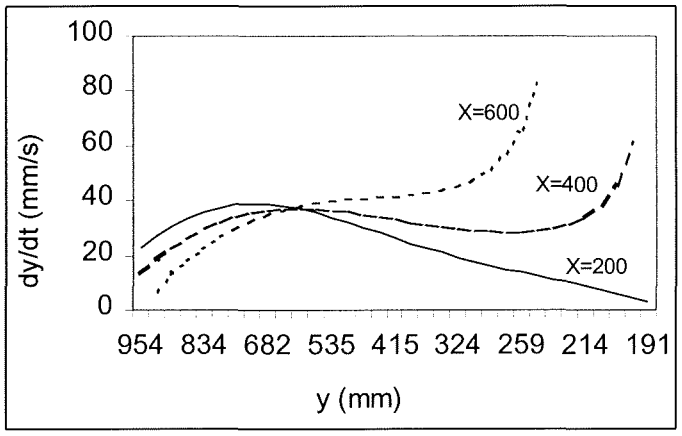


FIGURE 4.11 LINEAR VELOCITY V DISPLACEMENT FOR LINK C = 650MM

The results show that the end-point velocity increases rapidly as a limit is approached, beyond which the angles for maintaining a constant X cannot be computed. For the remaining values of displacement, velocity is limited to around 40 mm s^{-1}, with levels of variation highly dependent on x and C (link extension). A user evaluation would be required to determine how usable a system was that employed this method of control, i.e. to establish to what extent a user may accommodate variations in speed.

However, for particular configurations, variations in $\frac{dy}{dt}$ are small. For example, for link C = 500 mm and x = 400 mm, $\frac{dy}{dt} = 34 \pm 4$ mm s^{-1} for 628 mm travel. In other words, the maximum variation of speed over 79% of the range of linear movement was 12 % .

4.2.4 Top-level motor controller pseudo-code

At the highest level, the microcontroller is concerned with :

i) Servicing any requests from the UIS. This may involve :

- setting or resetting motor brakes;
- modifying the target positions of one or all of the axes;
- setting the maximum permissible speed; and,
- providing positional information for the UIS.

ii) Modifying motor drive and brake signals by :

- deciding if brakes have been set (forced on) by the UIS;
- making a local decision for brakes to be set if target positions were previously reached;
- determining whether any positional error has been exceeded and motors should be moved.

Software was implemented in C to achieve this, the pseudo-code of which is provided in figure 4.12 below. For commented code listings refer to appendix C. However, to supplement the code documentation, a number of functions that are called by the main program are expanded in pseudo-code form in the following sections, these include :

- a 'move' function - calculates magnitude of output signal for an axis;
- a 'read current position' function - reads position of all axes;
- a 'read' function - reads a byte from the microcontroller's serial port; and,
- a 'transmit' function - transmits to UIS through serial port.

```
Micro controller main program, begin
    ensure motor brakes are set
    set the initial positions for each axes as target positions
    repeat forever :
    begin
            if  input has been received from UIS, read it and
            begin :
                    if input is BRK then
                            set brake on for all axes
                            set speed to zero for all axes
                            set target position to current position for all axes
                    if input is HLT then
                            bring each axes to a stop by setting the target position to
                            the current position ± a small pre-specified amount.
                    if input is Hn then
                            bring axis n to a stop by setting the target position to
                            the current position ± a small pre-specified amount.
                    if input is RST then
                            reset motor brake for each axes
                    if input is WIn
                            transmit the current position of axis n
                    if input is Mnd then
                            set target position of axis n to maximum value in direction d
                    if input is Sk then
                            set maximum speed for all axes to k
                    if input is Vn then
                            set maximum speed for axis n to following byte
                    if input is Pn then
                            then read two bytes containing target position and
                            set target position of axis n.
            end ( of  new input block )

            check whether any axes require moving by
            begin
              read current position of all axes and calculate positional error
              if the brakes are not currently forced on by the UIS, then
              begin
                    if  status for all axes = target reached then
                            set brakes on
                    otherwise, if the brakes were previously set, then
                            reset brakes
                    For each axis
                            if positional error > permissible error then
                                move the axis towards target and
                                set the axis status to target not reached.
                            otherwise
                                set the axis status to target reached.
              end
            end ( of check for move block )
    end (of  repeat forever loop)
end (of main program)
```

FIGURE 4.12 CONTROLLER MAIN PROGRAM PSEUDO-CODE

4.2.5 The Move function

Initial tests showed that better control of the arm was found to be possible if two proportional constants were used for each axis. This allowed the algorithm to cater for the fact that for most axes the damping due to friction, and the offset due to gravity, was different in each direction. Two arrays were therefore used to contain these constants.

If a step input was applied to the motor of any axis in response to a large positional error, the motor torque generated would cause the current limit setting to be exceeded. To cater for this, a simple low-pass filter was implemented in software, limiting the rate at which the drive signal may change.

An averaging value is calculated for output using the first order equation :

$$y = \beta y_t + (1 - \beta) y_{t-1} \qquad 4.1$$

where y is the value output, y_t is the output as calculated proportional to the error signal, y_{t-1} is the previous signal output. The characteristic constant β determines the effect to which a new calculated output value effects the actual output. As rapid acceleration of the motors was not required, this constant was set fairly low. The actual current limits may be adjusted through a potentiometer mounted on each motor drive board. Tests were undertaken iteratively to determine a β value low enough for the most sensitive current limit setting. See appendix C for implementation

4.2.6 Reading axes positions

The 'read current position' function implements a software filter similar to that described above. This reduces noise on the feedback signal, complementing the hardware filters implemented on the positional feedback board. Pseudo-code for reading from a single axis is shown in figure 4.13.

```
Function to move Axis n
begin
        multiply positional error by gain Kq or Kp dependent on direction to
        the determine drive-signal magnitude

        if drive-signal magnitude > maximum level for current speed setting
        then
            drive-signal = maximum level for current speed setting
        if accelerating
        then filter output
            drive-signal = drive-signal × alpha constant
            drive-signal = drive-signal + previous drive-signal  × (1 - alpha constant)
        output drive-signal
end
```

FIGURE 4.13 MOVE FUNCTION PSEUDO-CODE

```
Function to read axis position
begin
    set a count equal to the required data sample length
    select the appropriate input channel with the analogue switch
    initiate A/D conversion
    set a variable LastSample to value read from A/D converter
    repeat while count > 0
            initiate A/D conversion
            set a variable Sample to  value read from A/D converter
            set LastSample = LastSample × (1 - alpha constant)
            set LastSample = LastSample + Sample × alpha constant
            decrement count
    end of repeat
end of function
```

FIGURE 4.14 PSEUDO CODE FOR READING AXIS POSITION

4.2.7 Serial IO

The 8032 has a Universal Asynchronous Transmitter Receiver (UART) to handle serial communication. This was configured as an 8-bit UART, with a baud-rate determined by one of the 8032's onboard counter timers. Transmission of a byte is achieved by writing the byte to a special purpose register (SBUF), and a byte is received by reading from SBUF. Flags set by the serial control register (SCON), allows for transmit and receive status to be determined.

Reading a byte (character) from the serial port is achieved by :

i) checking the SCON flag to determine if a character is ready;

ii) clearing the SCON flag; and,

iii) reading a byte from SBUF.

If a dialogue is in progress, the next byte of the dialogue is requested by the GetNextByte function described in figure 4.15 below.

The transmission of a byte is achieved by :

i) waiting until SCON flag indicates UART is ready to transmit;

ii) clearing SCON flag; and,

iii) writing a character to SBUF.

A dialogue requiring two bytes (a word) to be transmitted may call the SerialWordOut function shown in figure 4.16.

```
function GetNextByte
begin
    set a timer variable to zero
    transmit the JMCL NXT command

    while a character is not ready and timer < acceptable wait period
    begin
        increment the timer variable
    end

    if timer < acceptable wait period
        read and return character from SBUF
    otherwise
        return ERROR (calling function will transmit ERT);
end
```

FIGURE 4.15 FUNCTION TO GET THE NEXT BYTE OF A DIALOGUE

```
function SerialWordOut
begin
    split word into 2 character variables msb and lsb
    initialise a timer variable to zero
    transmit msb
    while a response character is not ready and timer < acceptable wait
    begin
            increment timer
    end
    if timer < acceptable wait
            read character (data)
                    if data = NXT
                            transmit lsb and return not ERROR
    otherwise
             transmit ERT and return ERROR;
end
```

FIGURE 4.16 FUNCTION TO WRITE A DATA WORD TO SERIAL PORT

4.3 Performance characteristics

This section summarises measurements of the manipulator's performance characteristics, achieved with the control system described above. The measurements were taken as a part of the design process, in parallel with the design decisions described earlier in the chapter.

As discussed below, a compromise was involved when attempting to meet the speed and accuracy requirements of the design specification for each of the manipulator's axes. For ease of reference, the manipulator is described here as consisting of 4 links, and 6 axes as labeled in figure 4.17.

This section begins with a discussion of the two linear axes (1 and 5). These allow for movement through the vertical plane, and for extension of link C. The rotational axes 2 and 4 are then considered, as their simultaneous control allows for movement through the horizontal plane in SCARA mode. An analysis is presented to allow prediction of performance in SCARA mode from the measured performance of these two axes in 'joint' mode.

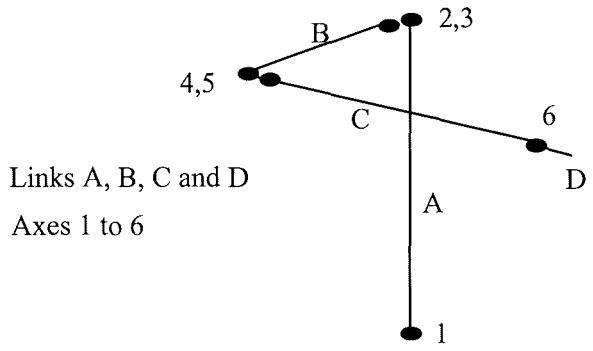


FIGURE 4.17 MANIPULATOR CONFIGURATION

Finally, an assessment of axis 3, used to provide vertical articulation, is presented. As a 3 degree-of-freedom end-effector is currently under development, the characteristics of Axis 6, which is used to control a temporary gripper, are not discussed.

4.3.1 Axes 1 and 5 – velocity

Ideally, the operating speeds of each of the manipulator's axes would be set to allow a velocity at the manipulator's end-effector corresponding to that detailed in the design specification, i.e. a maximum operating speed of 100 mm s^{-1}, with fine-control of 50 mm s^{-1}. The magnitudes of the drive signals to each axis could thus be determined empirically, as part of the design process. However, initial tests indicated that aspects of the manipulator's construction meant that the required speed levels would not be achievable. For the linear axes, speeds were limited principally by the unacceptable levels of acoustic noise generated by friction between the plastic strips used as linear bearings, and the manipulator's casing (the hollow casing acting as an acoustic amplifier).

The design specification required a noise level of no greater than 40 dB at 1 m. A noise level meter was used to record the noise generated (type 2203, Bruel & Kjaer, Naerum, Denmark). The meter was configured with a frequency response matching an 'A' weighting scale, providing weighting corresponding to the relative annoyance typically produced by different frequency components. Noise levels of around 65 dB(A) were measured at angular speeds of around 1500 rev/min for axis 1, and 1800 revolutions/min for axis 4. One approach would have been to reduce axis speeds until levels below 40 dB(A) were generated. However, the user evaluation reported below, highlighted the fact that the type of noise being generated was also a significant factor. In particular, variation in pitch and amplitude with the manipulator in motion was reported to have a significantly negative effect on the user's impression of the system. Consequently, a more

subjective approach was taken to establishing the maximum speed of each axis: speed levels were reduced until noise levels were deemed acceptable by the user (and designer). This limited the angular velocities of axes 1 and 5 to 750 r.p.m. and 900 r.p.m respectively.

Axis 1 may produce a movement through the vertical plane at a velocity given as :

$$v = \frac{\omega\, d}{60} \qquad 4.2$$

Where v is linear velocity (mm s^{-1}), ω = angular velocity (r.p.m.), and d is the lead-screw pitch (mm). As d is 2 mm, velocities of up to 25 mm s^{-1} were attainable. Similarly, axis 5 may produce an extension to link C at a speed of up to 30 mm s^{-1}.

Although falling short of the design specification, the values computed above correspond to the fastest speed settings for the two linear axes. The decision was made to define two further speed levels (medium and slow), providing a degree of consistency with the remaining axes. Slow was set at approximately half of fast speed, with medium falling near the mid-point. The resulting speed levels are summarized in Table 4.2 below.

Axis	**Speed**	ω / (rev min^{-1})	v / (mm s^{-1})
1	slow	360	12
5	slow	480	16
1	med	540	18
5	med	720	24
1	fast	750	25
5	fast	900	30

Table 4.2 Speed levels (axes 1 & 5)

4.3.2 Axes 1 and 5 - repeatability

A number of measurements were undertaken to determine how repeatability varies with speed and load. Four positions along the range of each axis were selected as target positions. A dial-gauge was used to measure the variation in positioning around the target. Each set of measurements produced a cluster of positions, from which a center point was calculated. Repeatability was estimated by examining the maximum variation from the center point, and the average variation. The results are summarised in tables 4.3 to 4.7 below.

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	0.8	0.7
Med	0.9	0.7
Fast	1.4	0.8

Table 4.3 Axis 1 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	1.0	0.7
Med	1.2	0.8
Fast	1.7	1.0

Table 4.4 Axis 1 (load = 1 kg)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	0.9	0.7
Med	1.0	0.7
Fast	1.3	0.9

Table 4.5 Axis 5 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	0.9	0.4
Med	0.9	0.7
Fast	1.3	0.8

Table 4.6 Axis 5 (load = 1 kg)

4.3.3 Axes 2 and 4 - velocity

As with the linear axes described above, practical considerations resulted in limiting the maximum operating speeds for axes 2 and 4 to levels below the design targets. The principal limiting factor was the variation of friction throughout the axes' range, particularly for axis 2, and the fact that this variation had a greater impact on positional accuracy at higher speeds.

Deterioration of performance with increasing speed was quantified by measuring repeatability at a number of positions throughout the axis range. For each axis, 5 positions were selected, from which a sample of 8 measurements was taken. Measurements were taken at the end of link C, with the link fully extended, providing a worst-case configuration. As with the linear axes, repeatability was estimated by quantifying the maximum and average distances from cluster mid-points. The process was repeated as the speed of the axis was increased. Figure 4.18 below summarizes the performance of axis 2.

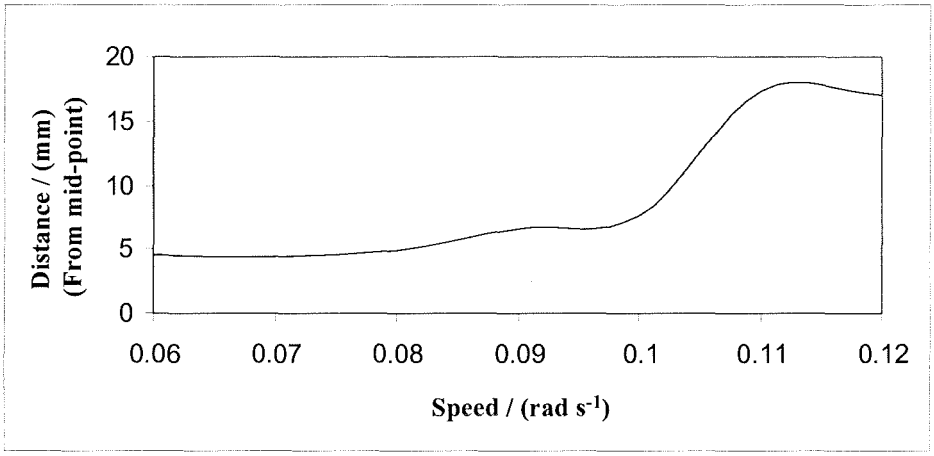


FIGURE 4.18 REPEATABILITY ESTIMATES (AXIS 2)

As can be seen, if repeatability were prioritized above speed, the operating speed may be set below 0.08 rad s^{-1}, before the deterioration in repeatability is evident. However, as shown below, this would result in speeds well below those required. Thus, as a compromise, a value of 0.1 rad s^{-1} was selected, avoiding the sharp deterioration in repeatability displayed at 0.11 rad s^{-1} and above.

Figure 4.19 shows the deterioration of repeatability for axis 4 as being more gradual than axis 2. However, setting the maximum speed of the two axes at significantly different levels, may decrease the usability of the system, as the concept of 'fast' would take on very different meanings for each axis. The maximum operating speed of axis 4 was therefore set slightly greater than axis 2 at 0.14 rad s^{-1}.

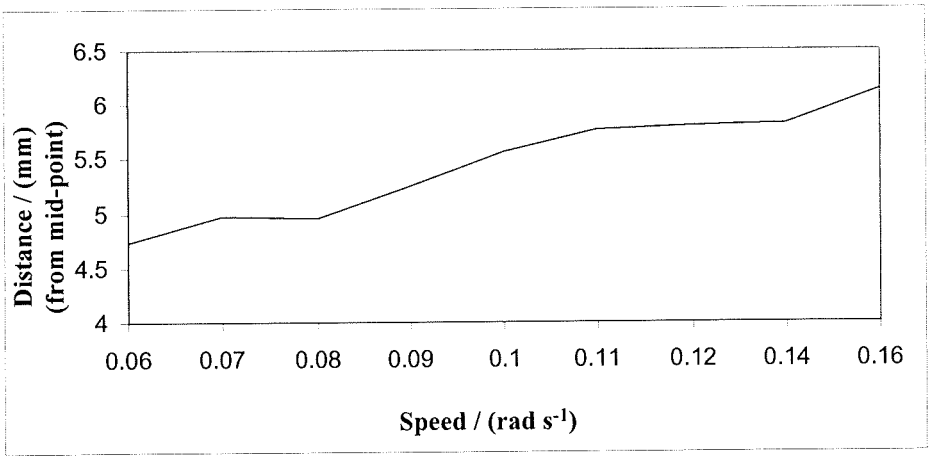


FIGURE 4.19 REPEATABILITY ESTIMATES (AXIS 4)

With the maximum speed levels established, two further speed levels were defined (as with the linear axes), and are summarised below. The angular velocities of both axes are shown, with calculated end-point velocities. These correspond to the speed of movement in an arc at the end of link B for axis 2, and link C for axis 4. i.e.

$$v = \omega B \tag{4.3}$$

for axis 2, where B is link length (375 mm), and ω is angular speed in radians s^{-1}. For axis 4,

$$v = \omega C \tag{4.4}$$

where C is 690 mm.

Axis	Speed	ω / (rad s^{-1})	v / (mm s^{-1})
2	slow	0.06	23
4	slow	0.08	55
2	med	0.08	30
4	med	0.11	76
2	fast	0.10	38
4	fast	0.14	97

Table 4.7 Speed levels axes 2 and 4

4.3.4 Axes 2 and 4 - repeatability

A process as described for the linear axes was undertaken to estimate the levels of repeatability for axes 2 and 4, the results of which are summarized below.

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	4.9	4.6
Med	4.0	4.8
Fast	8.2	5.4

Table 4.8 Axis 2 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	5.0	4.9
Med	5.1	4.9
Fast	6.7	6.1

Table 4.9 Axis 2 (load = 1 kg)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	4.5	4.1
Med	4.7	4.2
Fast	5.1	4.8

Table 4.10 Axis 4 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	4.8	4.1
Med	4.7	4.6
Fast	5.4	5.1

Table 4.11 Axis 4 (load = 1 kg)

4.3.6 Axis 3 – velocity

The remaining axis allows for movement through the vertical plane, and provided characteristics similar to axis 4, in that the friction remained fairly constant through the axis range, and thus degradation of performance was more gradual than with axis 2.

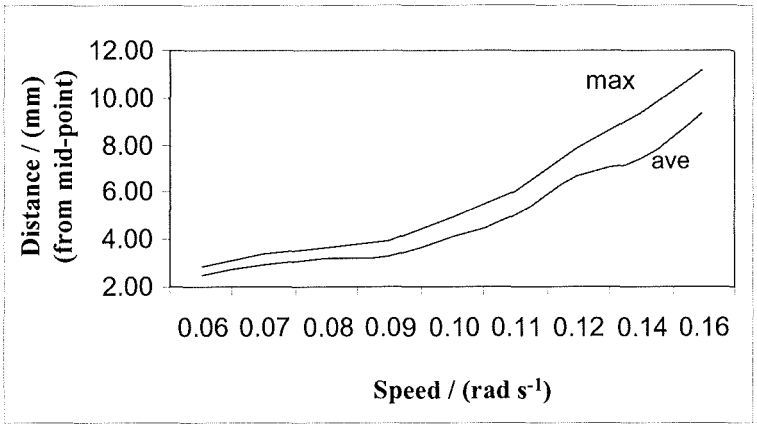


FIGURE 4.20 REPEATABILITY ESTIMATES (AXIS 3)

However, as with axis 4, an operating speed was selected to be of a comparable level to the remaining axes. A speed of 0.12 radians s^{-1} was chosen. The two additional speed settings and corresponding end-point speeds are summarized below (link C fully extended).

Axis	**Speed**	**ω / (rad s^{-1})**	**v / (mm s^{-1})**
3	slow	0.07	48
3	med	0.10	69
3	fast	0.12	83

Table 4.12 Axis 3 speed levels

4.3.7 Axis 3 – repeatability

As with the previous axes, a number of measurements were taken to estimate levels of repeatability. Again, measurements were taken at the end of link C, with the link fully extended to provide a worst-case configuration.

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	7.5	6.1
Med	7.7	6.2
Fast	8.1	6.4

Table 4.13 Axis 3 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	7.5	6.2
Med	7.5	6.2
Fast	7.9	6.3

Table 4.14 Axis 3 (load = 1 kg)

4.4 Concluding remarks

To summarize, an embedded microcontroller-based motor control system has been implemented at a one-off component cost of £440. Up to eight DC servo motors may be driven using PWM closed-loop position and open-loop speed control. A modular approach to system design has been taken, to allow for ease of maintenance through the replacement or servicing of system modules. A communication protocol has been defined (JMCL), allowing full functionality of the system to be controlled via a serial interface.

Initial tests provided estimates of the performance currently achievable by the manipulator. Table 4.15 summarises the largest estimates of repeatability for each of the axes, rounded up to the nearest mm.

Axis	**Repeatability / (mm)**
1	2
2	9
3	9
4	6
5	2

Table 4.15 Repeatability estimates

The target repeatability given by the requirements specification is 10 mm. As can be seen, control of any individual joint can achieve this, however, the cumulative error of movement involving more than one joint may exceed this.

The principal factor determining the magnitude of repeatability was mechanical, namely the back-lash that exists in the gear mechanisms. As would be expected, repeatability is improved if a target position is always approached from the same direction. Typical values for 'single-approach' repeatability are provided for axes 2 and 3 in tables 4.16 and 4.17.

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	2.2	1.9
Med	2.7	2.3
Fast	4.8	3.7

Table 4.16 Axis 2 (no load)

Speed	Repeatability measure / (mm)	
	maximum	average
Slow	2.3	1.6
Med	2.5	1.7
Fast	3.3	2.9

Table 4.17 Axis 3 (no load)

Thus estimates of 'single-approach' repeatability for axes 2 and 3 are 5mm and 4mm respectively, as compared to the 9mm estimate for 'actual' repeatability.

Future developments of the prototype should address the degree of back-lash within the gear mechanisms, however, it was considered reasonable to expect that the current levels of repeatability would suffice for initial evaluations. This approach may be justified considering that the estimates are 'worst-case' in that they presume the arm to be fully extended, thus for much of the working envelope, repeatability will be lower than the estimates. Additionally, 'single-approach' repeatability can be exploited by pre-programmed routines, as well as by competent users.

Selecting appropriate speed levels involved a trade-off between speed and repeatability for axes 2, 3, and 4, and speed and noise for axes 1 and 5. Thus improving the manipulator's speed performance would also require mechanical modifications. The current maximum speed attainable is less than that required, this is particularly evident for cartesian control with around 40 mms^{-1} possible through the horizontal plane, and 25 mms^{-1} through the vertical plane.

In summary, a number of short-comings have been identified whilst assessing the manipulator's current performance capabilities. These are mainly mechanical in nature, and will therefore not be addressed as part of the current phase of the project. Chapter 8 outlines a user evaluation of the Middlesex Manipulator, allowing the impact of the manipulator's performance to be addressed in subjective terms.

Section III

User Interface Design

Chapter 5

HCI and Interactive System Design

Chapter one provided an outline to the thesis and identified project objectives, including the development of a user interface for the Middlesex Manipulator. Chapter 2 provided general design criteria, which included the development of a system that is:

- easy to use;
- easy to learn how to use;
- efficient to use through the combination of different types of input and feedback devices, and
- adaptable to the preferences, functional ability and experience of specific users.

The following chapter explores how techniques developed within the field of human-computer interaction (HCI) may be applied to these objectives. The overlap between the fields of Rehabilitation Robotics and HCI has previously been extremely limited. Consequently, this chapter provides an overview of the field of HCI and a description of the most common HCI evaluation techniques. The chapter concludes that most HCI techniques do not formally address diversity, adaptability, multi-modality and device novelty to the extent required by Rehabilitation Robotics (or Assistive Technology in general). However, within certain constraints the techniques are applicable, as demonstrated during the user interface design presented in Chapter 6, and the evaluation presented in Chapter 8. These ideas are then explored further in Chapter 9, with the development of a novel evaluation methodology based on a common HCI task analysis technique.

5.1 Introduction

The field of HCI has experienced rapid growth over the passed two decades. The ever-increasing use of computing technology within home and working environments has considerably increased the number and diversity of users, and the expectation that the usability of systems should be high. In 1984, Smith and Mosier (1984) estimated that for a typical software project the user interface accounted for 30-35% of the code written. A study undertaken just six years later, estimated that this had grown to as high as 60% (MacIntyre, et. al. 1990) - with the field of HCI growing accordingly.

Nielson (1994) describes the objectives of HCI practitioners by examining the issues that determine system acceptability:

System Acceptability

- cost;
- compatibility;
- reliability;
- usefulness;

Usefulness is defined as being the combination of usability and utility, where utility is the value of that which can be achieved with the system. Finally, usable systems should be:

- efficient to use;
- easy to learn;
- easy to remember;
- subjectively pleasing, and
- preventative of errors.

As can be seen, a similarity exists between the objectives of rehabilitation robotics research, and those of the field of HCI. However, until recently, the two fields have been fairly isolated from each other, with few reports of the systematic application of HCI techniques to rehabilitation robotic systems. A similar situation was recognised to exist within mainstream robotic research,

as reported by Anzai (1994), who called for the recognition of a new research paradigm: human-robot-computer interaction. Within the field of assistive technology, there is now a growing interest in the development of formal techniques to address usability issues (for example, Edwards (1995); Poulson et. al. (1996)). A recent rehabilitation robotics conference in Bath U.K., reflected this growing interest, containing three consecutive papers discussing the application of HCI techniques to system design (Dowland et. al., (1997); Keates and Robinson (1997); Parsons et. al. (1997)). This chapter provides a review of HCI evaluation methodologies that are common to interactive systems design, and examines how these may be applied to the current project objectives.

5.2 The product design life-cycle

A number of techniques have been developed to promote system usability that may be employed at various stages within an iterative design cycle. A typical design cycle begins with the generation of a requirements specification. This usually involves the refinement of a brief problem statement into a detailed specification of the functionality and performance that the system is required to provide. Systems analysis techniques, and more recently Object Oriented Analysis techniques (OOA), have been developed to model data and tasks within a problem domain.

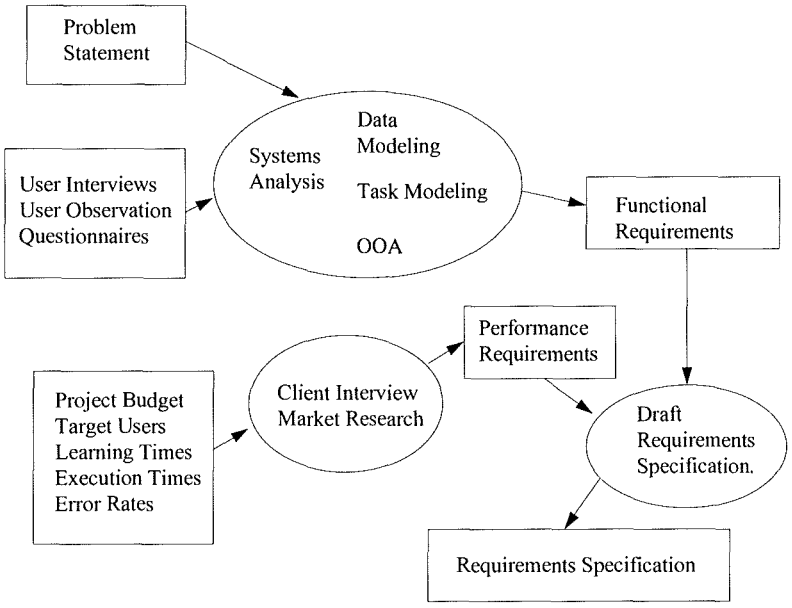


FIGURE 5.1 GENERATING A REQUIREMENTS SPECIFICATION

The development of a design solution may take the initial form of a written design description, a paper mock-up or a prototype. HCI evaluation methodologies may be used to evaluate the system against the original requirements specification, allowing the design to progress towards a solution acceptable to the client. Various methodologies have been designed to be used at different stages of the product design life-cycle, focusing on different qualitative and quantitative characteristics of the interface. Techniques also vary in the expertise, money and time required for their implementation.

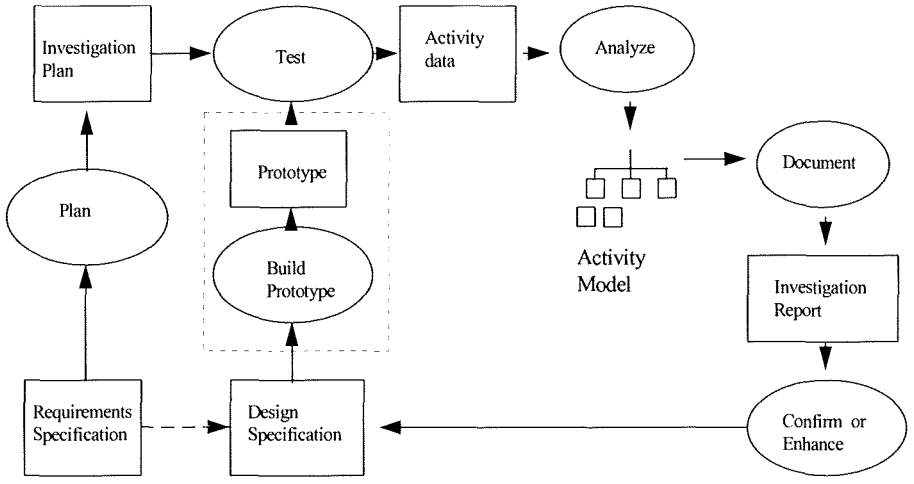


FIGURE 5.2 AN ITERATIVE SOFTWARE DESIGN CYCLE

To present an overview of HCI evaluation methodologies, this chapter uses the broad classifications: Analytic, Usability Inspection and Experimental. Examples of each are provided.

5.3 Analytic techniques

Analytic techniques provide formal ways of describing a problem domain, forms of user interaction, and models of computing systems. As part of the design process, the techniques may be employed to generate or verify requirement specifications or design specifications. Models of interaction may be analysed to allow an assessment of the functionality, consistency and

complexity of a user interface. In some cases, predictions of task completion times, task learning times or error rates may be made. This section outlines a number of common techniques and discusses their suitability for use in assistive technology design.

5.3.1 Task Analysis

Task analysis is a method for providing an abstraction of tasks that users are required to undertake. A common form of analysis, Hierarchical Task Analysis (HTA), is a systems analysis tool that has been adapted for use in HCI (Annet and Duncan, 1967). The technique forces a designer to focus on the details of an application, ordering the cognitive and physical processes required to accomplish a goal.

Information concerning the task is first gathered using a data collection method such as activity sampling, observation, documentation, structured interviews or questionnaires. The data is then organised into a hierarchy of goals, sub-goals and operations. A goal is defined as something the user wishes to achieve. Goals are decomposed into sub-goals, dependent on the level of detail (granularity) appropriate to the analysis. Operations are defined as the activities that must be undertaken to achieve the goals.

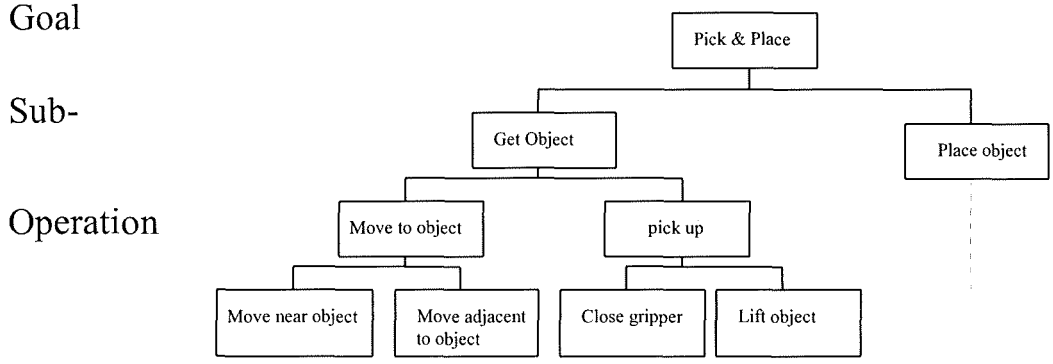


FIGURE 5.3 HTA REPRESENTATION OF A PICK AND PLACE TASK

The original context for the application of HTA was training. As a result, the technique was concerned with the empirical analysis of existing tasks. No information necessarily results from an analysis regarding the appropriateness of the structure of the tasks modeled, or possible alternatives. Furthermore, HTA does not encapsulate procedures for generating or evaluating a

design. Instead, the task model resulting from HTA would typically be used as an input to further forms of analyses or design, as discussed below.

5.3.2 Grammar based analysis

Formal languages have been developed to allow the description of the structure of a user interface, and models of interaction - an early example of which is Command Language Grammar (CLG (Moran, 1981)). CLG was developed as a designer's model of interaction, allowing for a description of the inputs to and outputs from a computing system. CLG separates the conceptual components of a system (user's mental models) from the command language used for interaction, and describes the relationships between these components.

CLG adopts a top-down approach to design, structured at the following levels of description:

Conceptual	Task level
	Semantic level
Communication	Syntactic
	Interaction
Physical	Spatial
	Device

The task level is concerned with what the system is supposed to do, and may be described by a task model such as that produced by HTA. The semantic level then defines the system's entities (conceptual objects) and operations (actions that may be performed on objects). The sequences of operations required to complete tasks are then described as methods using a form of pseudo-code. The pseudo-code used may be regarded as a grammatical representation of a semantic net[1].

The syntactic level is then concerned with describing the structure of the language used for interaction. Within a given context (such as attempting to complete a particular conceptual

[1] A semantic net is formal graphical language for representing facts about entities within a domain of interest, see Dym and Levitt, 1991.

operation), there exists a set of commands, and state variables modified by those commands. Figure 5.4 provides the context of arm movement within a manipulator controller application.

```
ARM_MOVE_CONTEXT =   (A COMMAND CONTEXT
                     STATE_VARIABLES = ( SET :        CURRENT_POSITION
                                                      TARGET_POSITION)
                     DESCRIPTORS = (SET :             XYZ_COORDINATES)
                     COMMANDS = (SET :                ARM
                                                      SHOULDER
                                                      ELBOW
                                                      HAND
                                                      IN
                                                      OUT
                                                      STOP
                                                      EXIT)
                     ENTRY_COMMANDS = (SET : MOVE))
```

FIGURE 5.4 CLG SYNTACTIC LEVEL DESCRIPTION

The interaction level of analysis then describes the actions of a user. These become specific to a particular system as defined at the physical level of analysis. CLG allows for the description or definition of the structure of a user interface for a particular system without requiring that the entire system be defined. However, the method does not incorporate any metrics that would allow for the usability of a design to be predicted or evaluated. This shortfall was addressed by Reisner (1981), with the development of a production-rule based grammar referred to as Task Action Language (TAL). TAL attempts to describe the cognitive factors of what a user has to learn and remember to complete tasks. The rules governing interaction are described in terms of:

1. terminal symbols (the words in the language);
2. non-terminal symbols (constructs that show the structure of the language);
3. a starting symbol;
4. meta-symbols (+ *(and)* , | *(or)*, *:= (is composed of)*
5. rules constructed from the above

The premise behind TAL is that well designed systems will require fewer and shorter rules and terminal symbols to describe the system than complex or inconsistent systems. Hence evaluation is possible by comparing the descriptions of alternative interface designs. However, a number problems exist with this approach. A well-documented issue, as described by Johnson (1992), is that TAL lacks a model of the user, and hence there can be no certainty that the rules and terminal symbols match the cognitive aspects of behavior. However, this statement has

additional connotations for assistive technology design. When designing a user interface for the diverse user group addressed by assistive technology, it would be problematic to derive a cognitive model representative of the entire user group. The statement also suggests that the focus of the analysis is the cognitive complexity of tasks. However, for much of the user group of assistive technology, the physical actions required to complete a task would be more significant than the cognitive factors, and deriving a user model to represent the physical abilities of the user group is not addressed.

1 joint movement ::=	select start | change joint selection + select start | change direction selection + select start | change joint selection + change direction selection + select start
2 select start :: =	issue start gesture | issue voice command
3 issue start gesture ::=	move head forward + pause + move head back

FIGURE 5.5 EXAMPLE TAL RULE DESCRIPTIONS

The lack of a user model is partially addressed by Task Action Grammar (TAG), a development of TAL (Payne & Green, 1986). TAG uses a formal grammar to assess usability based on the syntactic and semantic complexity and consistency of rules used to describe interaction. TAG introduces the notion of a dictionary of simple tasks. These are the fundamental components or operations within a task description that define the granularity of the analysis (corresponding to move head forward or pause in the TAL example of figure 5.5). Simple tasks are defined as those that may be performed without problem solving or iteration. Thus the user model underlying the analysis can be assumed to be correct if the selection of simple tasks is correct. As discussed by Johnson (1992), identifying simple tasks relies upon the intuition of the analyst. This weakness increases in significance for the design of assistive technology, where the simplicity or otherwise of a task will be highly dependent upon the functional ability of the user.

5.3.3 GOMS task analysis

Incorporating a user model within task analysis was more formally addressed by Card, Moran and Newell (1983), with the development of GOMS analysis (Goals, Operators, Methods and Selection Rules).

The components of a GOMS analysis may be described as:

- A Goal - something the user wishes to accomplish.
- Operators - an action that the user executes
- Methods - sequences of steps that accomplish a goal
- Selection - selection rules that exist when more than one method may be used to achieve the same goal.

The model resulting from a GOMS analysis combines user characteristics and interface characteristics in the context of the task. The user characteristics are derived from the model human processor (MHP). This views a user as consisting of three subsystems: the perceptual, cognitive and motoric subsystems, with each subsystem having its own memory and processor. Information is processed by each subsystem, and transferred to an adjacent subsystem within a finite time referred to as cycle time. The memory of the perceptual subsystem is divided into auditory and visual, each with a specific capacity and decay time for stored information. Similarly, the cognitive subsystem is divided into working memory, which has a finite capacity and decay time, and long-term memory, which has an infinite capacity and decay time[2]. Thus information is encoded symbolically by the perceptual subsystem and then passed in to working memory. Previously stored information is retrieved from long-term memory, allowing a decision to be made about a response, which may then be executed by the motor subsystem. The parameters of the MHP were derived from psychological theory, and empirically (Card et. al., 1983), and include measurements of basic operations relevant to user interaction, such as mouse moves, mouse clicks and locating items on a screen.

Similar to the grammar-based techniques described above, a GOMS analysis requires as an input a model of the task to be analysed. This may then be described in a form similar to a procedural programming language as described by Kieras (1988). The task or goal is divided into sub-goals, which are described by the methods (procedures) required to complete each sub-goal. The methods consist of a sequence of simple actions or operators which, dependent upon the granularity of analysis, would relate to the perceptual, cognitive and motor activities of the MHP.

[2] Infinite capacity for LTM is justified on the basis that the analysis is of learnt tasks, thus the procedural knowledge required already exists.

A GOMS analysis allows for the functionality of a system to be verified, and the consistency and complexity of the interface to be assessed. An advantage of GOMS over the techniques described above, is that approximate predictions of task completion times may be made. A comparison of common HCI evaluation methodologies (Nielsen and Phillips, 1993), demonstrates that GOMS provides the best alternative to experimental evaluation, when estimating the relative usability of possible interface configurations. This is particularly the case where task completion times, or ease of menu navigation, are critical to usability.

GOMS techniques may be implemented with a design description, or incomplete prototype, and can therefore be used early on in the design process. However, the complexity of the techniques has restricted their popularity: they are far more dominant in research environments than in commercial environments. They are also unable to address a number of usability issues, particularly subjective issues, which are better addressed by techniques outlined below.

As GOMS task analysis uses the MHP as a user model, its application to assistive technology interface design is problematic. The notion of a 'typical user' is not applicable to the target user population, and the development of novel and adaptable interfaces requires consideration of the characteristics that make individuals different. However, the GOMS approach makes explicit reference to the user characteristics relevant to user interaction. Chapter 9 provides a framework whereby the characteristics of a specific user may be estimated and included within a GOMS-like analysis. An experiment is reported that uses this approach to predict the relative usability of various interface configurations, based on variable user and device characteristics.

5.4 Usability Inspection Techniques

Usability inspection techniques have been developed to allow for more rapid and cheaper evaluations than analytic or experimental approaches allow. These may be performed by usability experts, or software designers, and allow for the assessment of an interface against accepted usability guidelines. Two common forms of evaluation are Heuristic Evaluation and Cognitive Walkthrough, as described below.

5.4.1 Heuristic Evaluation

Heuristic Evaluation is a method for finding usability problems in a user interface design, by having a small set of evaluators examine an interface and judge its compliance with recognised usability principles, the "heuristics" (see Nielsen, 1992). Heuristic Evaluation may be viewed as having evolved from Guideline Reviews. These were inspections where an interface is checked for conformance with a comprehensive list of guidelines.

Typically, guidelines would be fairly specific, such as:

- Provide displayed feedback for all user actions during data entry; display keyed entries stroke by stroke ...
- The computer should provide some indication of transaction status whenever the complete response to a user entry has been delayed ....

Problems arise with the application of guidelines. Guideline lists can be up to 1000 in length, and require significant expertise to apply. They can be vague, contradictory, or defined at an inappropriate level of specificity. A form of usability inspection was therefore developed, that involves the application of a far more general set of design guidelines referred to as heuristics. Evaluators are trusted to use their experience and intuition to identify whether a guideline makes sense or not in a particular context, and how to apply it. Heuristics focus the evaluator's attention on aspects of an interface that are often sources of trouble, making detection of problems more likely. The original set of usability Heuristics (Nielson, 1992) are:

use simple and natural dialogue	speak the user's language
provide clearly marked exits	provide short cuts
minimise user memory load	provide good error messages
be consistent	prevent errors
provide feedback	

Each heuristic summarizes concepts with which experienced designers should be familiar:

Simple and natural dialogue

- Simplify as far as possible, reduce items to be learnt and remembered.
- Provide as much information as is needed (and no more) when and where it is needed.
- Interface should match task: provide a natural mapping between user concepts and computer system concepts.
- Sequence of operations should match the way users would naturally do things.
- Allow user control of sequences of events.
- Use appropriate graphics and colour.

Speak the user's language

- Stick to user's perspective.
- Avoid restricting naming conventions.
- Use metaphors where appropriate.

Provide clearly marked exits

- All dialogue boxes should have cancel/escape.
- Use undos.
- Allow interrupts.

Provide short cuts

- Allow frequently used operations to be performed rapidly.
- Use abbreviations, function keys etc..
- Reuse interaction history.
- Use default values.

Minimise user memory load

- Recognition is easier than recall, exploit computer's ability to store.
- Use dialogue boxes to allow selection of options.
- Use default values to show typical values or formats.
- Provide hints as to valid input, i.e. ranges.

- System should be based on a small number of pervasive rules.

Provide good error messages

- Should be phrased in clear language.
- Should be precise.
- Should be constructive.
- Should be polite.
- Use multiple levels.

Be consistent

- A specific command or action should always have the same effect.
- Format information consistently on all screens and dialogue boxes.
- Be consistent with other interfaces (where appropriate).

Prevent errors

- Use selection in preference to typing.
- Confirm commands.
- Avoid commands that are too similar.

Provide feedback

- System should continuously inform the user what it is doing, and how it is interpreting user input.
- Don't wait for errors (positive feedback, partial feedback).
- Avoid abstract and general terms.
- Vary persistence appropriately.
- Warn of system response times where appropriate.

An Heuristic evaluation is typically undertaken by a team of 3-5 evaluators, including HCI experts and software developers. A set of typical user tasks would be walked through, and features of the interface examined. Heuristic Evaluation is less formal than most evaluation techniques, and is quicker and cheaper to implement. It can be used with early prototypes or paper mock-ups, and is therefore valuable in early stages of the design process, or where time and money are limited. However, research has shown the technique to be less comprehensive

than experimental evaluation. A study by Desurvire et. al. (1993) showed that an application of the Heuristic Evaluation technique identified only 44% of the problems found through user observation.

Many, if not all, of the heuristics are appropriate to some forms of assistive technology. However, the heuristics are derived from guidelines, most of which assume that an able-bodied person is interacting with a graphical user interface or command-line system, using a keyboard or mouse, whilst viewing a VDU. Interface design for assistive technology also has to consider the diversity of the potential user group, combinations of possible input devices, various feedback devices, and a system that may be adaptable. It is theoretically possible to generate a comprehensive yet unified set of specific guidelines dealing with voice recognition, voice synthesis, gesture recognition, scanning systems, multi-modality, adaptability, user diversity e.t.c. However, research so far has typically addressed these issues separately.

A reasonable conclusion may be that standard heuristics provide a useful tool to help designers of assistive technology, provided they are used with the understanding that they are not sufficiently extensive to address issues that are unique to the field. This approach was tested during the development of the interface for the Middlesex Manipulator. As described in Chapter 8, a group of 5 undergraduate Computer Science students undertook an heuristic evaluation of a prototype of the interface. Potential usability problems identified were then compared with problems that occurred during user observation.

5.4.2 Cognitive Walkthrough

Cognitive Walkthrough (Polson and Lewis, 1992), is an evaluation methodology that focuses on 'ease of learning'. The technique is adapted from established software design walkthroughs, and is based on the model of Learning by Exploration described below. The procedure consists of stepping through actions and considering the behavior of the interface and its effect on the user. Actions are identified that are difficult to choose or perform. The result is a list of claims, as to why the given steps may be problematic. These are based on theoretical argument, empirical data or common sense gained through experience.

Polson & Lewis (1992) present a model of the cognitive processes involved in successful exploration. This allows for evaluation criteria to be extracted from the model to allow a designer to identify points during an interaction where a typical user is likely to fail. The model of learning by exploration is based on two theories: the Theory of Actions (Norman, 1986), and the Construction Integration Model (Kintsch, 1988). Norman (1988) presents a summary of the processes involved in performing and evaluating a task:

A Theory of Actions:

- Establishing the goal
- Forming the intention
- Specifying the action sequence
- Executing the action
- Perceiving the system state
- Interpreting the system state
- Evaluating the system state with respect to the goals and intentions.

These seven stages may be viewed as an approximate model of user activity, not a complete psychological theory. In reality the stages may not be discrete entities, and would be likely to exist in parallel.

The Construction Integration Model, Kintsch (1988), describes how users integrate representations of perceptual input with background knowledge to form a representation, which will allow them to complete a task. A goal structure is constructed from a description of the user's task. A goal structure is similar to a goal hierarchy used in task analysis, with a top goal representing the overall task, intermediate goals defining a task decomposition, and lowest level goals describing actions. Goals are represented by propositions. These are linked to: other goals, propositions representing background knowledge, propositions representing objects in the environment, and to actions. These links are associative, and may be regarded as allowing activation to flow from top level goals through connecting links to lower level actions. When an action becomes sufficiently activated it is executed. Any response by the system is observed, deactivating any accomplished goals, and building new propositions. These propositions

represent new goals, and changes in the environment caused by the last action. The new propositions are linked to the existing network, allowing activation to continue to spread through the network, until the top level goal is achieved. The goal structure is initially incomplete, and fragments are generated through interaction. The method aims to establish whether cues and background knowledge are sufficient to generate an appropriate goal structure.

The model of learning by exploration combines the ideas presented above to describe how a user may learn to use a system by a process of exploration. As with construction integration, a complete goal structure is initially unknown, but is discovered by repeating the following sequence of steps:

Model of Learning by Exploration

- Goal setting: users start with a rough description of what they want to accomplish.

- Exploration: users explore the system's interface to discover actions useful in accomplishing their current task.

- Selection: users select actions they think will accomplish their current task, often based on a match between what they are trying to do and the interface's description of actions.

- Assessment: users assess progress by trying to understand system responses, thus deciding whether the action they have just performed was the correct one, and to obtain clues for the next correct action.

The cognitive walkthrough procedure simulates the user's cognitive processes as the user interacts with an interface. In its original form, a printed set of specific questions is used, designed to reflect the cognitive model outlined above. As this was perceived as being time consuming by evaluators, a simplified version of the Cognitive walkthrough was developed. This involves walking through typical user tasks with a detailed design description. At each stage of interaction the following questions are asked, relating to the: goal setting, exploration, selection, and assessment stages of the model of learning by exploration.

- Will the user form the appropriate intention?

- Will the correct action be made sufficiently evident to the user?

- Will the user connect the correct action's description to what they are trying to do?

- Will the user interpret the system' response to the chosen action correctly, that is, will the user know if he or she has made a right or wrong choice?

The Cognitive Walkthrough technique, like Heuristic Evaluation, may be used early on in the design cycle, as only a design description is required. However, the technique has proved less effective than Heuristic Evaluation (Desurvire, 1993). This is partly because the technique focuses on 'ease of learning', and focuses on problems rather than solutions. These factors have contributed to the technique being less popular as a usability inspection method, than Heuristic Evaluation.

Whereas Heuristic Evaluation is derived from the experience of system designers, the Cognitive Walkthrough technique is based on models of user cognition. However, as was the case with designer experience, the models do not cater for the diversity of potential users of assistive technology. In order to answer the questions that form the basis of the technique, it has to be possible to think in terms of a 'typical' user. As discussed in the previous chapter, a need has been demonstrated for assistive technology to cater for the differences between individual users. However, it is feasible that the technique may be of value if systems are being designed or configured for clearly defined sub-groups of the physically disabled population, for example, those with recent high-level spinal cord injury.

5.5 Experimental Evaluation

Experimental evaluation forms a valuable tool for interface design, but as it consists solely of the application of scientific method to interface evaluation, it is discussed here in less detail than the previous techniques. Formal experiments are undertaken where measures of usability can be expressed in a quantitative form, such as: task success, task completion time or error rates. Ideally, a hypothesis is tested within a controlled environment, with a suitable sample of the user population, allowing for data analysis to test for statistical significance.

Experimental evaluation is a relatively expensive form of evaluation, and is typically reserved for critical stages of product development, such as market analysis, feasibility testing or product acceptance. Informal experimentation may be employed where more subjective user feedback is required, such as perceptions of interface complexity. These typically take the form of user observations, questionnaires or interviews. If the appropriate conditions are met, particularly in questionnaire design, statistical analysis may be used to interpret the results obtained.

Informal experiments involving user observation, questionnaires and surveys have been used extensively during the development of assistive technology, and form an important part of most product design cycles. However, the use of formal experiments as is common in 'mainstream' HCI is problematic. As with the previous techniques, this is partly due to the extreme diversity of the potential user group. HCI experiments are often designed to quantify the effect of varying a particular feature of the interface - the independent variable. For example, the feature's effect on time, or error rates (dependent variables) would be measured. If the subject group consists of a representative sample of the user population, and all extraneous variables are controlled, then the result may hold validity for the user population as a whole.

The ability to vary features of an interface is of particular interest to designers of systems that are required to be adaptable, or use novel input devices. For example: adjusting the number or order of options on a menu, varying the speed of a scanning system, adjusting the size of a vocabulary of gestures e.t.c. However, if the potential user group is too diverse, then an experiment cannot

easily be designed around a representative sample of users, and therefore the general effects of these variations can not be estimated. Again, an experiment could be designed to address a sub-group, or even better, to address a specific individual. However, the cost and time required to do this for every individual, and for all possible combinations of interface features would be inhibitive.

5.6 Summary

As discussed in section 5.4, the development of interactive systems for rehabilitation robotic devices, and for assistive technology in general, requires the consideration of a number of factors that are not formally addressed by established HCI evaluation methodologies. These may be summersied as being:

- *Diversity*. If systems are to be developed for a significant proportion of the disabled community, then the variation in user functional ability is vast. Most evaluation methodologies are either based on the idea of the typical user, or require that representative samples of the user population are available.

- *Multimodality*. Limitations in user's functional ability, as well as stringent safety concerns, suggests that the development of systems that may employ more than one mode of communication would be advantageous. Existing evaluation methodologies provide no formal way of examining the effects of the simultaneous use of two or more input devices

- *Adaptability*. The design of systems that can be configured to match user's functional ability, requires that an assessment of functional ability forms part of the evaluation process.

- *Novel Input Devices*. The employment of novel input devices, such as gesture recognition systems and voice systems, introduce factors that are not catered for by standard evaluation methodologies.

Informal experimentation is a valuable tool for the design of assistive technology, and this chapter has discussed how other evaluation methodologies may be applied to assistive technology in a limited or modified form. An Example of applying a Heuristic evaluation is provided in Chapter 8. The possibility of adapting GOMS task analysis into a form suitable for use in configuring adaptable user interfaces was also discussed. This may be of particular use, where device and user characteristics should form a part of the configuring process. The methodology is outlined and tested in Chapter 9 of this thesis.

Chapter 6

The User Interface System

This chapter describes the development of the User Interface System (UIS) for the Middlesex Manipulator. User tasks identified in Chapter 4 are modeled by providing descriptions of the actions that constitute a task. The descriptions were then refined, and represented using Hierarchical Task Analysis (HTA), a method for decomposing tasks into goals, sub-goals and lower-level actions as described in Chapter 5. This allowed for the functionality required of the UIS to be grouped into a number of modes of control, allowing for a modular approach to system design.

In accordance with the design requirements specified in Chapter 4, the objective of the work reported in this chapter, was to define a software architecture that allows specific system implementations to be adapted for the user in terms of the modes of control selected, input devices, feedback devices and style of interaction.

6.1 Modelling User Tasks

The objective of modelling user tasks, was to define a set of modes of control that provide the system functionality required. As there are a large number of robotic systems in existence, the results of this approach to some extent may be anticipated. For example, systems typically provide a combination of joint control, movement through cartesian space, pre-taught positions, or pre-programmed routines. However, as shown below, the modelling provides :

- additional detail as to the appropriate structure of control modes;
- a formal description of system functionality which may be evolved into a design solution;
- a model against which actual system use may be compared, to allow design modification or verification; and,
- a model that can be used to assist in configuring a system for a specific user (constituting a novel use of task analysis as presented in Chapter 9).

As discussed in Chapter 5, Hierarchical Task Analysis (HTA) is a method for describing the process of problem solving by decomposing a task into goals, sub-goals and lower-level actions. HTA is used as a tool in software development, typically by describing how users undertake tasks with existing systems, allowing for the systems to be updated or replaced. For the current application, user tasks were first described informally by considering how an able-bodied person may undertake the task, or how similar tasks are achieved with existing rehabilitation robotic systems (video footage of the Manus, Handy-1, and RAID systems were employed for this purpose).

Of the top eighteen tasks, three are expanded below, as these were found to be representative in terms of the lower level actions identified.

i) Pick and place - can be achieved through the combination of joint and cartesian control, with pre-taught positions being used where appropriate.

ii) Painting - includes the use of pre-programmed routines that should be performed relative to the current position of the end-effector.

iii) Feeding - includes the use of a pre-programmed routine that utilizes pre-taught absolute positions.

6.1.1 The pick and place task

Figure 6.1 provides a description of the components of a typical pick and place task. Consideration of the constraints involved when undertaking the task with a robotic system, allows for control modes to be associated with the various components of the task.

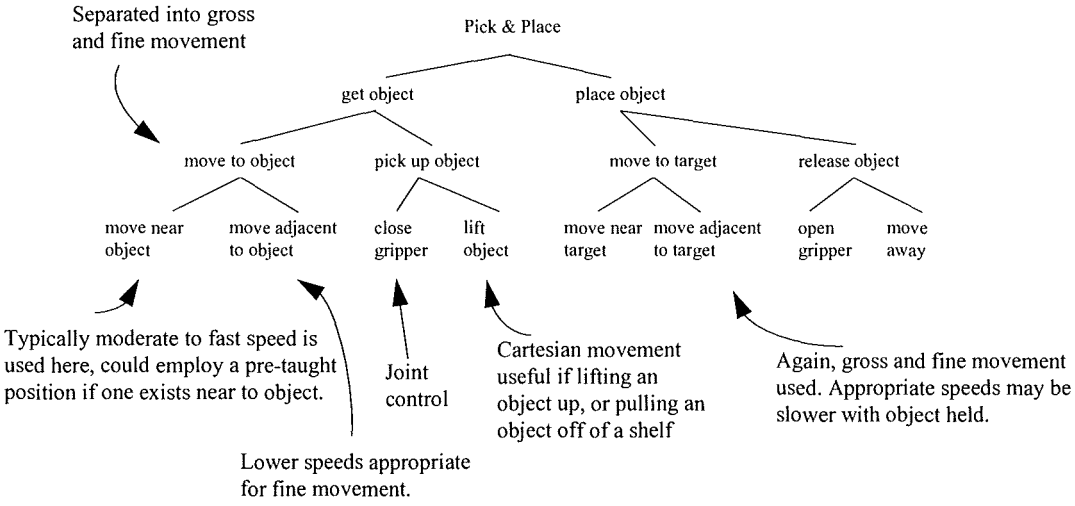
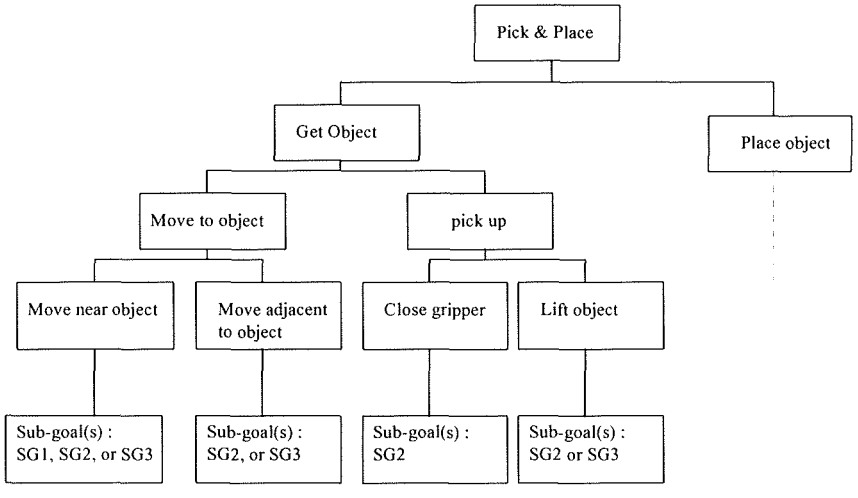


FIGURE 6.1 INITIAL PICK & PLACE TASK DESCRIPTION

The task may then be represented in HTA form, making reference to sub-goals that need to be further defined.



Sub-goal descriptions

SG1 - Select and move to a pre-taught position, at an appropriate speed
SG2 - Select and move a joint , at an appropriate speed
SG3 - Move in a specific direction at an appropriate speed

FIGURE 6.2 TOP LEVEL HTA DESCRIPTION OF PICK AND PLACE TASK.

Definition of the sub-goals provides information regarding their structure, and provides an indication of the control commands that the UIS will be required to support, and the order in which they may occur.

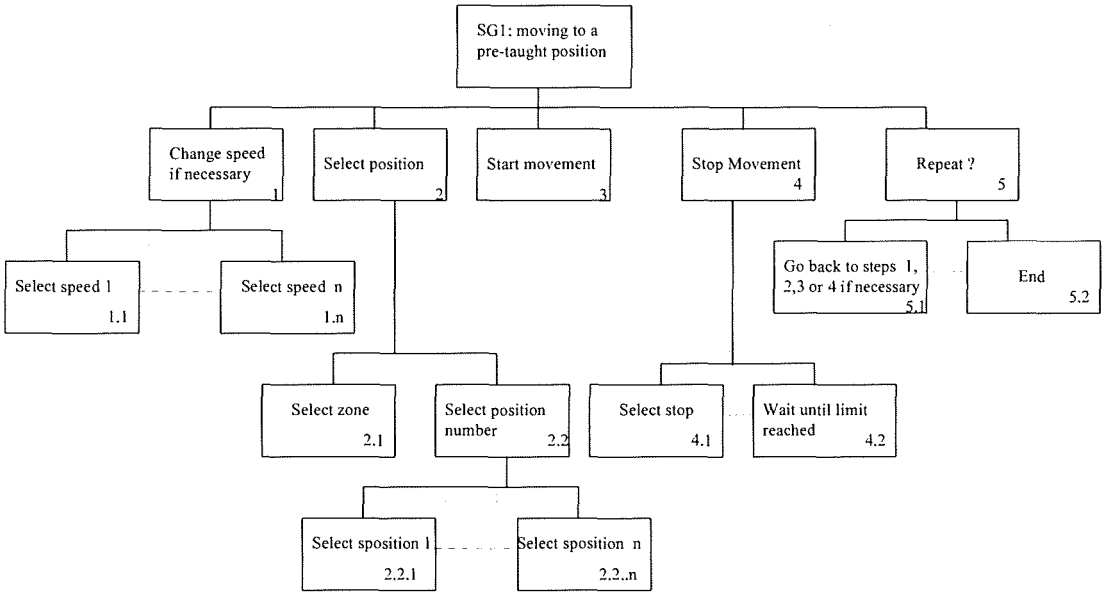


FIGURE 6.3 SUB-GOAL TO MOVE TO A PRE-TAUGHT POSITION

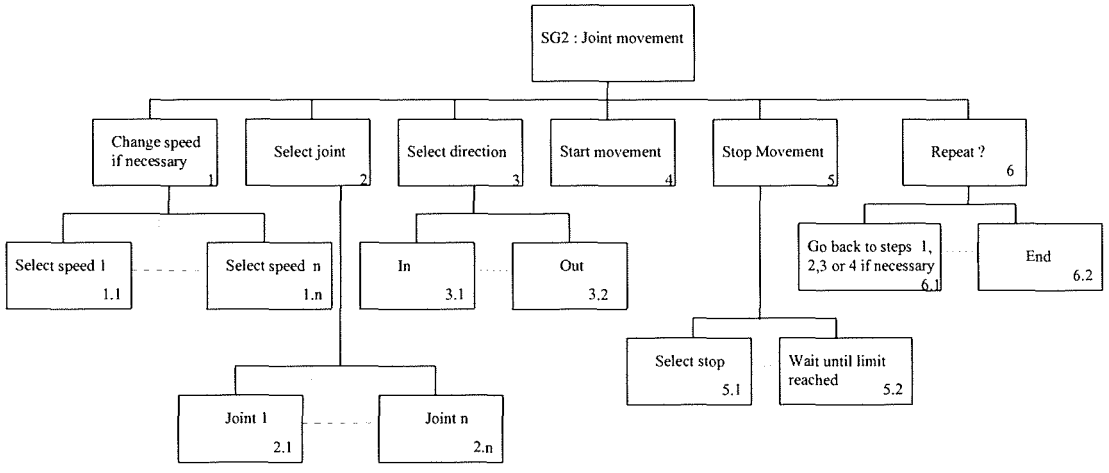


FIGURE 6.4 SUB-GOAL TO MOVE A JOINT

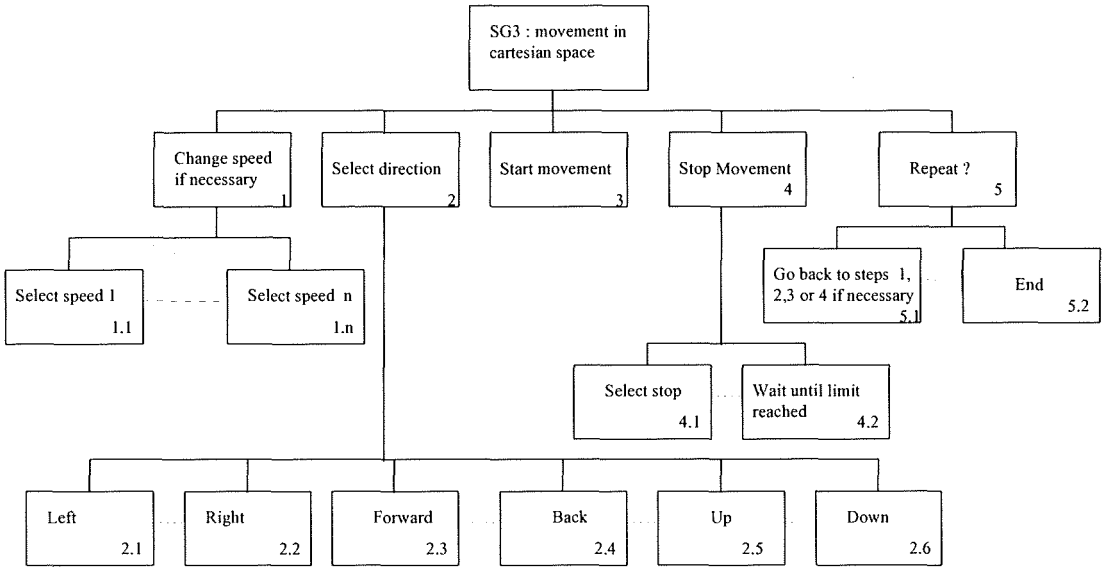


FIGURE 6.5 SUB-GOAL TO MOVE IN A PARTICULAR DIRECTION

The model provides a description of one way of accomplishing a task. However, the analysis cannot ensure that the structure identified will optimize the usability of an interface design based on the model. This will be dependent on :

- how accurately the initial task description reflects tasks being undertaken with the Middlesex Manipulator (unknown until the Manipulator has been used);
- how frequently and in what order the lower-level actions are performed (effects breadth *v* depth and ordering of menu options);
- how many, and what type of input devices are employed;
- the form of user interaction employed; etc.

The model, along with those developed below, may act as a requirements specification to be evolved into a design solution assisted by the evaluation methodologies outlined in Chapter 4. However, as the system is to be designed to be adaptable to specific users, the objective is not to resolve the issues listed above for the general case, but to allow them to be resolved for each individual case. Hence, the model presents an appropriate level of modularity for system design, i.e. the modes of control required, corresponding to the general subgoals used in each task description.

6.1.2 The painting task

A description of several of the prioritized user tasks, suggested that the system should allow for pre-programmed routines that may be executed relative to the current position of the manipulator's end-effector. This is the case where tasks require that a particular trajectory is repeated at different absolute positions in the workspace, such as : shaving, combing hair, gardening, and painting. A model is developed with reference to a painting task in figures 6.6 to 6.8 below.

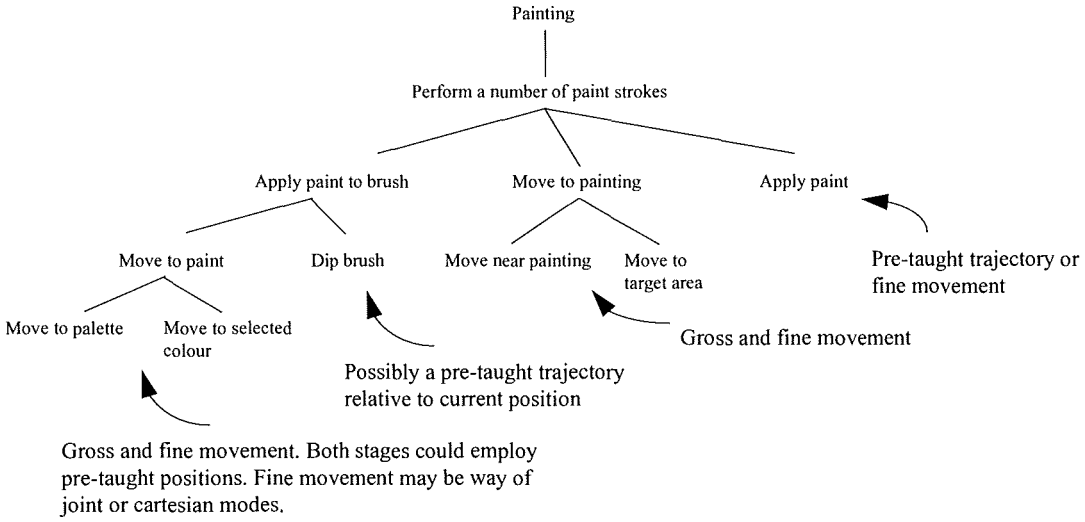


FIGURE 6.6 INITIAL DESCRIPTION OF PAINTING TASK.

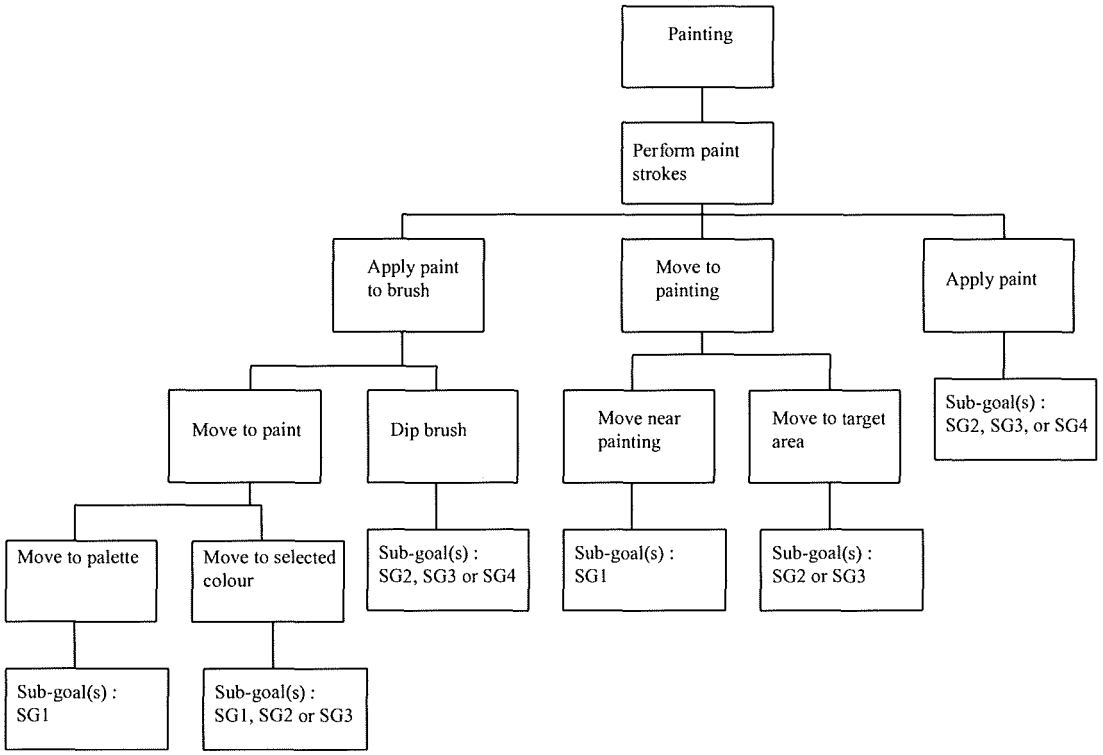


FIGURE 6.7 TOP LEVEL HTA DESCRIPTION OF PAINTING TASK.

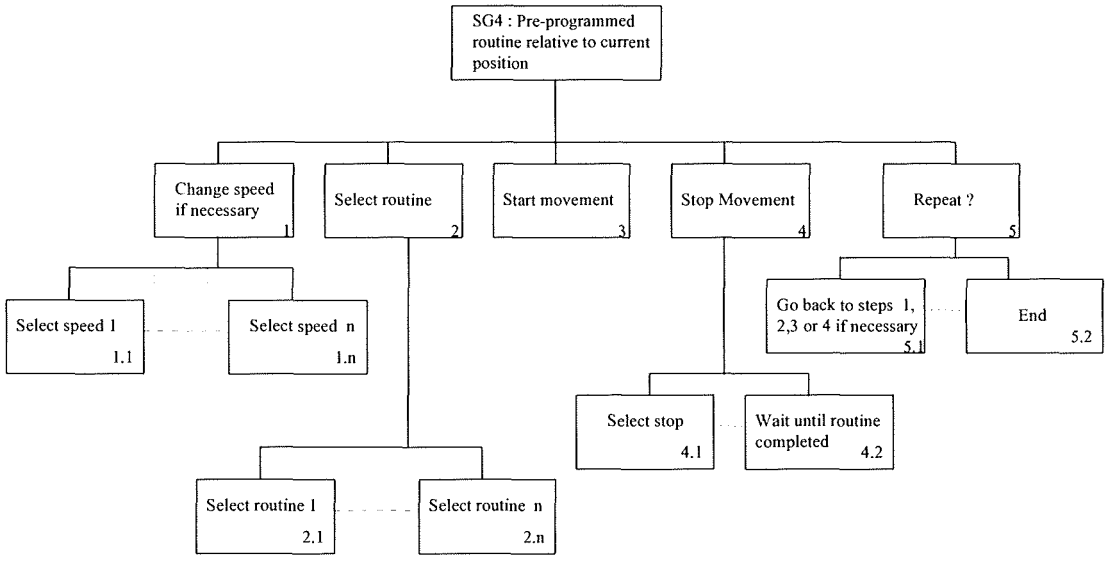


FIGURE 6.8 SUB-GOAL TO EXECUTE PRE-PROGRAMMED ROUTINE

6.1.3 The feeding task

The following description of a feeding task is similar to that employed by the Handy - 1 feeding aid.

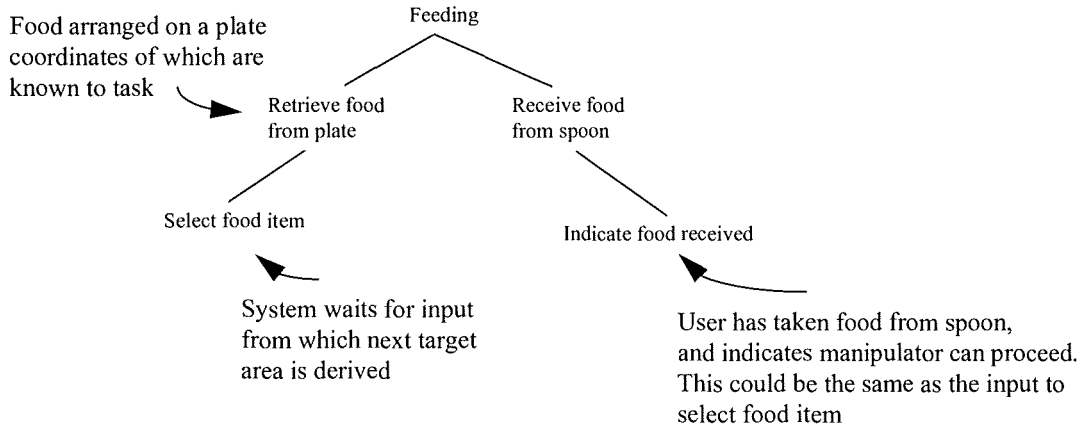


FIGURE 6.9 INITIAL DESCRIPTION OF A FEEDING TASK

This may be achieved with a mode of control that accesses pre-taught absolute positions in a pre-determined order, can be programmed to wait for user input, and can accept user input to determine next target position. An appropriate model is shown in figure 6.10.

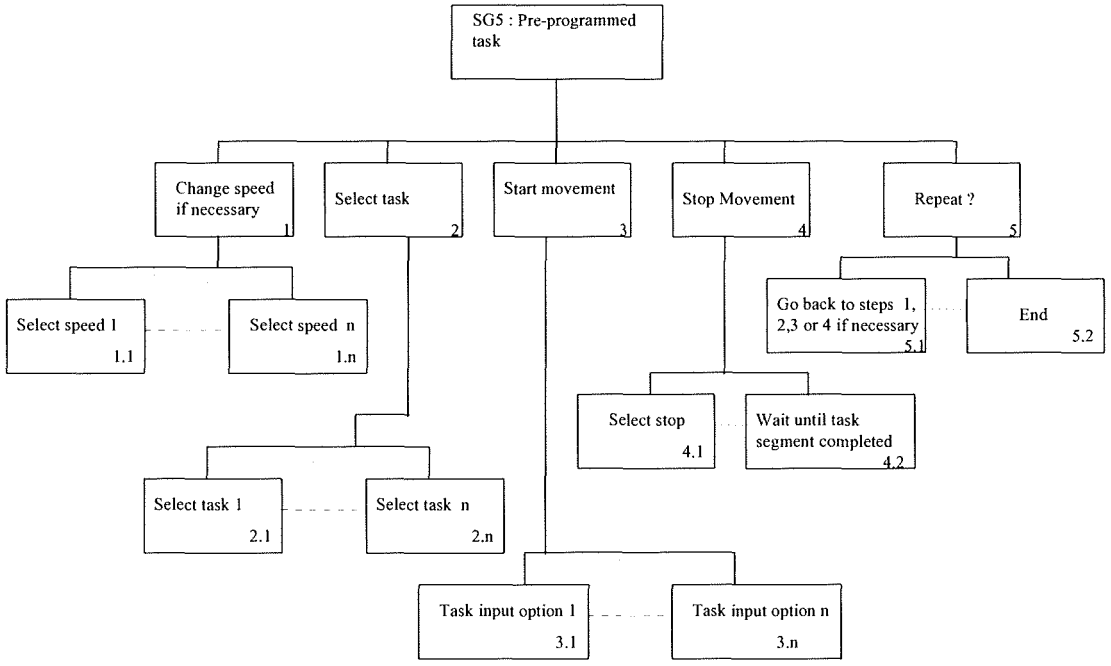


FIGURE 6.10 SUB GOAL TO EXECUTE A PRE-PROGRAMMED TASK

6.2 Modes of control

The task analysis described so far has identified the following possible modes of control :

- positional (movement to a pre-taught position)
- joint (movement of a specific joint)
- cartesian (movement of the end-effector in cartesian space)
- routine (performing a pre-taught trajectory relative to current position)
- task (executing a pre-taught task that accesses pre-taught absolute positions)
- speed (setting manipulator speed levels)

Further modes are required to allow the teaching of positions, routines or tasks :

- teach position (record the current position of the end-effector as a pre-taught position).
- teach routine (record a trajectory)
- teach task (record a task).

The recorded positions are grouped into zones to assist with ease of recall. Each zone contains a number of positions. There are currently up to 8 zones, each containing up to 8 different positions[1]. A position is recorded by first moving the Manipulator to the target area using either cartesian or joint modes. The angular settings for each of the Manipulator's axes is then written to a file, using the teach position mode structured as shown in figure 6.11.

[1] The potential number is much greater, limited only by the available disk space and acceptable menu-depth.

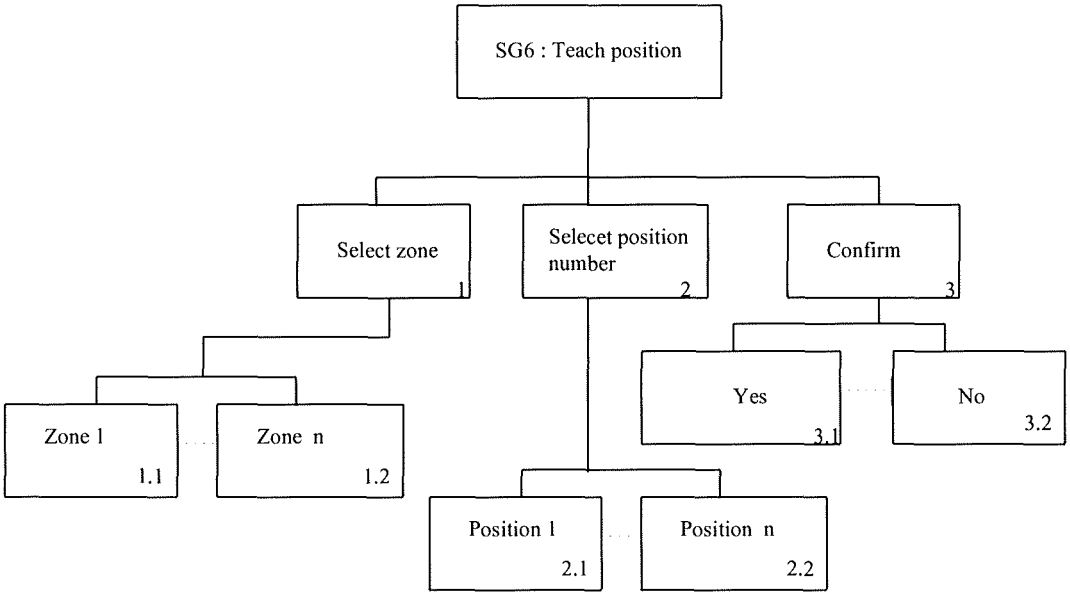


FIGURE 6.11 STRUCTURE OF TEACH POSITION MODE

Pre-programmed routines may be taught by selecting a routine number and recording the current position as routine origin. The Manipulator is then moved using joint or cartesian modes, with the offset from the origin for each axis recorded at points along the trajectory.

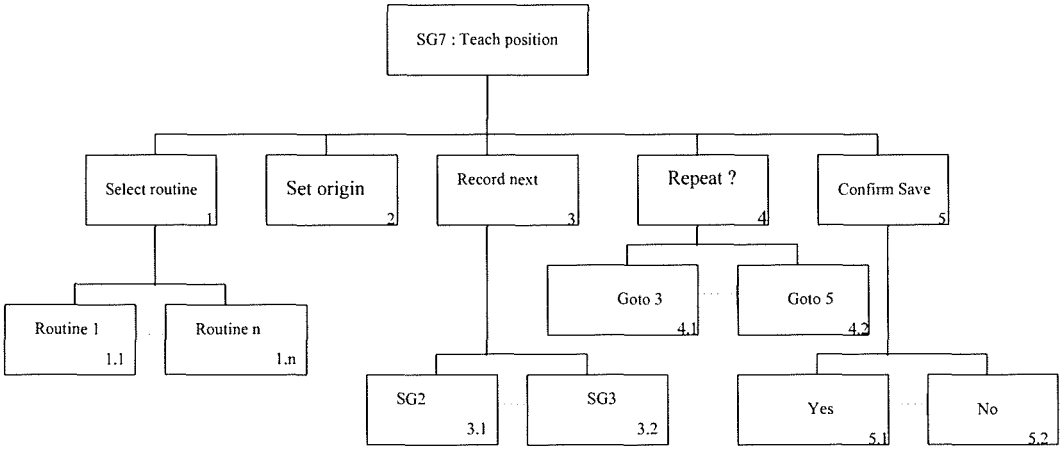


FIGURE 6.12 STRUCTURE OF TEACH ROUTINE MODE

Currently, the definition of pre-taught tasks is achieved using a text editor, as opposed to via options presented by the UIS. A template may be used to create C code that can be compiled into a Dynamic Link Library (DLL) and called by the UIS task control mode. The DLL coordinates communication to the motor control system, in response to input from the user. Clearly this

approach does not allow for the easy recording of tasks by end users, and would be required to be replaced by a more user-friendly approach during further system development. An example template is provided in Appendix D.

6.3 The UIS software architecture

The review of rehabilitation robotic systems presented in Chapter 2 provided evidence to suggest that systems should be designed to provide a limited amount of functionality for novice users, and increased functionality for more experienced users. The number of different control modes identified above, and their complexity (particularly of the teach modes), tends to suggest that a system with all control modes present would be unsuitable for a large number of potential users. This section describes a software architecture that allows for systems to employ an arbitrary number of modes of control.

To allow systems to be configured to match specific individual requirements, adaptability must allow selection of :

- The number and type of input and feedback devices.
- The number and type of modes of control.
- The number of selectable items within a mode of control (number of speed levels, pre-taught positions, etc.).
- The ordering of options within a mode of control (to reflect user priorities).

The components of the user interface system may be described using an object oriented analysis and modelling tool such as the Unified Modelling Language (UML). UML was chosen to provide a high level description of the system, as it is rapidly becoming an industry standard (see Booch et al. 1998).

The principle system components identified during the task analysis and requirements specification are :

- a motor control system;
- modes of control; and,
- interface components.

Figure 6.13 provides a UML diagram illustrating the relationships between these components[2]. The model allows for multiple interface components, each of which may be responsible for processing user input, dispatching an input command to a mode of control module, and displaying the current set of possible input commands. Multiple modules may be present, representing possible modes of control. These are responsible for receiving the input command, maintaining the set of current possible input commands, dispatching motor control commands to the motor control system and monitoring the status of the motor control system.

The model shows that many-to-many relationships are possible between interface components and modes of control, and highlights the potential complexity of message routing between modules.

[2] The UML notation used is specified in appendix I.

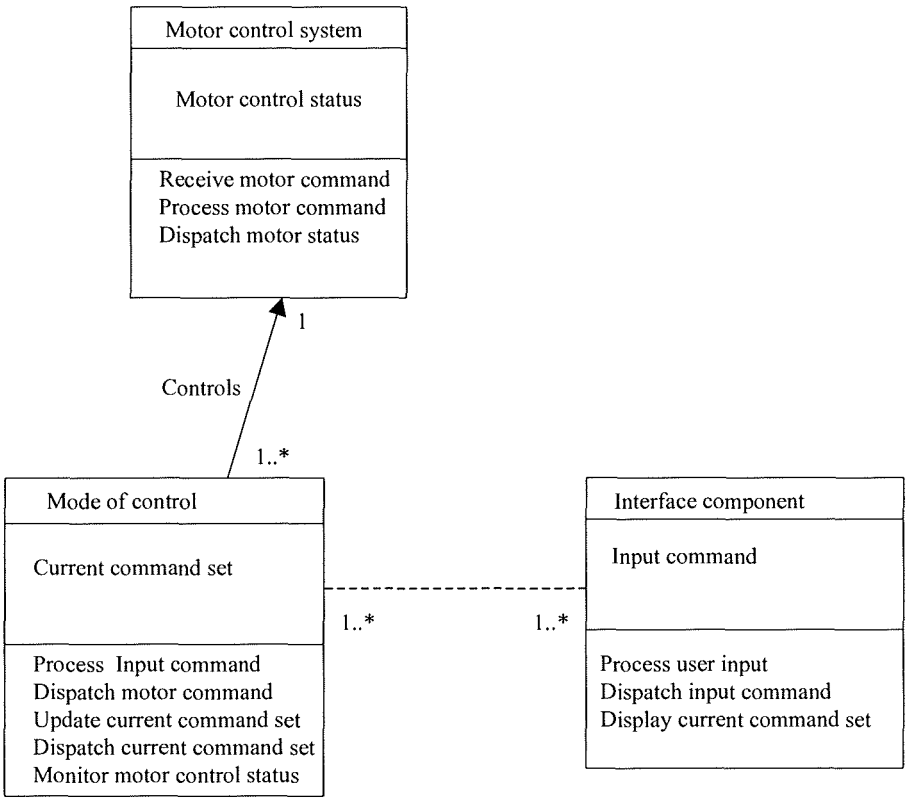


FIGURE 6.13 USER INTERFACE SYSTEM COMPONENTS REPRESENTED AS UML CLASSES

This issue was addressed by introducing an additional module referred to as the Dialogue Manager as shown in figure 6.14. The dialogue manager is responsible for activating a control module, in response to an interface component dispatching the first in a series of input commands. The active control module will then determine the subsequent command sets, which are forwarded by the dialogue manager to the interface components present. This sequence of events is portrayed as a UML sequence diagram in figure 6.15.

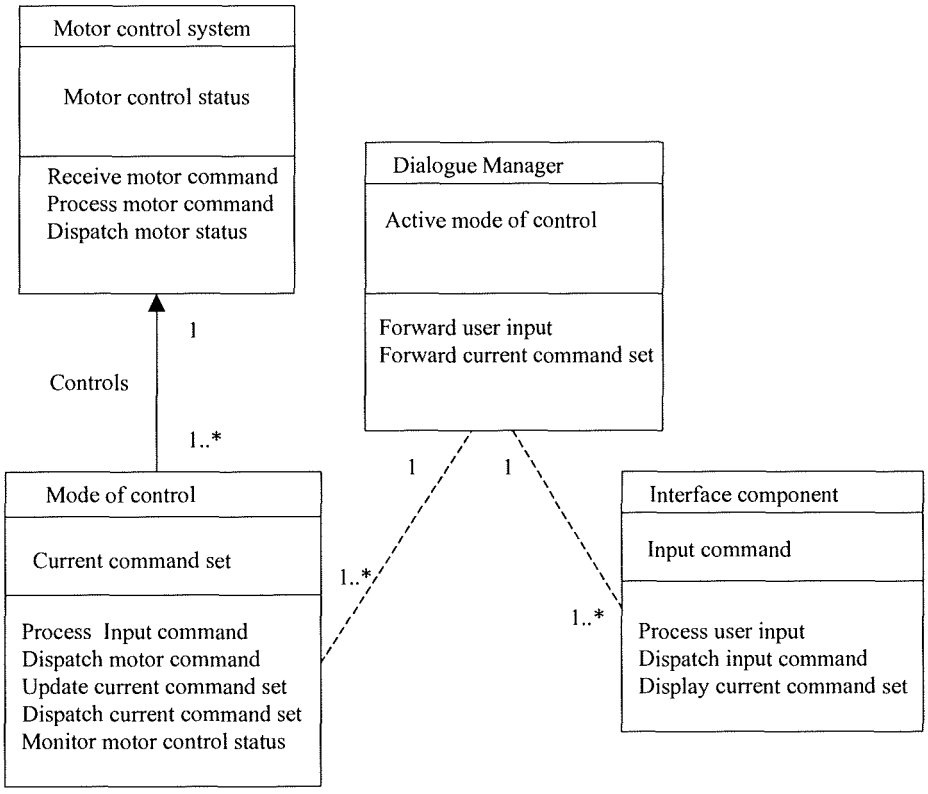


FIGURE 6.14 USER INTERFACE SYSTEM WITH DIALOGUE MANAGER CLASS

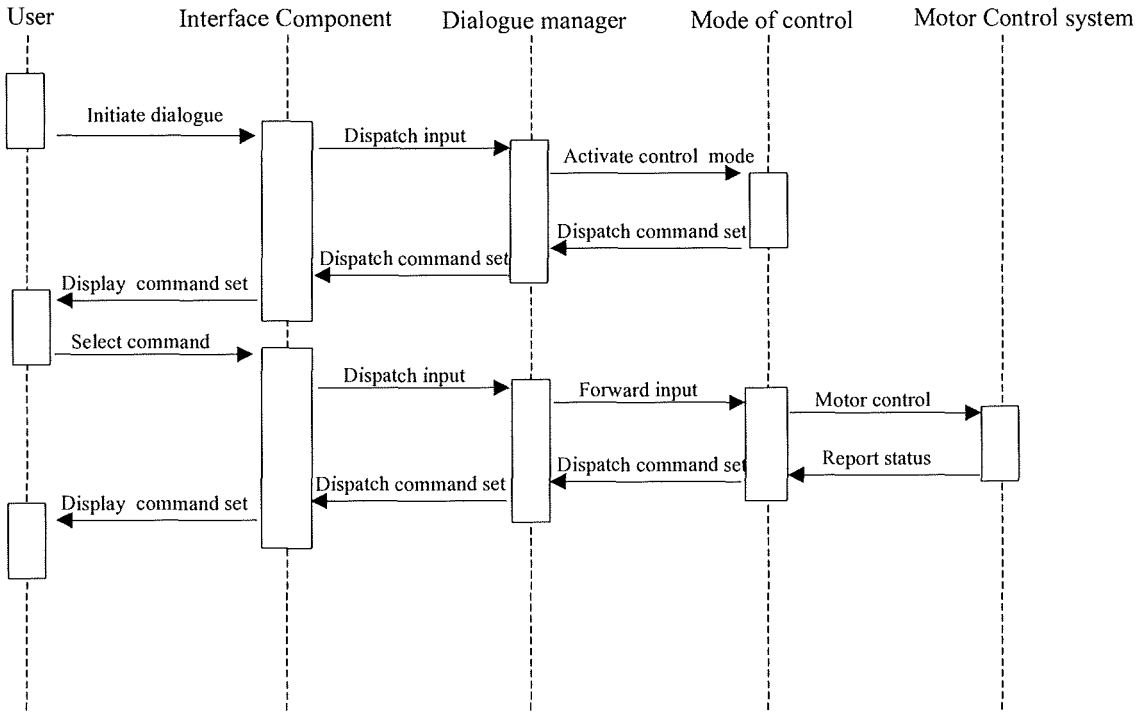


FIGURE 6.15 UML SEQUENCE DIAGRAM OF INTER-MODULE INTERACTION

As discussed in Chapter 4, a PC was selected as the platform for the user interface system This allowed for the development of Windows applications that can run in a multi-tasking environment. Modularity was achieved by developing modules as separate applications that will be either present or absent within specific system implementations. Configuring a system would simply be a process of loading and running the selected applications.

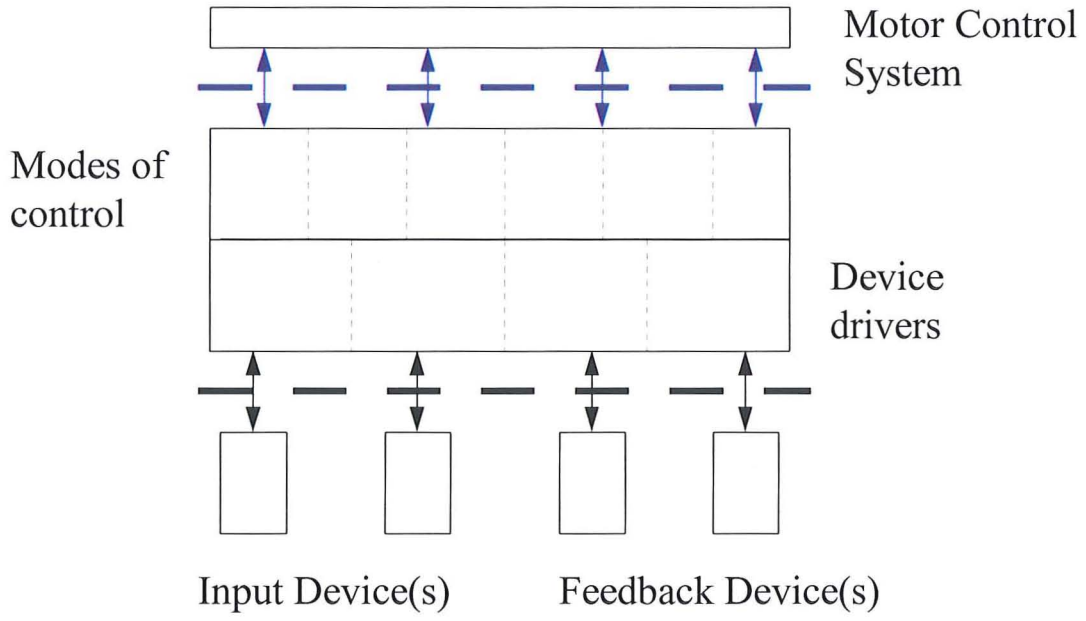


FIGURE 6.16 A MODULAR USER INTERFACE SYSTEM

Similar architectures have been developed for artificial intelligence applications, such as the blackboard model (Erman et. al., 1980), and the agent model (for example, Brown et. al. 1995). The Blackboard model employs a number of separate processing modules referred to as knowledge sources. Each knowledge source periodically examines the blackboard, a global data structure, and is designed to react to a specific set of conditions, generating an output which updates the blackboard. Collectively the knowledge sources contain all of the knowledge required to solve a problem, and they cooperate in iterating the blackboard towards a solution state. Processing that employs agents is similar, in that each agent has a specific responsibility, and acts autonomously in accordance with its responsibility, but cooperates with other processing agents within the problem domain.

As shown if figure 6.17, a similar approach was taken for development of the UIS. Each mode of control is regarded as a module with specific responsibilities, as are the device drivers that communicate with input and feedback devices. A module referred to as a dialogue manager coordinates communication throughout the system.

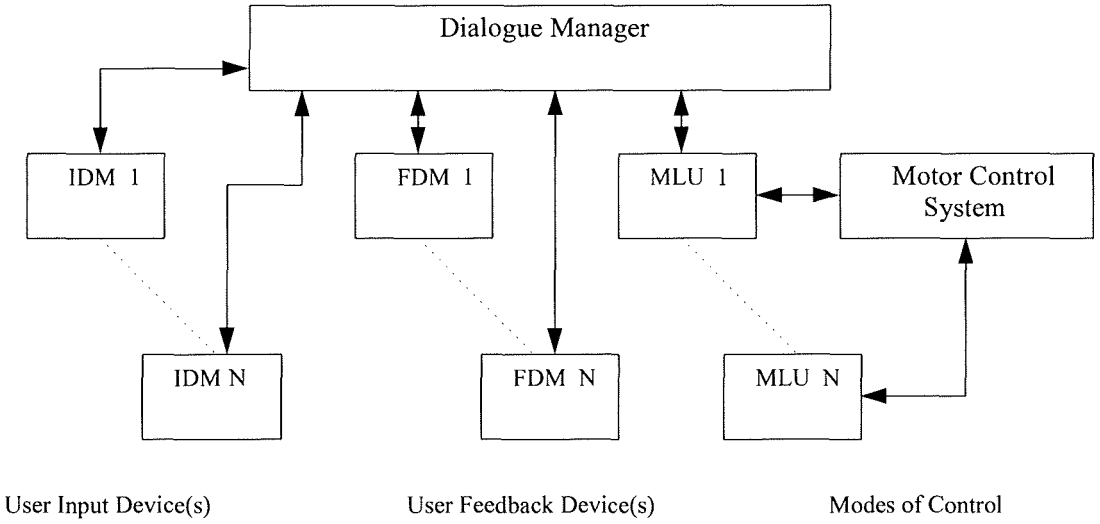


FIGURE 6.17 SYSTEM COMPONENTS

IDM	Input device module. Effectively a device driver for a particular type of input device, but also determines style of interaction (i.e. scanning system, direct menu selection etc).
FDM	Feedback device module (Device drivers for an LCD screen, voice synthesis etc.).
MLU	Modal logic unit. Implements the logic required for a specific mode of control.

The system may be regarded as a single-client multiple-server environment, with the Dialogue Manager acting as client. Communication between modules is achieved through the use of dynamic data exchange (DDE). DDE is a method of inter-process communication, using shared memory to pass data between applications, and a protocol to synchronize communication.

Message passing between the applications that represent the different modules of the UIS is achieved using Window's Dynamic Data Exchange Management Library (DDEML). The

DDEML is a dynamic link library (DLL) provided with Windows to manage DDE conversations. When the UIS is being configured, the Dialogue manager is registered with the DDEML as a client, and may then initiate a conversation with any servers present with the system. The servers also register with the DDEML, and provide details as to the types of conversations that they support.

Data may be transferred from the client to a server (poked), and may be received from a server in three ways :

- the client may request an item of data;
- the client may request to be advised of changes of state within the server (advise loop); and,
- the client may request to automatically be updated with data reflecting the server's change of state.

When the Dialogue Manager is initially run it needs to establish a conversation with at least one IDM and one FDM, and will report an error if not successful. Otherwise, the Dialogue Manger then attempts to establish a conversation with any MLUs present within the system. The number of conversations established will determine the functionality of the system, and the Dialogue Manager is able to instruct the IDM and FDM of the menu options to be provided to the user.

The three initial stages in the process of configuration are therefore :

- Dialogue Manager registers with DDEML as a client;
- All UIS server applications register with DDEML, indicating the type of conversations they support;
- Dialogue Manager establishes a conversation with at least one IDM and at least one FDM;
- Dialogue Manager determines system functionality by establishing conversations with all other UIS server applications present;
- Dialogue Manager instructs FDM and IDM of menu options to be provided to user.

When the user then operates the system, one of the MLUs present will become active, depending on the mode of control selected. The Dialogue manager is then responsible for receiving any user input from the IDM, and dispatching this to the active MLU and the FDM. As each MLU is responsible for implementing the logic corresponding to a particular mode of control, the MLU will provide the Dialogue Manager with the appropriate set of commands for each stage of user interaction. The MLU also provides the output to, or receives input from, the motor control system. Each set of commands received from the MLU is dispatched by the Dialogue Manager to the IDM and FDM.

The activity present within the UIS during system operation is therefore :

- Dialogue Manager receives input from IDM
- An MLU is activated.
- Dialogue Manager dispatches MLU response to IDM and FDM allowing for the next stage of user interaction.
- Dialogue Manager passes new user input to MLU.
- MLU controls the content of user interaction, and establishes a JMCL dialogue with the motor control system.

The task analysis described in section 6.1 identified appropriate modes of control, and provided an outline of the structure of each mode. This allowed for a user command language to be defined, with each command being associated with a specific stage of user interaction. This is

referred to as JUCL (Juvo User Command Language) and is defined in Appendix E. JUCL commands constitute the messages sent between the system modules in response to user input. Appendix F provides code listings for a Dialogue Manager, IDM, FDM, and an example MLU.

6.4 UIS Implementation (Version 1)

Two methods for presenting the JUCL commands to the user were employed. The first was used during initial development and evaluation of the Middlesex Manipulator, and involved presenting the JUCL commands in the form of a flat menu system. Figure 6.18 illustrates a typical interface configuration as presented to the user. The sequence of screens shown, simulates the user moving one of the manipulator's joints. The Windows display is used here to simulate the commands as would be presented on a custom feedback device such as an LCD display unit.

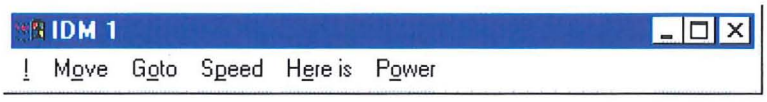


FIGURE 6.18A TOP LEVEL MENU

FIGURE 6.18B JOINT SUB-MENU

FIGURE 6.18C DIRECTION SUB-MENU

FIGURE 6.18D STOP SUB-MENU

When a scanning system is in use, a bar moves across the display, pausing below each command as illustrated in figure 6.19 below.

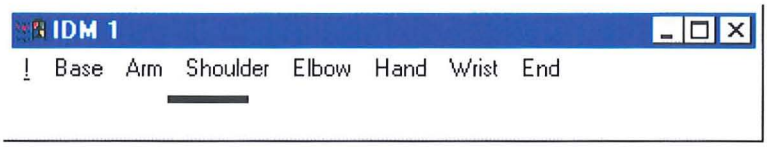


FIGURE 6.19 A SIMPLE SCANNING SYSTEM

As described above, the number of modules present within a specific system configuration would determine which JUCL command options would be presented via the interface. Additionally the mode of input (gesture, voice or trackball), would be determined by the type of modules present.

6.5 UIS Implementation (Version 2)

The second form of interface employed the Microsoft Windows dialog based graphical user interface. This allowed all control options to be presented simultaneously, which would potentially allow for faster task completion. However the interface required the user to be fairly competent when using a mouse or trackball. This requirement led to the development of a 'Head Mouse' as described below.

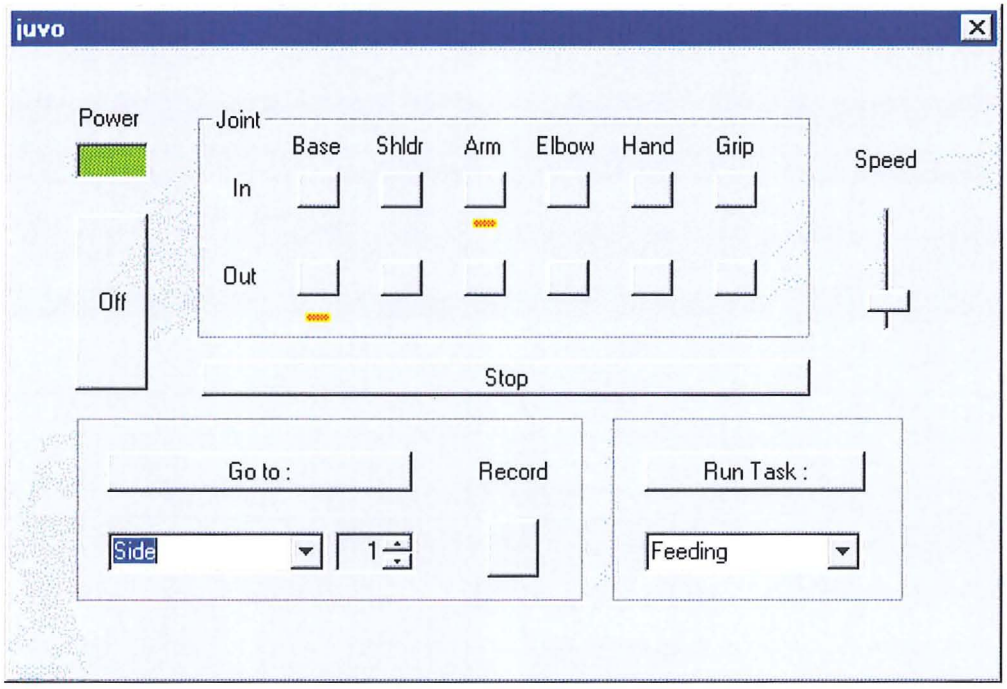


FIGURE 6.20 DIALOG BASED USER INTERFACE

6.6 User interface input and output devices

For initial system evaluation a number of input device modules were developed to allow comparison of various input and output devices.

Trackball or standard mouse input

A trackball may be used to directly select options from the interface when the user has sufficient controlling ability to manipulate a pointing device. Alternatively, if a user is capable of making a number of arbitrary but fairly consistent movements with a trackball, these may be interpreted as gestures as described in Chapter 7.

Voice Recognition

A commercial voice recognition system was employed (Advance Research Technologies, UK), allowing direct selection of items from the menu based user interface. This was achieved by using the vendor's software to train the system to recognise the appropriate set of commands. The system then automatically sends Windows Menu ID messages to the active window when a command is recognised.

Electrolytic Tilt Sensors

Chapter 7 describes the development of gesture recognition software capable of encoding signals from electrolytic tilt sensors mounted on the head or limbs of the user. These were used for either direct menu selection, or adding functionality to scanning systems.

The Head-Mouse

During the final stages of project development a two-axis solid-state tilt sensor was identified (Crossbow Technologies, USA), providing greater operating range (±75°) and a faster response than electrolytic tilt sensors. These were connected to a personal computer via a general purpose input output/card, allowing software to be developed to enable the sensor to be used as a head-mounted pointing device.

FIGURE 6.21 SILICON MICRO-MACHINED TILT SENSOR

6.7 Summary

This chapter discussed how task analysis may be used to provide a description of the functionality required of the UIS. One of the weaknesses of the approach is that the tasks being modeled may not be completely representative of how tasks will be undertaken with the target system. However, the models provide a starting point for system design allowing for an appropriate design solution to be evolved. The task analysis identified appropriate levels of modularity within the system design. This allowed for the development of a highly modular software architecture which may be configured to present varying levels of functionality, different input and feedback devices, and different styles of interaction. The effect of this adaptability on system usability was addressed during the initial system evaluations as discussed in Chapter 8. The following chapter describes the development of a set of input device modules based around gesture recognition.

Chapter 7

Gesture recognition for user input

Chapter 3 of this thesis identified design requirements for the Middlesex Manipulator, which included support for a range of user input and feedback devices. This Chapter describes the development of gesture recognition systems that may be used as input devices for the User Interface System described in Chapter 6. Gestures are monitored by tilt-sensors mounted on either the head or the limbs of the user. User gestures may also be generated from a standard trackball or mouse device.

The chapter explores the suitability of pattern recognition algorithms for encoding user gestures. A Dynamic Programming algorithm and various artificial neural networks were compared as pattern classification systems.

The objectives of this chapter are therefore to :

- describe the development of circuitry that allows tilt-sensors to be used to encode gestures;
- compare a Dynamic Programming algorithm with various artificial neural network configurations for pattern classification;
- describe a Windows application that allows a trackball to be used to encode gestures; and,
- describe how a gesture recognition system may be incorporated into the Manipulator's UIS.

7.1 Gesture encoding with tilt-sensors

Gestures offer an important form of user input for assistive technology. Most physically disabled people are able to partially control at least one part of their body, and the encoding of simple gestures allows for potentially greater signal bandwidth than is achievable with simple switches[1]. A significant amount of research has addressed the use of gestures as a means of communication for assistive technology, for example McEachern et. al. (1994); Harwin and Jackson (1990); and Keates et. al (1997). This research focused on the use of a sophisticated transducer - the Polhemus Isotrack system (McDonnel Douglas Electronics, Colchester VT USA). The Polhemus allows six degrees of freedom to be monitored : x,y,z, pitch, yaw and roll. McEachern describes how data from the sensor may be processed to direct a manipulator towards a target being pointed to by the user, or to follow a path being described by the user's gestures. The major disadvantage of including the Polhemus within the design of a system is its high cost at over £8000 per device.

An alternative and cheaper method of encoding gestures is reported by Harrington et. al. (1995). The approach taken was to mount a set of accelerometers on an arm for the encoding of arm-gestures. Four accelerometers were successfully used to classify eight different gestures, these could be assigned either semantic meaning or numeric values allowing for a user interface system to be navigated. However, the cost of a set of accelerometers, though cheaper than a Polhemous, could still be fairly inhibitive running in to several hundreds of pounds.

The use of electrolytic tilt-sensors allows for the encoding of gestures in 2 dimensions. The sensors offer an attractive solution, as they are relatively cheap at £30 each, and are small (16mm × 7mm) and light (4 g) . However, the sensors are designed for fairly slow moving bodies, having a time constant just below one second (slow enough to allow the electrolytic fluid to settle). This would be likely to effect the potential bandwidth of the system, though this would also depend on the sophistication of the pattern classification algorithm employed.

Two sensors were purchased from The Fredricks Company (Huntingdon Valley, PA, USA). Each sensor consists of a tubular glass envelope partially filled with an electrolytic fluid which contacts metal electrodes. The impedance of the sensor is dependent on the angle of tilt, allowing

[1] Signal bandwidth is used here to refer to the number of distinctly different signals that may be generated with a device.

the sensor to be included in a circuit providing an output voltage proportional to tilt angle, as shown in Figure 7.1 below.

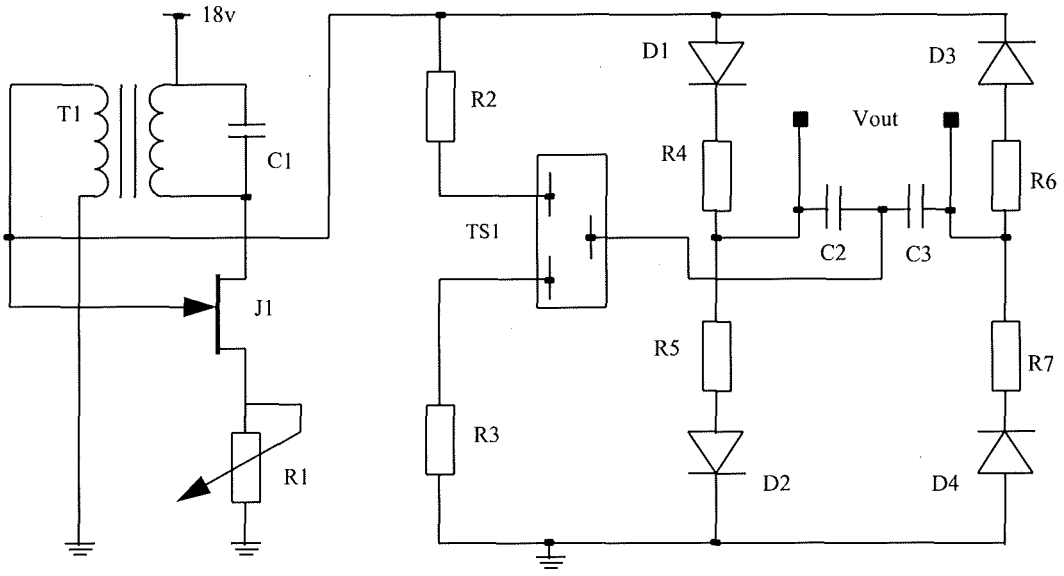


FIGURE 7.1 TILT SENSOR CIRCUIT

R1 22k2, R2 - R5 2k2, R6 - R7 1k4 R8 1k C1 33nF C2 - C3 25uF
D1 - D4 1N4148 J1 J2N3819 TS1 – Tilt Sensor

The circuit shown is adapted from the manufacture's product sheet. An FET is used to form a tuned-drain oscillator to generate the 20 KHz AC signal required for the sensor. The sensor is included in a bridge circuit. The diode configuration ensures that normally the voltages generated across C2 and C3 are approximately equal, and hence the output voltage is approximately zero. As the sensor is tilted, the impedances in series with R2 and R3 will vary, causing a DC voltage to be generated across the output.

Initial tests employed a National Instruments general-purpose data acquisition card (the Lab PC+ card). However, cost constraints would necessitate the development of a purpose built card for the sensor's inclusion within the Middlesex Manipulator control system. For encoding head gestures, the senors were mounted on a baseball cap as illustrated in Figure 7.2.

FIGURE 7.2 TILT SENSORS MOUNTED ON A BASEBALL CAP

Initial tests produced a response to simple head gestures as shown in figures 7.3 and 7.4 below.

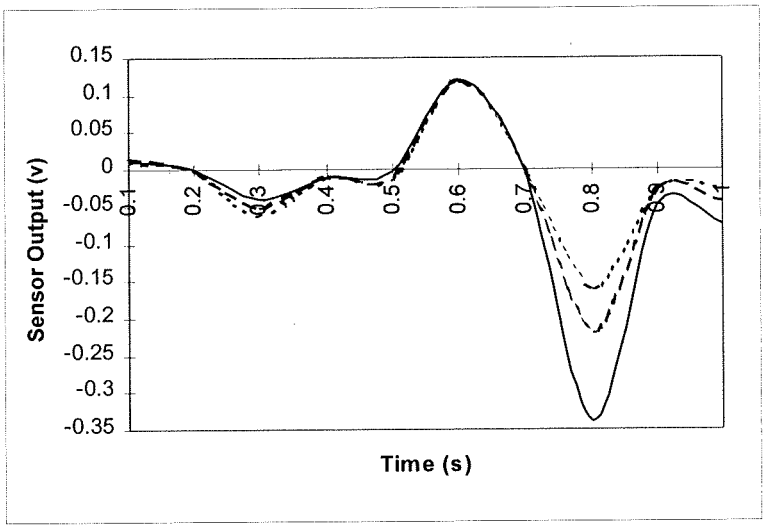


FIGURE 7.3 SET OF 3 GESTURES, PREDOMINATELY IN THE X PLANE

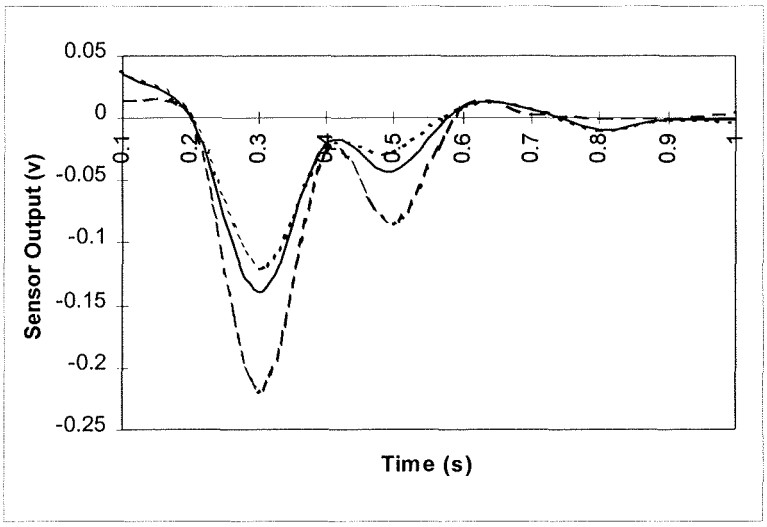


FIGURE 7.4 SET OF 3 GESTURES, PREDOMINATELY IN THE Y PLANE

Each graph shows the output from 2 sensors mounted along orthogonal planes on a baseball cap. The first half of the graph records movement from left to right, and the second from front to back. Each graph shows three attempts at repeating the same gesture. The sensors are designed to provide output voltages of up to 1V. The lower voltage levels of the peaks shown above are mainly due to the slow time response of the sensors. However initial indications were that the sensors could be used as simple switches, or applied to a pattern classification system as described below.

7.2 Pattern Classification

Two approaches to pattern classification were investigated : Dynamic Programming (a template matching algorithm) and artificial neural networks. Research by Tew and Gray (1993), showed that a dynamic programming algorithm (DPA) can be used to successfully classify hand gestures issued by a mouse. The principle advantage of the algorithm is its low computational complexity, and unlike neural networks and Hidden Markov Models, no training of the system is required. As discussed by Tew and Gray (1993), DPAs were popular in the 1970s for speech recognition, and in the 1980s for handwriting recognition but have been superseded by more powerful techniques such as neural networks for these fairly complex classification problems. The objective of the work reported here was to assess their suitability to the classification of

gestures encoded by tilt sensors. This was done by comparing the performance of a DPA with a single layer perceptron artificial neural network (SLP). As the name suggests, the SLP is a network containing just one layer of neurons, and hence offers a level of computational complexity similar to the DPA. The SLP offers fast and reliable convergence during training (Hrycej, 1991), however, its application to pattern classification has been limited mainly due to the SLP's requirement that pattern classes must be linearly separable in feature space (Wasserman, 1989). The work reported here tested whether this constraint would render the SLP inappropriate for the current application.

As discussed below, the performance of the SLP proved superior to the DPA. Tests were then undertaken to compare the more popular multi-layer perceptron artificial neural network (MLP) with the SLP. The backpropagation algorithm was employed for network training. The resulting network was more computationally complex than the SLP, and can require lengthy training times. The objective of this stage of the work was to assess the trade-off between computational complexity and network performance. Finally, a Radial Basis Function training algorithm (RBF) was employed. The RBF offers shorter training times than the MLP, and in certain circumstances can provide similar or better performance (Bishop, 1995).

Part of the work reported below, including implementing and testing the SLP, was undertaken by an MSc student under the supervision of the author (Gellrich, 1995).

7.2.1 The Dynamic Programming Algorithm

The following description of a dynamic programming algorithm is adapted from Tew and Gray (1993). A template representing each gesture to be classified is produced by sampling the gestures at regular intervals, producing a set of template vectors. This is then compared to any subsequently sampled gestures by use of a matrix as described below.

Initially, the vector representing the template, and a vector representing the gesture sample to be classified are applied to the sides of a matrix as shown in figure 7.5a.

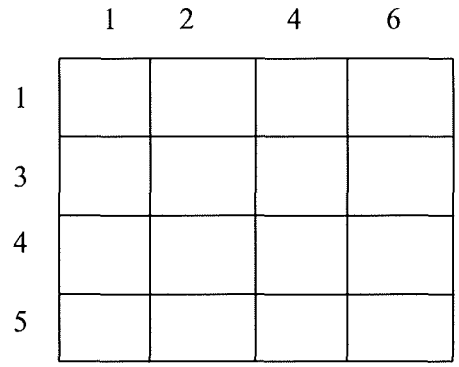


FIGURE 7.5A VECTORS APPLIED TO MATRIX

	1	2	4	6
1	0	1	3	5
3	2	1	1	3
4	3	2	0	2
5	4	3	1	1

FIGURE 7.5B CELLS COMPUTED AS VECTOR DIFFERENCE

Template vector $X = \{1, 3, 4, 5\} = \{X_j\}\ (0 < j <= 4)$ (7.1)

Sample vector $Y = \{1, 2, 4, 6\} = \{Y_i\}\ (0 < i <= 4)$ (7.2)

Each cell of the matrix is then computed by calculating the modulus of the difference between vector values corresponding to the column and row of the matrix, as shown in figure 7.5b.

Form matrix A as : $a_{i,j} = |y_i - x_j|$ (7.3)

The lowest values in the matrix lie closest to the leading diagonal, and for identical vectors, the diagonal would contain only zeros. A new matrix is now formed applying a local constraint from the top left cell of matrix A, to the bottom right cell. The constraint defines the set of processed elements in the matrix that must be considered in order to determine the new value for the next unprocessed element. To calculate a new value for location (i,j), the values in three locations are inspected : (i-1,j), (i, j-1), (i-1, j-1). The lowest value amongst these is added to the value already present in cell (i,j).

Form matrix B as : $B_{i,j} = A_{i,j} + \min (A_{i-1,j-1}, A_{i-1,j}, A_{i,j-1})$ (7.4)

Boundary conditions shown in figure 7.5c must be applied to start the process. The local constraint is then applied to each element from scanning from top left to bottom right.

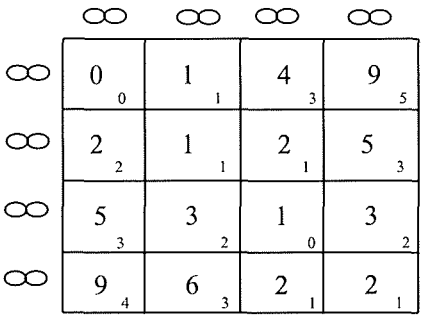


FIGURE 7.5C APPLYING LOCAL CONSTRAINT

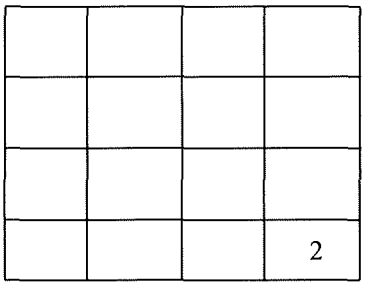


FIGURE 7.5D RESULTING COMPARISON

Finally the value in the bottom right cell of the matrix is used to determine the quality of match between the vectors. If the value lies below some pre-determined threshold level, then the sample vector is said to match the template.

7.2.2 The Single Layer Perceptron

An SLP neural network attempts to establish relationships between sets vector pairs. For a given set of input vectors **I** a set of output vectors **O** are produced. For a network that provides the required relationship (a trained network) the output vectors **O** are equivalent to a set of target vectors **T**. The structure of an SLP is illustrated in figure 7.6.

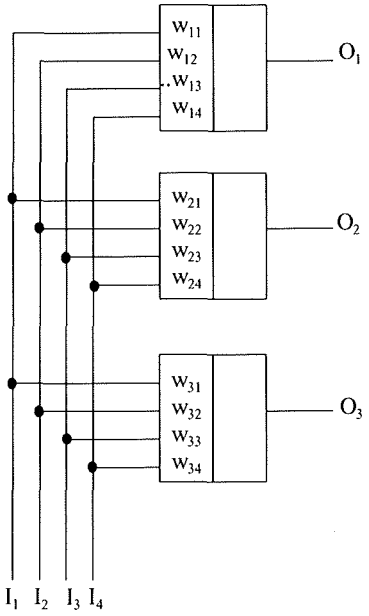


FIGURE 7.6 SINGLE LAYER PERCEPTRON WITH 4 INPUTS AND 3 OUTPUTS

Four input neurons and four output neurons exist, and each output is connected to every input. The strength of these connections are determined by a set of weights.

The value of each output is given as :

$$O_i = \sum_{j=1}^{N_i} w_{ij} \times \mathbf{I}_j \tag{7.5}$$

Where N_i is the number of neurons in the input layer, W_{ij} is the weight connecting the j *th* input to i *th* output, and if N_O is the number of outputs then 1 j N_O

The values of the weights are determined during the process of network training, typically employing the Delta-Rule. This uses the difference between the target and actual output to estimate the required change of weight value :

$$\delta w_{ij} = r\,(T_i - O_i)\,\mathbf{I}_j \tag{7.6}$$

Where δw_{ij} is the estimated change in value for weight w_{ij} , T_i and O_i are the target and actual values of output i, I_j is the j *th* input, and r is a learning rate coefficient. A new weight value is therefore computed as :

$$w_{ij}\ (n+1) = w_{ij}\ (n) + \delta w_{ij} \tag{7.7}$$

Network training is accomplished by continually applying each of the input vectors to the network, and adjusting the weight values to minimize the network error (difference between outputs and targets for all vector pairs). Typically, a 1-of-n coding scheme is used for the target vectors. If there are 8 pattern classes and 8 outputs, successful gesture classification results in one of the outputs having a value of one, with the remaining outputs at zero.

7.2.3 The DPA and SLP compared

A vocabulary of gestures was generated based on samples performed by 10 subjects (a group of undergraduate students were recruited as subjects). A vocabulary size of 8 provides an appropriate bandwidth for the Manipulator's interface system. The vectors generated from the gesture samples were 40 data points in length. As there were to be 8 classes of gesture, the SLP

had 40 inputs with 8 neurons in the output layer. Figure 7.7, shows four examples of a typical head gesture. As can be seen, variation between members of the same gesture class may be in terms of amplitude or phase, or the presence of tremor superimposed on the signal. A mathematical model was developed to simulate the variation of gestures within a class. This allowed for a large set of gestures to be generated with a controlled degree of degradation. The performance of each algorithm could then be plotted as a function of degradation.

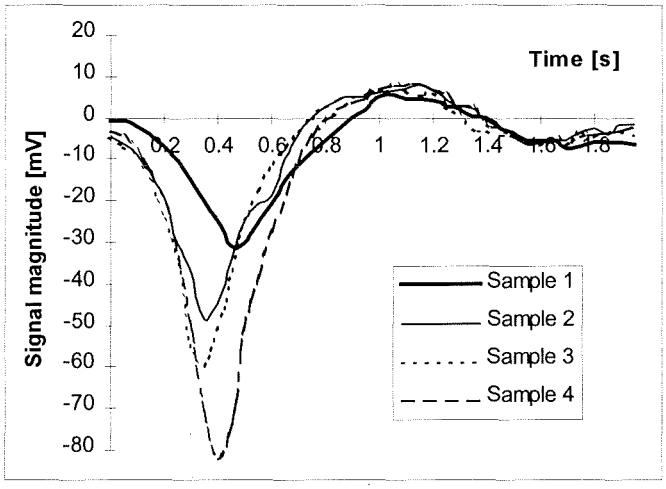


FIGURE 7.7 FOUR MEMBERS OF A TYPICAL GESTURE CLASS

The gestures were modeled as :

$$y = 30 \sin(\omega t - \pi) \quad : \quad 0 < \omega t < \pi \tag{7.8}$$

$$y = 5 \sin 2\omega t \quad : \quad \pi < \omega t < 2\pi \tag{7.9}$$

Allowing for degraded gestures to be generated using :

$$y = A \sin(\omega t - \pi)(1 + \phi) + C \sin k\omega t \quad : \quad 0 < \omega t < \pi \tag{7.10}$$

$$y = B \sin 2\omega t (1 + \phi) + C \sin k\omega t \quad : \quad \pi < \omega t < 2\pi \tag{7.11}$$

where A and B adjust signal magnitude, ϕ adjusts frequency, and C provides magnitude of tremor at a frequency proportional to k. Figure 7.8 shows the modeled gesture with varying degrees of magnitude, phase shift and tremor.

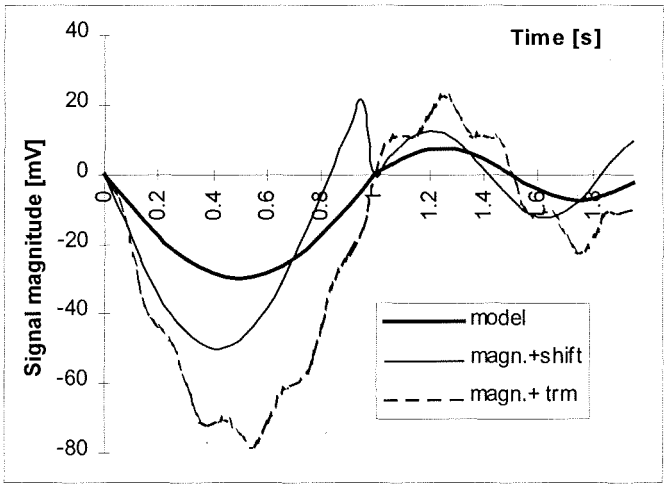


FIGURE 7.8 MODELED GESTURE WITH VARYING DEGREES OF DEGRADATION

Figure 7.9 below provides a graph of percentage variation in signal magnitude up to which gesture classification succeeded for varying degrees of shift in phase. Plots are provided for different degrees of tremor for the DPA, but are not shown for the SLP as the latter proved insensitive to tremor. Tests were performed up to magnitude variations of 200%, as this was representative of the worst gestures sampled. It can be seen that with no tremor or shift in phase the DPA failed at 18% magnitude variation. The SLP succeeded in classifying all gestures (up to 200% magnitude variation) for which the shift in phase was less than 16%.

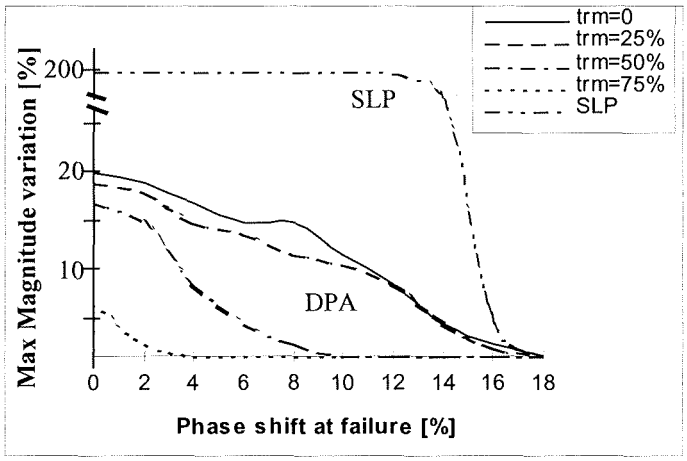


FIGURE 7.9 MAXIMUM VARIATION FOR SUCCESSFUL CLASSIFICATION

The SLP provided significantly greater classification performance than the DPA, when presented with typical levels of variation in user gestures. The linear separability constraint of the SLP did not restrict it successfully differentiating between user gestures. Hence, the results show that as

the two algorithms have comparable levels of computational complexity, the SLP would be considered the more appropriate of the two for the current application.

The following section describes the development of a more powerful pattern classification system using a multi-layered perceptron (MLP). This allowed the performance of the SLP to be compared with the MLP, and the advantage of the SLP's simplicity to be assessed.

7.2.4 The multi-layered perceptron

Figure 7.10 below shows the structure of an MLP that has two layers of neurons.

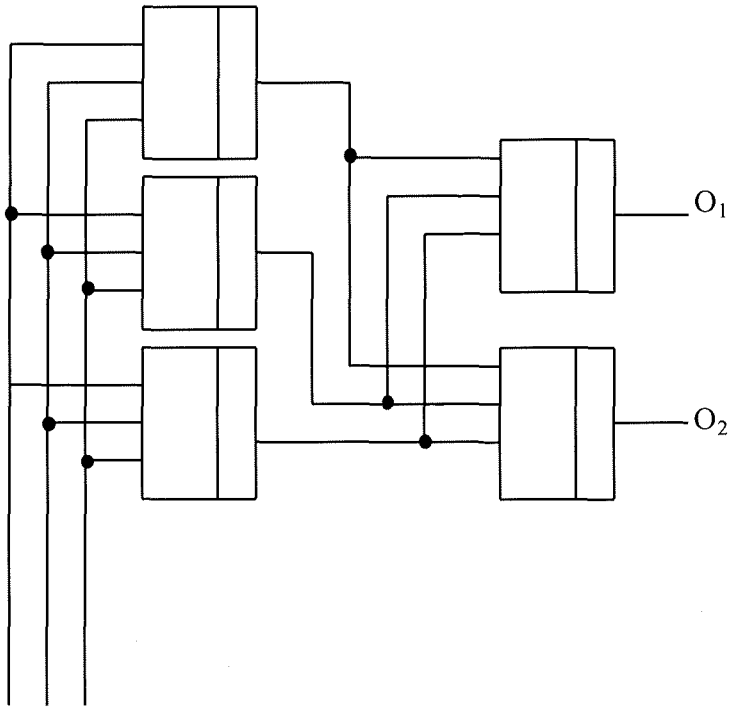


FIGURE 7.10 STRUCTURE OF A 2 LAYERED MLP

Each of the network inputs is connected to every neuron in the first layer via a set of weights. Similarly, each of the outputs of the first layer is connected to every input of the second layer. The actual number of neurons in each layer, and indeed the number of layers, is arbitrary, and is usually determined by comparing the performance of different network structures for a given application.

The output of each neuron is partially determined by the weighted sum of its inputs. This value is then applied to an activation (or squashing) function to determine the actual neuron output. The activation function is chosen to be easily differentiable, as the derivative is used in network training. The sigmoid function is typical used, given as :

$$O_i = (1 + e^{-S_i})^{-1} \tag{7.12}$$

Where O_i is the output of the i th neuron and S_i is the weighted sum value for i th neuron.
S_i is given as :

$$Si = \sum_{j=1}^{N_i} w_{ij} \times \mathbf{I}_j \tag{7.13}$$

Where N_i is the number of neurons in the input layer, W_{ij} is the weight connecting the j th input to i th output, and if N_O is the number of outputs then 1 j N_O

Network training is achieved with a set of input vectors representing the gestures to be learnt, and a set of target vectors representing the desired outputs. Wasserman (1989) describes the process of training as consisting of the following steps :

1. select an input-output pair of training vectors and apply to the input vector to the network;
2. calculate the output of the network;
3. calculate the difference (error) between the network output and the desired output;
4. adjust the weights of the network to minimize error; and,
5. repeat steps 1 to 4 with each training vector pair until the network error is acceptably low.

Steps 1 and 2 can be described in vector form. An input vector **I** is applied, and an output vector **O** is produced. The weights for each layer of the network may be considered as being a matrix, thus :

$$\mathbf{O} = F(\mathbf{IW}) \tag{7.14}$$

where **W** is the weight matrix and $F(x)$ represents the sigmoid activation function. Weight adjustments for neurons in the output layer is achieved using a modification of the delta-rule. The output of a neuron is subtracted from its target value to provide an error signal. This is then multiplied by the derivative of the activation function :

$$\text{If } F(x) = \frac{1}{1+e^{-x}} \quad \text{then } F'(x) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) \tag{7.15}$$

A delta value for a single neuron is therefore calculated using :

$$\delta = \text{Out } (1 - \text{Out}) (\text{Target} - \text{Out}) \tag{7.16}$$

To calculate the change required of a weight connecting neuron p in the first layer to neuron q in the second layer, the δ value for neuron q is multiplied by the output of neuron p and a training rate coefficient η typically between 0.01 and 1.0.

$$\Delta w_{pq} = \eta \, \delta_q \, Out_p \tag{7.17}$$

The weight change for connections to neurons in the first layer (or any hidden layers) may not be determined in the same way, as no target vectors exist for the layer. Instead, the δ value for neuron p in the first layer is calculated by propagating back the δ value calculated for the output layer, using :

$$\delta_p = Out_p (1 - Out_p) \left(\sum_{q=1}^{N_q} \delta_q w_{pq} \right). \tag{7.18}$$

Where N_q is the number of neurons in the output layer.

Training time and network stability may be improved by adding a term to weight adjustment proportional to the previous weight change (Rumelhart et. al. 1986).

This term is referred to as momentum, and is denoted as α. The weight change for connection between neuron p and neuron q is therefore given as :

$$\Delta w_{pq}(n+1) = \eta(\delta_q \, Out_p) + \alpha [\Delta w_{pq}(n)] \tag{7.19}$$

7.2.5 The MLP and SLP compared

The vocabulary of gestures described above was used to compare the classification performances of the MLP and SLP networks. The gestures were used to train the 2 networks, allowing for frequency of subsequent classifications to be determined. The structure of the SLP was as above. Similarly, the MLP had 40 inputs and 8 outputs. The number of neurons in the MLP's hidden layer was determined by recording classification performance as the number of neurons was varied from an initial value of 6 to a final value of 30. The performance improved as the layer size was increased to 18, and then leveled off. The size of the hidden layer was therefore set at 18.

Once trained, both networks proved capable of successfully classifying all eight gestures. Here, a miss-classification is defined as the wrong output neuron having the highest value for a given input. Using this form of calculation, initial tests produced classification rates of 84% for the SLP and 91% for the MLP (an average from three subjects attempting to perform a total of 90 gestures). However, it is useful to estimate the certainty with which a classification has been made, as it is this, combined with the set threshold level, that would determine whether the network output should be interpreted as a positive recognition of one of the defined gestures. This was approximated by expressing the difference between the highest output and the second highest as a percentage :

$$C = \frac{O_p - O_q}{O_p} \times 100 \tag{7.20}$$

where O_p is the output of the winning neuron, O_q is the second highest output and C is the level of certainty.

The results of the classifications for both networks would then be 69% (SLP) and 82% (MLP). As would be expected, better performance was achieved with the MLP. The cost of this improved performance is the training time required of the MLP. However, with the ever increasing processing power of personal computers, training times are decreasing in significance. With the moderate size of the MLP described above, and the fairly small set of sample gestures, network training typically lasted approximately 6 minutes with a processor running at 100 MHz.

7.2.6 The Radial Basis Function

As described by Hush and Horne (1993), a Radial Basis Function (RBF) network is a 2 layer network, whose output neurons form a linear combination of the basis (kernel) functions computed by the neurons in the input layer. Bishop (1995), argues that the classification performance of the RBF is comparable to an MLP employing the backpropagation algorithm. However, training times are typically lower than the MLP, as the learning processes is broken into 2 separate stages, the algorithms for which are fairly efficient. An RBF was therefore implemented as described below, to allow for its comparison with the MLP.

The basis functions used for the input layer produce a localized response to the input vectors, i.e. each neuron provides a significant non-zero response if the input vector falls within a small localized region of the input space. The basis function typically employed is a Gaussian kernel of the form :

$$u_j = \exp\left[-\frac{(\mathbf{x}-\mathbf{c}_j)^T(\mathbf{x}-\mathbf{c}_j)}{2\sigma_j^2}\right] \qquad j = 1,2,.....,N_1 \qquad (7.21)$$

where u_j is the output of the j^{th} node of the first layer, $\mathbf{x}$ is the input pattern and $\mathbf{c}_j$ is the weight pattern of the j^{th} node of the first layer, σ_j^2 is the normalization parameter for the j^{th} node, and N_1 is the number of nodes in the first layer.

The output layer node equations are given by :

$$y_k = \mathbf{w}_k^T \mathbf{u} \qquad k = 1,2,.....N_2 \tag{7.22}$$

Where y_k is the output of the k^{th} node, $\mathbf{w}_k^T$ is the transpose of the weight vector for this node, $\mathbf{u}$ is the vector of outputs from the first layer. N_2 is the number of nodes in the output layer.

Training the RBF network was achieved in two stages : an unsupervised training process for the first layer, followed by a supervised training process for the output layer.

procedure K Means
Initialize the cluster centers (weight vectors of the first layer) $\mathbf{c}_j$ $j = 1,2,....,N_1$
repeat
/* group all patterns with the closest cluster center */
for all $\mathbf{x}_i$ **do**
Assign $\mathbf{x}_i$ to Θ_{j*}, where $\mathbf{c}_{j*} = \min_j \|\mathbf{x}_i - \mathbf{c}_j\|$
endloop
/* Compute sample means */
for all $\mathbf{c}_j$ **do**

$$\mathbf{c}_j = \frac{1}{m_j} \sum_{x_i \in j} x_i$$

endloop
until there is no change in cluster assignments from one iteration to the next.
end.

FIGURE 7.11 K-MEANS CLUSTERING ALGORITHM

A K-Means clustering algorithm was used for the unsupervised training phase, this is outlined in figure 7.11 (adapted from Hush and Horne, 1993), where, m_j is the number of input patterns associated with cluster $\mathbf{c}_j$. Grouping input patterns with cluster centers forms a set of training patterns Θ_j . This is done on the basis of the minimum Euclidean distance between a given input pattern and the cluster centers. The normalization parameters σ_j^2, were obtained once the clustering algorithm was complete. These represent a spread of the data associated with each node, and were calculated as the average distance between the cluster centers and the training patterns, given by :

$$\sigma_j^2 = \frac{1}{M_j} \sum_{\mathbf{x} \in \Theta_j} (\mathbf{x} - \mathbf{c}_j)^T (\mathbf{x} - \mathbf{c}_j) \tag{7.23}$$

where Θ_j is the set of training patterns grouped with cluster c_j, and M_j is the number of patterns in Θ_j.

Learning in the output layer was performed after the parameters of the basis functions had been determined, this was accomplished using a Least Means Squared algorithm, (LMS) similar to that employed by the SLP mentioned above. The LMS is summarized in figure 7.12.

procedure LMS
  Initialize weights w_j to small random values j = 1,2,.....,N_2
  repeat
    Choose next training pair (**u,d**)
    /* Compute Outputs */
    for all *j* **do**
      $y_j = w_j u$
    endloop
    /* Compute Errors */
    for all *j* **do**
      $e_j = y_j - d_j$
    endloop
    /* Update Weights */
    for all *j* **do**
      $w_j(k+1) = w_j(k) - \mu\ e_j u$
    endloop
  until acceptable error
end

FIGURE 7.12 LEAST MEAN SQUARE ALGORITHM

The vectors constituting a training pair are the output of the first layer **u**, and the desired output of the second layer **d**. The actual output of the second layer is represented by **y,** with **e** being the error or the difference between actual and desired outputs. The weight vector for the second layer is **w**, and μ represents the network learning rate.

7.2.7 The MLP and RBF compared

An RBF was implemented with the structure employed for the MLP described above, i.e. 40 inputs, 18 neurons in the first layer, and 8 neurons in the output layer. The gestures generated for section 7.2.5 were used to train the RBF, allowing for the frequency of classification of

subsequently formed gestures to be recorded. As was anticipated, the training times for the RBF were lower than the MLP at approximately 2 minutes. However, the classification performance was far poorer at 52%. This may be explained by the fact that the training set used was small, with only 5 samples for each gesture to be recognised. The kernels computed by the neurons in the input layer are therefore less representative of a gesture class than would be achieved by a far richer training set. Bishop (1995) points out that the relative performance of the RBF is far greater when a rich training set is available. However, for the current application, any benefits gained over the MLP in terms of reduced training times would be lost due to the time that would be required to capture a larger number of training gestures from a user.

7.3 Configuring the Tilt-sensor for use with the UIS

The tests described above indicate that an SLP provides significantly better classification performance than a DPA, with a similar level of computational complexity. The MLP improved on the performance of the SLP, incurring training times that appear to be of a moderate level, and capable of being incorporated into the process of configuring the UIS. The user acceptance of the device and the classification system's performance would need to be determined by user testing. However, the initial results indicated that the sensor would not be appropriate for use in a direct-menu selection system. The slow time response of the sensors resulted in gesture lengths of up to 2 seconds. This length of time was required to ensure that each gesture in a set of 8 was adequately different from the remaining gestures. The result of this would be that a system employing direct-menu selection would provide slower user interaction than a scanning system, and since the cognitive demands of direct-menu selection are greater, the scanning system would appear to be the preferable style of interaction if tilt sensors are employed.

A scanning system requires a minimum of one gesture for operation, and can therefore be operated with the tilt sensor acting as a switch - tending to suggest that a pattern classification algorithm is not required. However, the use of such an algorithm has potential for recognizing involuntary movement, and can allow for added functionality. For example, one gesture may be used to select the current option, another to return to the previous stage of interaction, another to cancel dialogue and stop any movement of the arm. Consequently, increasing the bandwidth of an input device being used with a scanning system, reduces the number of options that the

scanning system needs to manage, and therefore can allow for more rapid user interaction. This latter approach was adopted for the development of a scanning system, the initial trials of which are reported in Chapter 8.

7.4 Gesture Encoding with a trackball

Trackballs have been used successfully as input devices for rehabilitation systems for those who have partial hand movement (for example, Verburg et. al. 1995). A program was therefore developed to allow the application of an artificial neural network to the encoding of hand gestures issued by a trackball. As shown below, this form of input does not suffer from the poor time response exhibited by the tilt sensors. This would allow for larger vocabularies of gestures to be more easily generated, and hence direct menu selection to be a feasible form of interaction.

7.4.1 Outline of a Gesture-Recognition Windows Application

A Windows application was developed to allow for the encoding of gestures in 2 dimensions. Windows applications generate 'mouse move' messages when an input device is being moved, these contain the x and y coordinates of the current position of the input device. A function was written to store a set of x and y coordinates as a vector, that act as an input to an MLP neural network. A description of simplified versions of the application's principal functions is provided below, for full code listings see Appendix H.

The EvLButtonDown function in figure 7.13 is a member function of the application's main Window class. The function is called in response to the generation of a Windows message indicating that the trackball has been depressed.

```
void TGestWindow::EvLButtonDown(uint, TPoint& point)
{
        START_REC = TRUE;
        StartX = point.x;
        StartY = point.y;
}
```

FIGURE 7.13 FUNCTION TO INITIATE GESTURE RECORDING

This is used to initiate the recording of a gesture. The function is passed a reference to a point structure containing the current coordinates of the input device. Two variables are initialised to record the position of the beginning of the gesture, and a flag is set to true to indicate to associated functions that gesture recording has been initiated.

```
void TGestWindow::EvMouseMove(uint, TPoint& point)
{
        if(START_REC)
        {
                // trace element counts coordinates already recorded
                //  restricted by size of network
                if(TraceElement<(InputNodes))
                {
                        x = point.x;
                        y = point.y;

                        x -= StartX;        //        Adjust relative to start
                        y -= StartY;

                //        Add coords to gesture structure
                        Gesture.x[TraceElement] = x;
                        Gesture.y[TraceElement] = y;
                        TraceElement++;
                }
                else   // End of Template Record
                {
                        START_REC = FALSE;
                        Result = Classify(Gesture);
                }
        }
}
```

FIGURE 7.14 FUNCTION TO RECORD GESTURE

The EvMouseMove function is called in response to the generation of a Windows message indicating that the trackball is being moved. If the START_REC flag has been set the coordinates passed as function arguments are added to a gesture structure which contains an array of x and y coordinates. This is repeated a number of times controlled by the TraceElement variable, and dependent on the size of the network. The Classify function is then passed the gesture structure, and returns an integer corresponding to the winning output node (the function provided in Appendix H has as a function argument a reference to an array of floats containing

all network outputs). As shown below, the classify function creates a vector **X** for input to the network from the gesture structure. Two functions are then called, the first calculating the output of the input (hidden) layer, and the second computing the output neuron values. The output neurons are then each inspected in turn, to determine whether their values are greater than any previously inspected. A variable Winning node is then assigned a value dependent on which neuron has the largest value.

```
int Classify(TGesture &Gesture)
{
        float MaxOutVal  = -9999; // largest output so far
        int WinningNode = -1;
        int g = 0;

        // Create X from gesture
        for(int k =0; k<VectorLength/2; k++)
                X[k] = (float )Gesture.x[k];

        for(k = VectorLength/2; k<K; k++)
        {
                X[k] = (float )Gesture.y[g];
                g++;
        }

        ComputeHiddenOut();
        ComputeNetOut();

        // find maximum output
        for(int i = 0; i<I; i++)
        {
                NetOut[i] = Y[i];
                if(Y[i] > MaxOutVal)
                {
                        MaxOutVal = Y[i];
                        WinningNode = i;
                }
        }
        return WinningNode + 1;
}
```

FIGURE 7.15 FUNCTION TO CLASSIFY GESTURE

```
void ComputeHiddenOut()
{
        // J = number of neurons in 1st layer
        for(j=0; j<J; j++)
        {
                U[j] = VectorMult(&WH[j][0], X);
                Temp = expl((long double)(-1*U[j]));
                U[j] = 1/(1+ (float)Temp);
        }
}

U = output of hidden layer
X = input to hidden layer
WH = weight matrix for hidden layer
```

FIGURE 7.16 COMPUTES OUTPUT OF HIDDEN LAYER

```
void ComputeNetOut()
{
        for(i=0; i<I; i++)
        {
                Y[i] = VectorMult(&WO[i][0], U);
                Y[i] = 1/(1+(exp(-1*Y[i])));
        }
}

Y = network output
U = Hidden layer output
WO = weight matrix for output layer
```

FIGURE 7.17 COMPUTES NETWORK OUTPUT

7.4.2 Trackball gesture-recognition : Initial Results

Initial tests of the performance of the MLP were undertaken by the author, classifying sampled gestures against a training set containing eight gesture classes. An application (shown in appendix H) was written, that requests the user to perform one of eight the gestures. The gesture number is selected at random, a classification attempted and the result recorded, this is repeated 40 times. The initial results were encouraging, with recognition rates between 95% and 100% regularly achievable. Additionally, unlike the gestures encoded with tilt sensors, the gestures can be performed in under 1 s.

As mentioned in section 7.18, the implications of gesture-recognition performance on the usability of an interface system would need to be determined through user testing. Chapter 9 describes an experiment designed to determine usability levels offered by an interface employing gesture recognition. Subjects include able-bodied and physically disabled people, allowing for the implications of the diversity of controlling ability within the subject group to addressed.

7.5 Conclusions

This chapter described how tilt sensors may be used as a form of input for the Manipulator's user interface, allowing for greater signal bandwidth than provided by a simple switch system. The slow time response of the sensors was found to limit the signal bandwidth for practical purposes, suggesting that the sensors would be more appropriately applied to a scanning system or keyboard emulation, rather than direct menu selection.

The SLP neural network was found to have significantly greater gesture classification performance than a DPA, at a similar level of computational complexity. However, an MLP improved on this performance with a moderate network size, and hence fairly low training times were incurred. An RBF was unable to offer similar performance levels, as the training set consisted of a small set of training samples which seemed to offer particularly poor training conditions for the RBF. The MLP algorithm was therefore adopted for the classification systems.

A Windows application was developed to allow gestures to be encoded with a trackball. An MLP was shown to be capable of achieving recognition rates of between 95% and 100% for a user without a physical impairment, repeatedly issuing a random selection of one of eight gestures. However, the initial tests performed did not allow for any general conclusions as to the suitability of either the tilt sensors or the trackball as a form of user input. Chapter 8 presents some preliminary findings from an initial user evaluation of the Manipulator system, where gesture recognition is one of the modes of input employed. Chapter 9 discusses an approach that would allow the controlling ability of specific users and input device characteristics to be assessed, allowing predictions of the relative usability of different interface configurations to be made.

Section IV

Evaluation

Chapter 8

System Evaluation

This chapter summarizes two evaluations of the Middlesex Manipulator that were undertaken as part of an iterative design-cycle. The first, a heuristic evaluation of the user interface system, was undertaken by a group of computing science students, to identify potential problems relating to user interaction. This allowed for an assessment of the appropriateness of heuristic evaluation to user interface design for assistive technology. The chapter shows that the heuristic evaluation provided an appropriate framework for the identification of a number of usability issues, some of which were then verified empirically.

An individual with spinal-cord injury was invited to undertake an evaluation of the manipulator and input devices while performing tasks from the prioritized task list. This chapter provides an analysis of the resulting usage data, and a discussion of the user's subjective feedback. This is compared with the design requirements provided in Chapter 3, and the general design criteria outlined in Chapter 2. The chapter concludes that two of the design criteria are not met, the consequences of which are discussed further in Chapter 10.

8.1 A Heuristic Evaluation

This section presents the results of a Heuristic Evaluation of an initial configuration of the Manipulator's user interface system. Two objectives were central to the work reported here. Firstly, to assess the efficacy of Heuristic Evaluation within the current design scenario, and secondly, to highlight potential improvements in system design. The evaluation was undertaken by the author and four undergraduate students undertaking a computing science module at Middlesex University (Human-Computer Interaction Evaluation Methodologies, COM3240 1996/1997).

8.1.1 Method

The students were provided with information regarding the background and objectives of the project. They were each then provided with a copy of the interface software. A configuration was provided in a form to match that used during the user evaluation as described in section 8.2 below. The commands presented by the interface constitute a subset of the command language defined in appendix E. Commands presented by the menu system are illustrated by the examples provided below. Figure 8.1 illustrates the sequence of menu selections required to switch the system power on. Initially the 'Power' command is selected from the top-level menu, activating a sub-menu containing three items. The 'On' command is then selected, returning the system to the top-level. Similar sequences of interaction are illustrated below for speed selection, moving to a pre-taught position, and moving a joint.

Move	Goto	Speed	HereIs	**Power**
On	Off	End		
Move	Goto	Speed	HereIs	Power

FIGURE 8.1 INTERFACE MENU SEQUENCE - SELECTING 'POWER ON'

Move	Goto	**Speed**	HereIs	Power
Slow	**Med**	Fast		
Move	Goto	Speed	HereIs	Power

FIGURE 8.2 INTERFACE MENU SEQUENCE - SELECTING 'SPEED MEDIUM'

Move	**Goto**	Speed	Herels	Power
Front	**Side**	Home	End	
One	Two	Three	Four	
Stop				
Move	Goto	Speed	Herels	Power

FIGURE 8.3 INTERFACE MENU SEQUENCE - MOVING TO A PRE-TAUGHT POSITION

Move	Goto	Speed	Herels	Power		
Base	Arm	**Shoulder**	Elbow	Hand	Wrist	End
In	**Out**	Wrong	End			
Stop						
Move	Goto	Speed	Herels	Power		

FIGURE 8.4 INTERFACE MENU SEQUENCE - SELECTING JOINT MOVEMENT

Each evaluator was provided with a description of a typical user task, employing each of the control modes presented by the interface.

Pick and Place task description

Switch on the power to the system, and set the speed to medium. Move to a pre-taught position near the target object. Set the speed to slow, and adjust the elbow and hand joints to approach the target. Close the gripper, and at medium speed move to a pre-taught position near the destination. Adjust the base and shoulder joints, then release the object.

Evaluators independently simulated undertaking the user task by walking through appropriate command sequences with the interface. Aspects of user interaction were recorded that could be deemed as conflicting with the usability heuristics outlined in Chapter 4. A meeting was then convened to allow the separate findings to be pooled and discussed. Where appropriate, a possible solution was suggested, and an attempt was made to estimate both the severity of the problem, and how difficult the problem would be to address. This allowed for decisions as to whether design modifications should be made, and if so, at which stage of the project's design cycle.

As discussed by Neilson (1994), problem severity may be estimated along two dimensions : impact and frequency. The approach taken here was to construct a Likert scale corresponding to

each dimension, allowing for problem severity to be estimated using the product of the average scores given by evaluators.

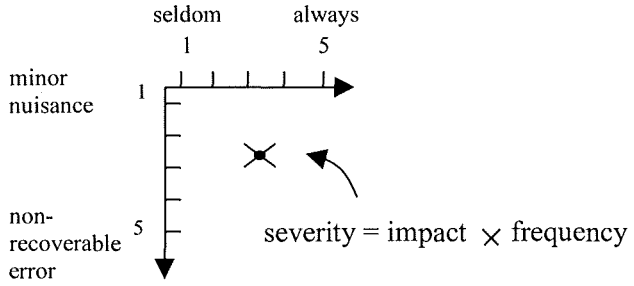


FIGURE 8.5 ESTIMATING PROBLEM SEVERITY

Estimating the difficulty that may be involved in providing a solution, was done by discussing the amount and type of work involved, ranging from code-editing, and code-development, to researching a novel solution. An estimation was arrived at collectively in man-hours.

8.1.2 Results

Each usability issue identified includes a reference to the relevant heuristic, the estimation of problem severity (PS) and solution cost (SC) in man-hours.

1. *Simple and natural dialogue.* PS = 12, SC = 4. The Stop command returns the user to the top-level menu. However, the user task as modeled suggests it may be more appropriate to be returned to joint selection, allowing a number of joints to be more easily moved in quick succession. A possible solution would be:

Move	Goto	Speed	Herels	Power		
Base	Arm	**Shoulder**	Elbow	Hand	Wrist	End
In	**Out**	Wrong	End			
Stop						
Base	Arm	Shoulder **Elbow**		Hand	Wrist	End
In	Out	Wrong	End			
Stop						
Base	Arm	Shoulder Elbow		Hand	Wrist	**End**
Move	Goto	Speed	Herels	Power		

FIGURE 8.6 MOVING SHOULDER AND ELBOW JOINTS

2. *Simple and natural dialogue.* PS = 4, SC = 4. The 'Wrong' option provides an undo facility whereby the user may return to the previous level of interaction. However, the interface repeats the same set of options, even though it may be inferred that the item previously selected is not required. It would be useful to remove the item, particularly if a recognition system is frequently confusing two commands. An alternative approach would be

Move	Goto	Speed	Herels		Power		
Base	Arm	**Shoulder**	Elbow		Hand	Wrist	End
In	Out	**Wrong**	End				
Base	Arm	**Elbow**	Hand	Wrist		End	
In	Out	Wrong	End				
Stop							
Move	Goto	Speed	Herels		Power		

FIGURE 8.7 MOVING ELBOW, WITH INCORRECT SELECTION OF SHOULDER

3. *Prevent errors.* PS = 5, SC = 1. Currently no confirm is required before movement of the manipulator commences. However, the trade-off would be a larger number of commands being issued for each move. The current system does not include a confirm command. However, it would be appropriate to include this as an option when configuring systems.

Move	Goto	Speed	Herels		Power		
Base	Arm	**Shoulder**	Elbow		Hand	Wrist	End
In	Out	Wrong	End				
Go	End						
Stop							
Move	Goto	Speed	Herels		Power		

FIGURE 8.8 MOVE SELECTION WITH 'GO' TO CONFIRM

4. (Provide short-cuts). PS = 9, SC = 8. For experienced users, it may be appropriate to allow movement of a second joint to be initiated whilst a first is already in motion, or to allow the speed of movement to be changed whilst a joint is already in motion. This could be achieved by adding an 'AND' option to the system as shown below:

Move	Goto	Speed	HereIs	Power		
Base	Arm	**Shoulder**	Elbow	Hand	Wrist	End
In	**Out**	Wrong	End			
Stop	**And**	Speed				
Base	Arm	Shoulder Elbow		**Hand**	Wrist	End
In	**Out**	Wrong	End			
Stop	And	**Speed**				
Slow	Med	Fast				
Stop	And	Speed				
Move	Goto	Speed	HereIs	Power		

FIGURE 8.9 INCLUSION OF 'AND' OPTION

This would be inappropriate for novice users, or users unlikely to develop a high degree of controlling skill. However, greater controlling flexibility and efficiency would be provided for skilled users. The option was therefore built into the interface for inclusion in the later stages of user trials.

5. *Speak the user's language.* PS = 3, SC = 4. The system presents three speed levels : slow, medium and fast. The actual joint speeds corresponding to these levels may be configured for individual systems. This brings into question whether the word 'fast' would always be appropriate. Enumerated speed settings would be an alternative, provided it was made clear to the user which number corresponded to the slowest option. Enumeration would also allow for more natural inclusion of a larger number of speed settings where appropriate.

6. *Simple and Natural Dialogue.* PS = 5, SC = 1. An 'End' option exists as part of the 'Power' sub-menu, however the option is redundant as either of the alternatives returns the user to the top level.

7. *Provide Feedback.* PS = 20, SC = 30. The feedback presented by the interface includes the list of commands that may be selected, and the previously selected command. Consequently, the appearance of the screen would be the same at certain points during the 'Teach' and 'HereIs' modes. This could be resolved by providing a display of interaction history, though this may clutter a relatively small LCD display. Alternatively, a symbol or icon could be developed for each mode of control, and made to appear in a corner of the display whilst the mode is active.

8. *Provide Feedback.* PS = 10 for novice users, SC = 8. When the manipulator is in motion, the interface does not explicitly state that the arm is moving, though movement can be inferred from the currently active menu options. As with mode status discussed above, an icon could be developed for this purpose.

9. *Provide Feedback.* PS = 8, SC = 8. When an axis limit is encountered, no feedback is provided to indicate that this has happened, and which of the manipulator's limits has been reached.

10. *Simple and Natural dialogue.* PS = 6, SC = 8. When an axis limit is encountered, the menu option that would allow the user to attempt to move the manipulator further beyond its limit is still present.

11. *Provide feedback.* PS = 20. SC = 4. When selecting a pre-taught position, or recording a position, the response time of the system is slightly slower than when issuing any other command (due to the fact that the positional settings for each of the axes is transmitted). The system should warn of response time by letting the user know that it is busy, thereby preventing the user from attempting to re-select the option.

12. *Prevent Errors.* PS = 25, SC = 4. When recording a position, the user may inadvertently overwrite existing data. A confirm option should be used to reduce the probability of this occurring.

Move	Goto	Speed	**HereIs**	Power
Front	**Side**	Home	End	
One	Two	Three	Four	
Yes	No			
Move	Goto	Speed	HereIs	Power

FIGURE 8.10 TEACH POSITION SIDE-ONE, WITH CONFIRM

13. *Minimise user memory load.* PS = 20, SC = 8. The interface provides no information within the 'HereIs' mode regarding which of the pre-taught positions remain unallocated. This could be achieved by greying menu-items, or varying text size.

14. *Prevent Errors.* PS = 25, SC = 8. The interface provides the option of moving to a pre-taught position, even if the position is currently un-allocated.

15. *Prevent Errors.* PS = 3, SC = 4. The use of the term 'HereIs' for the teach mode may be problematic. If a voice recognition system is employed, the user may be inclined to pronounce the two syllables separately, presenting problems for the recognition system. The mode name could be changed to 'Here'.

16. *Simple and Natural dialogue.* PS = 5, SC = 4. The option 'HereIs' appears before the 'Power' option, but is likely to be used less frequently.

17. *Speak the user's language & minimize user's memory load.* PS = 15, SC = 30. The system provides no opportunity for the user to provide names for the pre-taught positions.

18. *Provide feedback.* PS = 15, SC = 100. The interface provides no feedback as to how well the commands are being classified (relevant for voice and gesture recognition). A graphical device could be employed for this purpose.

19. *Prevent errors.* PS = 4, SC = 20. The system could request a confirm if the user attempts to set the speed to fast whilst the manipulator end-effector is in or near a zone normally occupied by the user.

Move	Goto	**Speed**	HereIs	Power
Slow	Med	**Fast**		
Yes	No			
Move	Goto	Speed	HereIs	Power

FIGURE 8.11 SET SPEED TO FAST WITH CONFIRM

20. *Provide good error messages.* PS = 5, SC = 4. An error message would occur during the evaluation, as the interface was not connected to the motor control system. The message provided was simply ' Communication error' which provides no information to the user (or personnel configuring the system) as to the nature of the problem, or how to solve the problem.

8.1.3 Conclusions

Undertaking an heuristic evaluation of the user interface provided the opportunity for a group of computer scientists to individually and collectively critique a system prototype. By applying accepted usability heuristics, a considerable number of design issues were highlighted (20), in spite of the limited experience of the evaluators. The validity of a number of the usability issues raised, requires to be verified through further analysis or user-testing. However, by highlighting the issues, they necessarily form a part of the design process. Design decisions resulting from the evaluation are aided by the inclusion of rough estimates of the problem severity and solution cost.

Following the evaluation, a number of modifications were made to the interface, corresponding to issues where the argument for design change is convincing and the solution cost is low. These correspond to the issues numbered : 6, 11, 12, 14, 15, 16 and 20.

In order to assess the impact of the design issues relating to user efficiency, two versions of a new prototype were developed. The first contained only those modifications mentioned above, and a second, which included modifications corresponding to design issues 1, 2 and 4. This allowed the user-testing as described below, to include an assessment of any increased efficiency in manipulator control resulting from the suggested design changes.

8.2 User evaluation overview

8.2.1 Background

This section summarizes the results of a user evaluation undertaken by an individual with spinal-cord injury within a laboratory environment. At the time of the evaluation the manipulator system was at the prototype stage, as opposed to the product stage. A number of required design modifications had already been highlighted, some of which are described in section 8.1, and the remainder are included below. Additionally, the manipulator employed a temporary single-axis gripper, in place of the incomplete three-axis end-effector. Consequently, the user evaluation was not designed as a product acceptance exercise, but as part of the design process. An individual (referred to below as the evaluator), was identified with a C4 incomplete spinal-cord injury. The evaluator had wide exposure to disability issues through employment as a counselor, and an

appreciation of technical design issues through pre-accident employment and education. The objectives of the evaluation were to:

- gain subjective feedback from a potential end-user regarding the appearance and performance of the system;
- determine whether user tasks from the prioritized task list could be successfully undertaken;
- quantify usability by examining the nature and frequency of user errors, learning times, and task completion times.
- determine the validity of the usability problems predicted by the heuristic evaluation described in section 8.1.

8.2.2 Method

The user evaluation consisted of the following four stages. Each stage was carried out on separate occasions, with separating intervals of up to one month. This allowed feedback to be incorporated into design modifications during the evaluation period.

Stage 1 - Familiarization.
The evaluator was provided with background information regarding the Middlesex manipulator, outlining the project's objectives and status. A description of the field of Rehabilitation Robotics was also provided, including videos of the MANUS and HANDY-1 systems. A demonstration of the interface system was given, during which the evaluator navigated the menu system using a trackball as an input device. The manipulator system was then connected to the interface, allowing the user to experiment with the system's basic operation (joint and pre-taught position modes). The voice and gesture recognition systems were introduced, and user data was recorded, allowing for the recognition systems to be configured for use during subsequent stages.

Stage 2 - The Feeding task
The feeding task was selected from the prioritised task list for the next stage of evaluation as the complexity of control demanded of the user is fairly low. A semi-structured environment was created, containing pre-taught positions around the food and user areas. The evaluator was required to retrieve food by accessing the pre-taught positions, and if necessary, utilizing joint

control[1]. The task was demonstrated using the voice, trackball and head-gesture input devices. The voice and trackball employed direct menu selection, whereas head-gestures were used with a scanning system. A video recording was made of the evaluator undertaking tasks with each of these input modes, providing comments on performance and usability as appropriate.

Stage 3 - Drinking/Pick & Place tasks

The next stage of the evaluation combined the slightly more complex Drinking and Pick & Place tasks. The user was required to:

Pick up a plastic straw, and place the straw in a cup. Turn a tap on and off, filling the cup. Pick up the cup, and carry it to an accessible position. Finally, replace the cup on the adjacent surface.

The task objects existed in an environment modified to allow ease of manipulation, however pre-taught positions were not provided. A video recording of the session was made for data analysis.

Stage 4 - Interview

Although feedback from the evaluator had been elicited throughout the evaluation, the final stage used a semi-structured interview to allow a more formal recording of user impressions. Questionnaires are of limited value for single-user studies, however, the approach provided structure to the interview, ensuring that issues addressed by similar studies were included. The approach would also facilitate the development of an appropriate interview or questionnaire format for use in subsequent product-acceptance evaluations.

8.3 Usage data summary and analysis

This section provides an analysis and summary of usage data collected from stages 2 and 3 of the user evaluation. A discussion of the subjective feedback is provided in section 8.2.5. Footage of sections of the evaluation is provided on a video which accompanies the thesis. Appendix J

[1]As initial evaluations employed a 6-axis manipulator, orientation of the end-effector for cartesian mode was not possible. Hence, evaluation of this mode of control is not included.

provides a listing of the prominent issues that may be observed in the video, along with timing information that allows the relevant events to be easily located.

8.3.1 Task duration

As described in Chapter 2, the subjective feedback from users of assistive technology ultimately determines the usability of that technology. However objective measurements such as task completion times, or times required to complete components of a task, are useful when comparing different systems, or highlighting problematic aspects of a particular system. The objectives of the work reported in this section of the thesis was to :

- estimate the times associated with the operations that constitute the feeding and drinking tasks;
- quantify the effects that design changes would have on task completion times; and,
- compare task completion times with those achieved by the HANDY 1 and MANUS systems.

The overall task completion time for the feeding task undertaken as stage 2 of the evaluation is difficult to quantify, as there is no clear end-point for the task (the plate was never completely cleared). Additionally, the time required to complete a feeding task would be strongly dependent upon the type and amount of food used, food preparation, whether an appropriately adapted plate and spoon were available, and the positioning of the plate with respect to the user. Addressing these factors would require a design exercise, which was not undertaken for the purposes of the initial evaluation. Consequently, the analysis focused on the time required to retrieve a single spoonful of food from the plate.

During the feeding task, the plate was placed approximately 1 m away from the evaluator, and the manipulator's speed was set at medium. After an initial familiarization period of approximately half an hour, the time required to retrieve a spoon of food by the evaluator was measured as 81 seconds (taken as an average of 10 runs). For comparison, the typical time required to retrieve food by the HANDY 1 feeding aid is around 8 seconds (measured from a promotional video : Handy 1 an aid to feeding, Rehab Robotics). Although there are a number of differences between the tasks undertaken by the two systems, an analysis of the evaluation video highlights a number of factors that contribute to the slower performance of the Middlesex manipulator. Firstly, the HANDY 1 is designed to undertake feeding by performing a pre-programmed task or routine.

Consequently, considerably fewer commands are required to be issued by the user than is the case with alternative modes of control. The Middlesex manipulator allows for pre-programmed routines to be executed, but for the purpose of the current evaluation, this feature was not exploited[2].

Figure 8.12 below, shows how the feeding task may be decomposed into four components : approaching the plate, scooping food, approaching the user, and stationary (waiting for next command to be completed). The results of the Heuristic evaluation discussed in section 8.1.2, suggested a number of improvements to the interface, including the use of an 'AND' option that would allow a command to be issued before a previously issued command was completed.

Task component	Duration
Approach plate	14 s
Scoop	34 s
Approach user	15 s
Stationary	18 s
Total	81 s

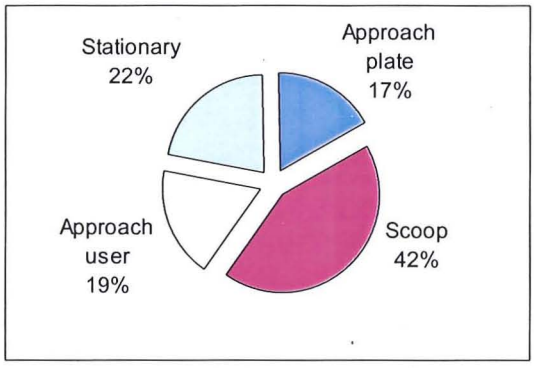


FIGURE 8.12 FEEDING TASK COMPONENTS

Within the feeding task, this allowed the evaluator to begin a dialogue to move to a pre-taught position before the previously selected position had been reached. This feature was implemented towards the end of stage 2 of the evaluation, and resulted in the task component times listed in figure 8.13 below.

[2] Control using pre-taught positions was employed, allowing more to be ascertained form the evaluation in terms of user interaction.

Task component	Duration
Approach plate	14 s
Scoop	31 s
Approach user	15 s
Stationary	5 s
Total	65 s

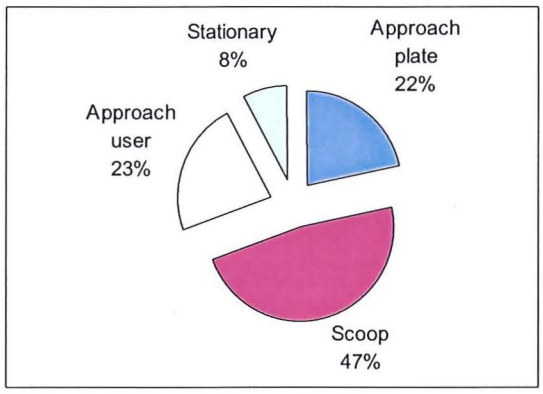


FIGURE 8.13 FEEDING TASK COMPONENTS (WITH MODIFIED USER INTERFACE)

Figure 8.13 shows the time required to retrieve food reduced from 81 s to 65 s, with the time that the manipulator is stationary reduced to 8% of the total[3]. This exercise provides evidence in support of the predictions made during the Heuristic evaluation, and suggests that the principal advantage of using the task mode as opposed to pre-taught positions, is the reduction in the cognitive demands placed upon the user.

Figures 8.12 and 8.13 show that a considerable proportion of the task is spent scooping food from the plate. The principal axis being operated to perform this action is the linear axis, axis 5. As described in section 8.1, the maximum speed of axis 5 was limited to 30 mm s^{-1}. Consequently, a medium speed had been set at around 24 mm s^{-1}. An alternative design decision would have been to provide one speed setting for the linear axes at 30 mm s^{-1}. This reduces the task duration by approximately 7 s. However, movement of the linear axis would still account for 41% of the total duration, suggesting that more significant design changes would be required to improve performance.

Task completion times for the drinking task were measured after a familiarization period of approximately half an hour, at which point a time of 7 minutes and 18 seconds was achieved. For the purpose of the following comparison, this is regarded as being representative of a novice user. Task completion times were also measured for an experienced or 'expert user' (the author), with the fastest run recorded as 4 minutes and 55 seconds. To allow these figures to be placed in a

[3] This figure is greater where voice control is employed, as interaction errors are far more frequent (see section 8.2.3.2).

wider context, two additional estimates were made : the time required for a PUMA 260 industrial robot to undertake the same task (operated by the author with a teach pendent), and the time required for the MANUS arm to undertake a similar drinking task, as shown in a promotional video (EEN EERSTE Experiment, December 1988). These values are compared in figure 8.14 below. The value of these comparisons is limited by the fact that different input devices are used by the different systems, and the controlling experience and functional ability of the operators varies. Additionally, the task undertaken by the MANUS is similar but not identical to that undertaken during the evaluation[4]. However, tele-operated control of a PUMA by an experienced operator may be regarded as representative of a limit achievable for the given task.

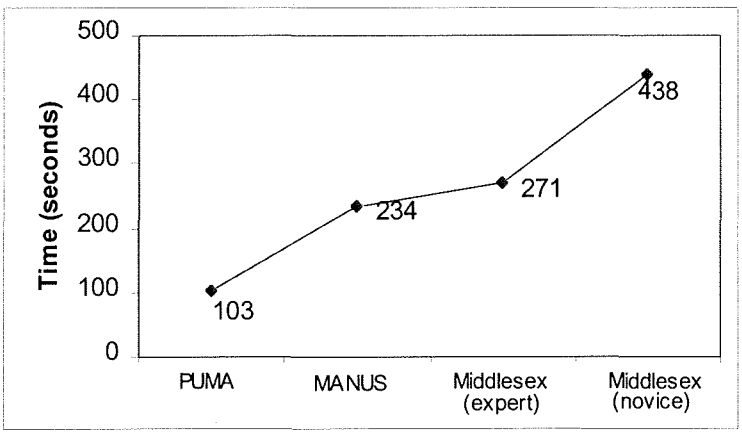


FIGURE 8.14 COMPARING DRINKING TASK COMPLETION TIMES

The time demonstrated by the MANUS represents performance acceptable to potential users (see Chapter 2). The times recorded of the Middlesex manipulator may also be acceptable, however, as part of an ongoing design cycle, it is valuable to examine how these times may be reduced.

For reference, Figure 8.15 below indicates the motor positioning for the first 5 manipulator axes. The speed levels employed for the angular axes (axes 2,3 and 4) were limited by the user's controlling ability, and were less than the maximum possible speeds for the axes.

However for the two linear axes (axes 1 and 5), the maximum speeds were employed. As discussed in section 8.1.1.1., these fall short of the original design targets.

[4] A straw is placed in a cup, which is then filled and presented to the user. However the tap is significantly different, as are the distances between objects.

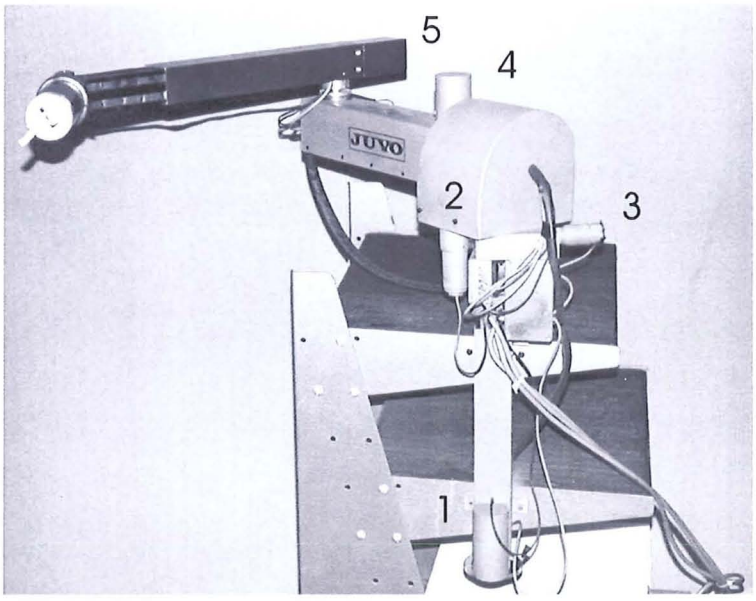


FIGURE 8.15 MANIPULATOR AXES 1 - 5

Consequently, as shown in figure 8.16, typically 47% of the total task completion time attributable to movement of the manipulator, corresponds to the linear axes.

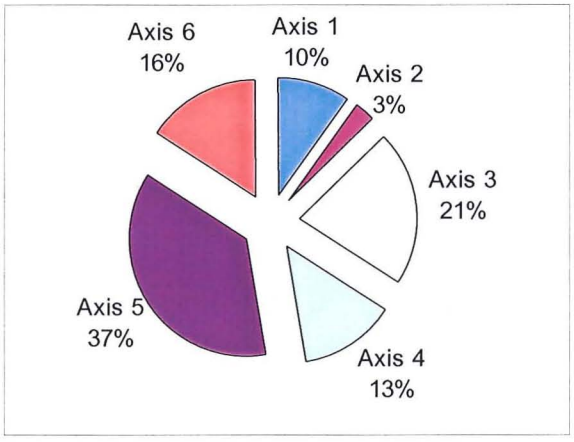


FIGURE 8.16 PROPORTION OF DRINKING TASK ATTRIBUTABLE TO EACH AXIS

The values shown in figure 8.16 are taken from a task undertaken by the author, with total task completion time of 271 seconds. This time is divided between movement, user interaction, and liquid being poured as illustrated by figure 8.17.

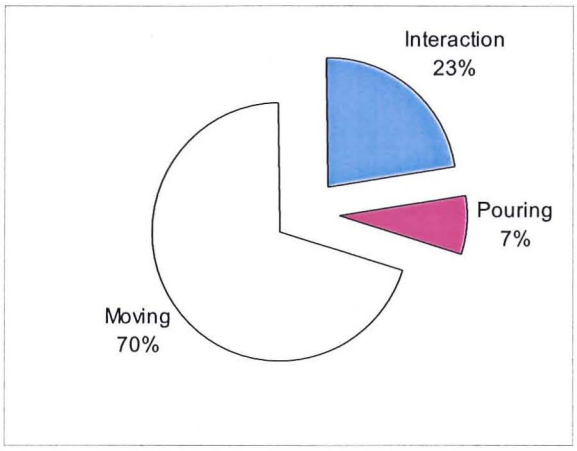


FIGURE 8.17 RELATIVE DURATION OF COMPONENTS OF THE DRINKING TASK

The total time attributed to movement is 190 seconds, of which 89 seconds is attributed to movement of the linear axes. If design modifications were implemented, allowing for the maximum speeds to be doubled to 60 mm s^{-1}, the resulting task completion time would be reduced to 226 seconds – a value previously deemed acceptable by MANUS users for such a task.

8.3.2 User interaction

The task completion times listed in the previous section relate to the manipulator being controlled by direct menu selection with a mouse as input device. The following section summarizes performance relating to the use of various input devices, namely : switch input , gesture recognition, and voice recognition.

Switch control
An electrolytic tilt sensor was employed as a switch, mounted on the evaluator's finger. This allowed for finger movement to select the currently highlighted option from a scanning system

Head Gestures
The head gesture recognition system described in Chapter 7 was used in conjunction with a scanning system. A vocabulary of 4 gestures was employed as follows :

Gesture 1 Used to select the currently highlighted option from the scanning system.
Gesture 2 Signifies STOP

Gesture 3 Short-cut to the MOVE sub-menu

Gesture 4 Short-cut to the GOTO sub-menu

Hand gestures (trackball)

A trackball was employed to allow encoding of simple hand gestures. A vocabulary of eight user-defined gestures allowed direct menu selection of enumerated menu commands.

Voice Control

A commercial voice recognition (Voice Server) was used to allow voice controlled direct-menu selection.

The evaluator used each of the forms of input to undertake a feeding task. Table 8.1 summarizes measurements made of the time required to retrieve food averaged over 5 runs. The percentage of time attributed to user interaction, and the times attributed to errors in interaction were also recorded.

	Food retrieval time (s)	% User interaction	Errors	Recovery time (s)
Mouse	65	8	0	0
Switch	109	40	1	9
Head	101	38	2	15
Voice	95	35	5	24
Trackball	> 300	-	-	-

Table 8.1 Comparing input devices

It became clear during the evaluation that the trackball gesture recognition system required a longer period of familiarization than was provided during the evaluation, if the device were to be used efficiently. Without appropriate familiarization, the trackball system results in slow user interaction, and places unacceptably high cognitive demands on the user. An approach to training an individual to use the trackball, and assess their performance is presented in Chapter 9. The following discussion focuses on a comparison of the remaining four modes of input.

As would be expected, directly selecting commands from a menu system provided the fastest form of interaction, and as the user had an appropriate degree of experience and functional ability, no errors in interaction occurred[5].

The switch input provided the slowest form of interaction, as a standard scanning system was employed (i.e. with a vocabulary size of 1, no short-cuts were available). Additionally, the 'AND' option discussed above could not be employed, as it is inappropriate to have any commands other than the 'STOP' command selectable whilst the manipulator is in motion. Interaction errors occurred corresponding to miss timing a selection. Errors occurred less frequently than with the more complex head gesture scanning system, but took longer to recover from.

The head gesture based system provided a moderate speed advantage over the switch input. Errors occurred corresponding to miss timing selections, either on the part of the evaluator, or as a result of the system failing to recognize a gesture with sufficient certainty (see Chapter 7 for a more comprehensive discussion of the recognition performance).

The Voice recognition system formed the second fastest method of control, though the frequency of errors was high. Typically, 24 commands would be issued during the retrieval of food from the plate. The error rate for voice input was measured at an average of 21%.

8.3.3. Learning effects

Task completion times for the drinking task are listed below, for 10 runs undertaken by the evaluator during stage 3 of the evaluation. The times were recorded after a familiarization period of approximately half an hour, and correspond to the time measured between commencing an approach to the straw, and presenting the cup to the user. The variation within the sample of 10 readings is fairly large (s.d. = 112.6) as there are a large number of variables that may effect the time measured. The principal cause for delay within a task was an overshoot of one of the manipulator's axes by the evaluator.

[5] Examples of interaction errors are : intending to select a specific command from the menu, and accidentally selecting another, or missing a command from a scanning system.

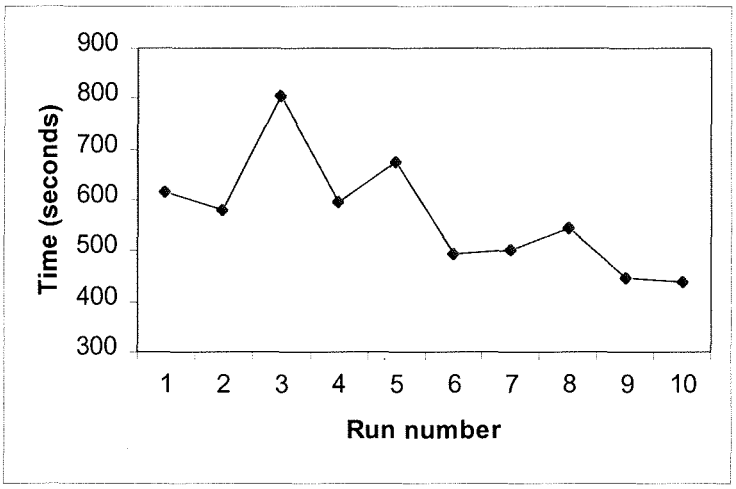


FIGURE 8.18 DRINKING TASK COMPLETION TIMES

The average task completion time measured was 569 s (9 minutes 29 seconds). However, figure 8.18 illustrates a downward trend, with times towards the end of the session approaching 438 s (seven minutes 18 seconds).

8.3.4 Conclusion

Analysis of the usage data captured during the user evaluation allowed for the predictions made during the Heuristic evaluation to be quantified, and for the performance of the various components of the system to be measured and compared.

The linear axes were identified as components of the system requiring improved performance in terms of speed. It was demonstrated that doubling the speed of the linear axes to 60 mm s^{-1} would allow the drinking task to be completed in a time comparable to that achieved by the MANUS system undertaking a similar task. It was argued that this is a useful yard-stick as user evaluations of the Manus have indicated acceptable levels of speed (see Chapter 2).

The benefits of direct menu-selection over scanning system in terms of menu-navigation were demonstrated, however it was shown that where a vocabulary exists greater than one, providing a scanning system with short-cuts increases menu navigation speeds for those unable to employ direct-menu selection.

Significant learning effects were demonstrated, with the evaluator achieving a final task completion time over a run of 10 trials, 23% faster than the average task completion time.

8.4 User feedback

8.4.1. Questionnaire design

Batavia and Hammer (1990) identified a number of consumer-based criteria for the evaluation of assistive devices. A modified Delphi Method was used to allow a panel of consumer experts to prioritise the issues in order of importance. The ordering was dependent upon the type of device under consideration, and is shown below for a robotic arm.

Effectiveness	Operability	Dependability
Affordability	Flexibility	Compatibility
Personal acceptance	Durability	Physical security
Learnability	Ease of maintenance	Supplier repair
Physical comfort	Consumer repair	Ease of assembly

As the current evaluation involves the use of a prototype, those issues regarding maintenance, repair and assembly were not included for questionnaire design. Additionally, assessments of durability and dependability would require a more prolonged evaluation within a home environment, and were therefore not addressed. The remaining topic areas were therefore :

Effectiveness	Operability	Affordability
Flexibility	Compatibility	Personal acceptance
Physical security	Learnability	Physical comfort

A second study was examined (Demers et. al., 1996) that highlights a number of satisfaction variables. A number of these were extracted that emphasize issues not explicitly referred to above, and are listed below.

Expertise	Accommodation by others	Safety
Dimensions	Appearance	Effort
Weight	Transportability	

A number of closed questions were then formulated as listed below. These allowed an initial response to be recorded on a 5 point scale. During the interview, a richer response would be drawn from the evaluator, by discussing each of the initial answers provided.

8.4.2 Questionnaire Responses

This section lists the closed questions used within the interview, and the evaluator's initial responses. The preferred form of input for the user was a mouse device, hence questions 1 to 18 refer to a manipulator system with a mouse used as input. Questions 19 to 26 address the relative usability of the different input devices available. These were repeated for each input device type, with the responses summarized in table form. Where the evaluator deemed a question inappropriate, no response is shown.

i) Effectiveness/Flexibility

1. Were user tasks successfully completed?

never		often		always
☐	☐	☐	☐	☑

2. Can tasks be undertaken in an efficient manner ?

not efficiently		fairly		very efficiently
☐	☐	☐	☐	☐

3. Is the manipulator flexible in the way that tasks may be performed ?

not flexible		fairly		very flexible
☐	☐	☑	☐	☐

ii) Expertise/Learnability

4. Was it easy or difficult to learn how to control the manipulator ?

very (difficult)		neither		very (easy)
☐	☐	☐	☑	☐

5. Did there appear to be a large, or a small amount of information to be learnt ?

a very small amount		neither		a very large amount
☐	☑	☐	☐	☐

iii) Operability/Effort/Comfort

6. Once you were familiar with the system, was the system easy or difficult to control ?

very difficult		neither		very easy
☐	☐	☐	☑	☐

7. Did use of the system require any mental effort ?

none		some		a great deal
☐	☐	☐	☑	☐

8. Did use of the system require any physical effort ?

none		some		a great deal
☑	☐	☐	☐	☐

9. Did use of the system cause any physical discomfort ?

none		some		a great deal
☑	☐	☐	☐	☐

iv) Appearance/Dimensions/Transportability

10. How would you describe the appearance of the system?

un-acceptable		fairly acceptable		completely acceptable
☐	☐	☐	☐	☐

11. How would you describe the size of the arm ?

un-acceptable		fairly acceptable		completely acceptable
☐	☐	☑	☐	☐

12. Using a system mounted on a mobile platform would be ?

un-acceptable		fairly acceptable		completely acceptable
☐	☐	☐	☑	☐

13. Using a system mounted on a wheelchair would be ?

un-acceptable		fairly acceptable		completely acceptable
☐	☑	☐	☐	☐

v) Acceptance, Compatibility, Affordability

14. Would you consider using a robotic arm for assistance at home ?

definitely not		not sure		definitely
☐	☐	☐	☑	☐

15. Would you consider using a robotic arm for assistance outside of the home ?

definitely not		not sure		definitely
☐	☑	☐	☐	☐

16. Do you think the use of a robotic device would be acceptable to others around you ?

definitely not		not sure		definitely
☐	☐	☐	☑	☐

17. Do you currently use forms of technology that you imagine would be incompatible with a robotic device ?

definitely not		not sure		definitely
☐	☑	☐	☐	☐

vi) Safety

18. Did you feel safe while using of the manipulator ?

definitely not		not sure		definitely
☐	☐	☐	☑	☐

vii) Input device comparisons

The following questions were repeated for each of the input device types used : Switch (scanning system), Head gesture (scanning system with short-cuts) Voice (direct menu selection), Mouse device (direct menu selection), Trackball (direct menu selection)

19. Was it easy or difficult to learn how to use the device ?

very difficult	neither	very easy

20. Once you were familiar with the device, was it easy or difficult to use ?

very difficult	neither	very easy

21. Did use of the device require any mental effort ?

none	some	a great deal

22. Did use of the device require any physical effort ?

none	some	a great deal

23. Did use of the system cause any physical discomfort ?

none	some	a great deal

24. The number of errors that occurred while using the device seemed :

un-acceptable	fairly acceptable	completely acceptable

25. The speed of communication the device allowed seemed :

completely unacceptable	fairly acceptable	completely acceptable

26. The physical appearance of the device is :

completely unacceptable	fairly acceptable	completely acceptable

Table 8.2 summarizes the results of the input device comparisons as follows :

Score = 5 for most positive response (i.e. no mental effort, completely acceptable appearance etc).

Score = 1 for least positive response.

Score = 0 if question deemed inappropriate and not answered.

	Mouse	Switch	Head	Voice	Trackball
Easy to learn	5	5	4	2	1
Easy to use	5	5	4	3	0
Mental effort	5	3	2	2	1
Physical effort	5	5	3	5	5
Discomfort	5	5	3	5	5
Error frequency	5	2	3	2	1
Speed	3	2	2	3	0
Appearance	5	5	3	3	5

Table 8.2 input device comparisons

8.4.3. Interview summary

The following section provides a discussion of the questionnaire results provided above, and a summary of the additional feedback provided by the evaluator.

Having seen but before using the manipulator, the evaluator expressed the expectation that the manipulator would 'probably not' be capable of undertaking feeding and drinking tasks. However, he felt that it was likely that the system would be useful for personal hygiene tasks. This latter prediction was based on personal experience of devices such as electric toothbrushes and electric razors, and the fact that many people with physical disabilities have a good range and control of head movement (specifically people with spinal-cord injuries). The evaluator's prediction that the arm would be unable to perform feeding and drinking tasks has to be interpreted in the context of the evaluator being unfamiliar with the idea of adapting an environment for a manipulator. This suggests that tasks should be defined in an appropriate form, and that task descriptions include a description of the operating environment.

The evaluator felt that the question regarding the manipulator's appearance were not of paramount importance, as the manipulator was at the prototype stage. However, he felt that aspects of the arm's appearance would need to be improved, such as :

- replacing the square edges and sharp corners with a more rounded feel;
- paying more attention to the use of colour (particularly brighter colours);
- using softer materials (i.e plastics); and,
- hiding all motors and cables.

The evaluator felt that the size of the arm was acceptable, provided that an appropriate park position existed. However, the evaluator felt that a system mounted on a mobile platform would be more popular than a wheelchair-mounted system, as the mobility of the wheelchair would be effected. The evaluator therefore felt that the manipulator would be of most use within semi-structured environments around the home. At the time of the evaluation, the evaluator was employed in an environment where the majority of people have physical disabilities, and felt that other forms of assistive technology (voice recognition for word processors) are of far greater importance than robotic technology.

The quality of movement of the manipulator was deemed as 'fairly good' and 'not jerky', though the level of responsiveness of the system often resulted in the evaluator overshooting a target (this effect reduced as the evaluator became more familiar with the system). The perceived accuracy of the system when moving to pre-taught positions was regarded as 'fairly good'.

When the manipulator was operated at elevated speeds (higher than those eventually adopted as the three speed settings) the level of noise generated by the manipulator was regarded as unacceptable. This was partly due to the varying pitch of the noise having a significant impact on the evaluator's confidence in the system.

The evaluator felt that it was extremely important that a command can be issued at any point during user interaction to stop the arm if it is moving. When presented with a scanning system that did not conform to this at certain points during interaction, the evaluator felt that use of the system was 'scary'.

The evaluator's overall impression of a scanning system was that it was often frustrating waiting for the required command to be selectable. Though this was regarded as less of a problem where a vocabulary of gestures allowed short-cuts to be employed. The favored form of direct menu selection was the use of mouse or trackball, as voice recognition suffered higher recognition error rates. Additionally, it was considered advantageous that use of the mouse draws less attention than the use of voice.

The system was regarded by the evaluator as being easy to learn, and easy to use. This may be attributable to the fact that the system's adaptability allowed for an interface of limited complexity to be provided to the evaluator. Options such as the number of pre-taught positions, and the number of speed levels available, were limited to the minimum required for the tasks of interest. Additionally, the modes of control corresponding to teaching and executing pre-taught tasks were not provided. However, the functionality provided ensured that tasks were always completed (i.e it was possible to recover from any errors committed). The evaluator felt that the system allowed flexibility in the way that tasks were achieved, as illustrated by the improved efficiency in controlling the manipulator (see figure 8.18).

8.4 Assessment against general design criteria

Chapter 2 of this thesis presented a review of the field of rehabilitation robotics, from which a set of general design criteria were defined. The following section provides a discussion of the manipulator evaluation results in the context of these criteria, estimating the degree to which each criteria has been conformed with or violated.

8.4.1 Criteria conformance

Cost

The design specification outlined in Chapter 3 included a target cost of £5000, a figure influenced by the current retail price of HANDY 1. Adopting purpose built motor control circuitry and low-cost embedded micro controllers resulted in a one-off component cost £440 for the motor control system. The DC motors incur the largest cost at £1380. Materials for the manipulator were calculated at approximately £400 (Heide & Roorda 1993, Buter & Veltman, 1996). Embedded PC platforms for the user interface system are available at £260 (RS Components). The gesture recognition system, along with an LCD feedback display may be included at £280.

The total component and materials cost for the current prototype is therefore £2760[6]. A cost for system design is not included, as the design has resulted from the research program of the author. The prototype cost is therefore comfortably within the target of £5000.

If a future prototype were to lead to a commercial product, a cost of production would need to be included. This may be partially offset by a reduction in component costs for bulk purchases.

Functionality

Design criteria specified that the system should be general purpose, providing functionality that addresses a range of user needs. The user evaluation demonstrated that feeding, drinking and 'pick and place' tasks can be successfully undertaken with the current prototype. The functionality of the system is likely to increase with the completion of the three-axis end-effector. This will increase the degrees of freedom of the manipulator, and allow for the inclusion of the cartesian mode of control.

[6] This cost excludes parts for the planned three axis end-effector

Performance

The design criteria required that the base-line performance characteristics should match the requirements of the user tasks addressed. The accuracy of the manipulator is reported in some detail above, however the success of the evaluator repeatedly completing tasks, demonstrates conformance with this criteria in terms of accuracy. Additionally, the system's payload was demonstrated as being ample for a range of user tasks. However, the limited velocity of the manipulator's linear axes suggests that this criteria may not be met in terms of speed. The evaluation demonstrated significantly slower task completion times than the MANUS system undertaking a similar task.

Mobility

The manipulator was originally designed for wheel-chair mounting. However, its dimensions suggest that problems with wheelchair mobility would result, as was the case with the MANUS system. The user evaluation suggests that for many users, a preferable option would be to mount the manipulator on a mobile platform. This was demonstrated as feasible during the evaluation.

The resulting weight of the prototype was 8kg, thereby conforming with the weight target.

Input devices

The user interface system's modular design allows for the use of a variety of different input and feedback devices. Any new device may be used with the system by developing an appropriate device driver module.

Variety of control modes

Similarly, the modular interface design provides a variety of control modes including tele-operation, pre-taught positions and pre-taught routines.

Adaptability

The design criteria required that ease of use should be enhanced by allowing systems to be configured to match individual user needs. Again, the modularity inherent in the interface design allowed for this. Additionally, the user evaluation confirmed the benefits of this approach.

Appearance

Whilst it should be recognised that the manipulator is at the prototype stage, it must be noted that significant modifications to the design are required to improve appearance. This includes : the removal of sharp edges, hiding the motors, leads and feedback sensors, and possibly a reduction in the manipulator's size. No design effort has previously been directed at addressing these issues, and the magnitude of the modifications required suggests that these issues should have been addressed at a far earlier stage in the manipulator's design process. As this has not been done, it can not be claimed that this criteria is in any way conformed to.

Safety

Safety was partially addressed by the inclusion of software limits, processor redundancy, and hardware stops. Subjective feedback from the evaluation suggested that the evaluator felt safe when operating the arm. However, it would be essential for future work to include a more thorough approach to ensuring system safety. This should include a formal assessment of the manipulator's impact on associated standards, such as IS0 7176 (wheelchair stability), and IEC 529 (degree of protection offered by enclosures).

Design modifications

The remaining design criteria stated that the design should facilitate future modifications to system performance and functionality, and to enhance user acceptance of the device. As, mentioned above, an area of concern in terms of performance is the velocity of the linear axes. This was limited by the noise generated by the bearings at elevated speeds. The manipulator's construction does not prevent the replacement of the bearing material with a more appropriate substitute, and the potential application of a lubricant. Thus the design would facilitate an investigation to this end.

The modular design of the user interface system would allow for rapid modifications to be implemented to the modes of control, and hence the manipulator's functionality. This is also true of the manipulator's end-effector, which is designed to have replaceable gripper attachments

The most problematic issue with regards to design modification is likely to be the system's appearance. As mentioned above, aesthetics has not been comprehensively addressed by the existing design. The manipulator's construction was not designed to house leads and motors, and employing additional casing would increase the manipulator's bulkiness, potentially detracting

from user acceptance. It should therefore be concluded that the current design does not facilitate modifications to improve appearance.

8.5 Summary

This chapter began with a heuristic evaluation of the manipulator's user interface system. The results demonstrated that although conventional HCI evaluation methodologies do not address all of the issues relevant to the design of assistive technology, they may still be of use within the early stages of the design process. Applying usability heuristics resulted in a number of design issues being addressed that had been overlooked during the initial design stages. The process also allowed for an estimation of the severity of each design problem, and the solution cost. The validity of a number of the issues raised was then shown by further evaluation.

The user evaluation demonstrated repeated successful task completion by a potential end-user. The evaluator was generally satisfied that the system was easy to use, and provided evidence to support the use of multiple forms of interaction.

The evaluation highlighted a number of required design modifications, relating to the speed of two of the manipulator's axes, and the system's appearance. The results of the evaluation were presented in the context of the design criteria identified in Chapter 2. It was shown that the manipulator conforms to all but two of these criteria. The consequence of this analysis is discussed further in Chapter 10.

Section V

Section V

Adapting the user interface

Chapter 9

Using Task Analysis to Configure an Adaptable User Interface

Chapter 6 presented the design of an adaptable user interface that may be configured to employ a number of different input devices and interaction styles, and to provide varying levels of functionality. The design allows systems to be configured to match the requirements and controlling ability of specific users. The configuration process can be based predominately on user preferences and the experience of the clinician. However, a procedure or technique is required to allow objective measurements of a user's controlling ability to be included.

This chapter describes the development and testing of a novel methodology that allows the relative usability of possible interface configurations to be predicted, based on individual user and device characteristics. An experiment was undertaken, assessing task completion time predictions, generated by an analysis based on GOMS Task Analysis (Goals, Operators, Methods and Selection).

The form of Task Analysis developed and applied within this chapter is unique, and was shown to consistently predict the relative usability of interface configurations.

9.1. Introduction

Clinical evaluations of rehabilitation robotic systems have indicated that a greater degree of acceptability may be achieved by developing systems that can be adapted to match user requirements, preferences, and functional ability (Kwee and Duimel, 1988; Topping, 1995). Motivated by this, Chapter 6 of this thesis presented the design of a novel interface and control system that may be configured for a specific user in terms of :

- Functionality

System functionality is determined by the number and type of software modules (referred to as modes of control) that are present within the system. Functionality may vary from that of a simple feeding-aide, to a fully user-programmable robot. The former requiring only one control command for operation, whereas the latter requires an extensive vocabulary of commands.

- Input Modality

The system supports a number of possible input modalities including a commercial voice recognition system, gesture recognition, and a variety of on-off switches.

- Interaction Style

Control commands are organised into a menu system displayed by an LCD screen. This may be navigated using either command encoding, direct menu selection, keyboard emulation, or various forms of scanning system.

The system's adaptability necessitates a methodology that may be employed to assist with the installation and configuration of the system. Typically, clinicians involved in the selection and installation of assistive technology attempt to match devices with user requirements employing an approach that may be described as "pseudo-systematic and subjective" (Kondraske, 1988). A similar approach may be required here to attempt to match system functionality with the tasks the user wishes to undertake, and to match input modality with user preferences and functional ability. However, for a given level of interface complexity and input device type, a more objective approach to selecting an appropriate interaction style may be possible.

For a given user, the appropriate style of interaction is likely to be that which allows for faster menu navigation and command selection, as well as conforming to user preferences (a hypothesis supported by the user evaluation reported in Chapter 8).

For a specific user, speed of interaction will be dependent upon how many distinctly different signals may be reliably issued with the device, and how long it takes to select a command with an encoded signal. As an example, direct menu selection is typically a faster form of interaction than a scanning system. However, where gesture recognition is being used, direct menu selection requires a larger vocabulary of gestures, which are typically more complex, taking longer to recall and issue. This may mean that for a specific individual, the combination of : input device, interface complexity and controlling ability, would result in negligible gain in interaction speed from direct menu-selection. Clearly, selecting the appropriate form of interaction based on this combination of variables would be difficult to optimise by purely subjective means.

In general terms, the variables of interest are therefore :

- the user's controlling ability (i.e. those aspects of general functional ability relevant to the control of the selected input device(s));
- the characteristics of the available or selected input device(s);
- the style of interaction selected;
- the nature and number of user tasks addressed; and,
- the number of available or selected control modes.

The variables that would typically be required to be minimised are :

- task completion time, and
- error rates.

The following analysis categorises these variables as follows :

- Controlling ability - fixed[1]
- Device characteristics - fixed
- User tasks - fixed
- Control modes - fixed
- Style of interaction - independent variable
- Task completion time - dependent variable.

Error rates are excluded, as it may be reasonably assumed that interface configurations that minimise error rates are likely to be those that allow for faster task completion times. Thus for a selected input device, set of user tasks and control mode, the analysis is required to determine the style of interaction that minimises task completion time.

9.2 Task Analysis for Interface configuration

GOMS task analysis techniques have been successfully applied to predicting task completion times, and have proved particularly successful at predicting the relative task completion times for different interface designs (Nielsen, Phillips, 1993). Based upon user models, the techniques avoid the overheads of user testing, and may be employed early on in the design cycle, to yield both qualitative and quantitative estimates of design options, such as task completion times, task learning times, interface consistency and functionality.

The objective of the experiment described below, is to determine whether the principles underlying Task Analysis may be extended and applied to configuring an adaptable system, if the appropriate individual user and device characteristics are included. These are used in place of the standard parameters derived from the Model Human Processor developed by Card, Moran and Newell (1983).

[1] A user's controlling ability may vary as a result of increasing experience, or variable states of health, or may remain constant if the user is already familiar with the input device in question. The analysis includes no facility for predicting variations in controlling ability and therefore assumes a degree of constancy in controlling ability.

The approach taken was to estimate these characteristics empirically, while subjects were trained to use a particular form of input device. User interaction was then modelled for typical user tasks, by decomposing task goals into sub-goals and lower-level actions, described in a format similar to Natural GOMS Language (NGOMSL), as outlined by Kierass (1988). This process is easily automated in the current context, as there is a limited set of interactions to be described.

9.3. Experimental Objectives

The principal objective was to quantify any correlation between predicted task completion times and measured task completion times for two configurations of the Middlesex Manipulator user interface. The particular configurations of interest were :

- A direct menu selection system. This employed a trackball as an input device allowing simple gestures to be issued by hand, and encoded. Vocabularies of enumerated gestures were used, with each gesture corresponding to a single menu option.

- A scanning system with short-cuts. As above, a trackball formed the input device, with the issuing of a particular gesture causing the currently highlighted menu option to be selected. Three further gestures were available that allowed the following short-cuts to be taken : i) jumping to the 'Goto' sub-menu, ii) jumping to the 'Move' sub-menu, and iii) immediately selecting the 'Stop' command.

The experiment required the development of an appropriate format for modelling user interaction, based on NGOMSL Task Analysis. A method was also required for obtaining estimates of user characteristics, once performance with a particular input device had become asymptotic.

The experiment's objectives may therefore be summarised as follows :

- To develop a novel form of task analysis based on NGOMSL task analysis that incorporates estimates of individual user and specific device characteristics in task modelling;
- To develop an appropriate method of estimating user and device characteristics; and,

- To quantify the correlation between task completion times predicted by task analysis, and measured task completion times.

9.4. Method

It was not the author's intention to create a homogeneous group of subjects, or to discount user differences through experiment design, as is typically the case where experiments compare interface designs. Rather, the current investigation involved measuring and using user differences within the analysis.

The concept of homogeneity may only be applied, and to a limited extent, to sub-groups within the disabled community, such as those with a certain level of spinal cord injury. However the Middlesex Manipulator is not geared towards any one sub-group exclusively. Additionally, standard classifications of disability are not likely to provide adequate descriptions of individuals where the use of assistive technology is concerned. A physical impairment that disables an individual from walking, may not result in an inability to use a head gesture recognition device. It is appropriate therefore to focus on the match between the user and device, expressed as controlling ability, rather than just the user's general functional ability.

It was preferable to create a situation where user differences were significant, to produce a range of predictions that could be correlated with a range of measurements. Ideally, this would have been achieved by selecting a subject group consisting of people with varying physical impairments. However, due to the time constraints for the current phase the project, only one of the subject group was physically disabled (the evaluator used in the evaluation described in Chapter 8). As a result, the practicalities of including Task Analysis within a clinical environment are not addressed here. Rather, the experiment focuses on the ability of a novel form of task analysis to capture user differences (whatever their origin) within the modelling process.

Students and members of staff within Middlesex University were recruited to form a subject group of 6 people in total. To increase the diversity of performance levels the subjects were not allowed to select their own gestures, but were required to use a pre-determined vocabulary of gestures. Three such vocabularies were created, containing gestures with varying levels of complexity.

9.4.1 Modelling Tasks and User Interaction

Chapter 6 discussed how hierarchical task analysis can be used to define components of user tasks. The results may be described in written form as shown in figure 9.1

> To pick up an object that is fairly close to a pre-taught position, and move it to another pre-taught position :
>
> - First, set a speed appropriate for gross movement,
> - then, use the control mode that allows you to move to a pre-taught position,
> - then, set a speed appropriate for fine movement,
> - then, adjust the Manipulator's joints as necessary,
> - then, close the gripper,
> - finally, use the control mode that allows you to move to a pre-taught position.

FIGURE 9.1 USER TASK DESCRIPTION

Typically, an NGOMSL analysis would be used to model user interaction, capturing details of the interface and task descriptions. At the highest level, this may appear as shown in Figure 9.2.

> Method to accomplish goal <Pick & Place>
> Step 1. Accomplish goal <Set Speed *Medium*>
> Step 2. Accomplish goal <Goto *Position 1*>
> Step 3. Accomplish goal <Set Speed *Slow*>
> Step 4. Accomplish goal <Move Joint *Base, Out*>
> Step 5. Accomplish goal <Move Joint *Shoulder, In*>
> Step 6. Accomplish goal <Move Joint *Hand, Out*>
> Step 7. Accomplish goal <Close Gripper >
> Step 8. Accomplish goal <Goto *Position 2*>
> Step 9. Report goal accomplished.

FIGURE 9.2 NGOMSL TASK DESCRIPTION

Each of the methods for the sub-goals referred to would then be defined as in figure 9.3 below. For typical user interaction, this would include the use of primitive mental operators, reflecting cognitive

processing, and external operators such as mouse moves and mouse clicks (see Card Moran and Newell, 1983).

Method to accomplish goal <Set Speed to Speed_Level>
Step 1. Retrieve from LTM* menu option *Speed*, and retain in WM†.
Step 2. Recall menu option and move mouse.
Step 3. Click left mouse button.
Step 4. Retrieve from LTM* menu item *Speed Level*, and retain in WM†.
Step 5. Recall menu item and move mouse.
Step 6. Verify result
Step 7. Forget menu item, forget menu option.
Step 8 Report goal accomplished.

* Long Term Memory
† Working Memory

FIGURE 9.3 NGOMSL SUB-GOAL DESCRIPTION

As the approach adopted here requires a set of operators that quantify a specific individual's characteristics, these must be easily observable for specific users during training. The four operators employed are :

i) Prepare

The Prepare operator corresponds to the time measured between forming an intention to issue a command, and beginning to issue a command. This would be catered for in NGOMSL analysis with the combination of primitive operators such as retrieve, retain and recall.

ii) Issue

This corresponds to the time taken to issue a command. Examples would be physically performing a gesture, or completing an utterance.

iii) Verify

As with the analyst defined operator common in NGOMSL, this corresponds to the user verifying that the action taken has had the desired effect.

iv) System Delay

Similar to system response time in NGOMSL. System delay time is required to account for the time the user would wait for an option to be selectable by a scanning system. As the same goal description is used to model both scanning systems and direct menu selection, System Delay can be set to zero for the latter. Similarly, the Prepare and Verify operators can be set to zero for a scanning system, as this would occur in parallel with System delay.

The model may then be expressed as :

Method to accomplish goal <Set Speed to Speed Level>
Step 1. System Delay
Step 2. Prepare to select Speed option
Step 3. Issue command corresponding to Speed
Step 4. Verify
Step 5. System Delay
Step 6. Prepare to select Speed Level
Step 7. Issue command corresponding to Speed Level
Step 8. Verify
Step 9. Report goal accomplished

FIGURE 9.4 TASK DESCRIPTION WITH NEW OPERATOR SET

9.4.2. Predicting Task Completion Time

A typical pick and place task was modelled as described in 9.4.1, with the values of the primitive operators dependent upon the user, the input device and the style of interaction. Two styles of interaction were considered : direct menu selection and a basic scanning system. The Task Analysis process was automated using a spreadsheet to contain task descriptions and operator values, and a module containing Visual Basic functions to perform the associated calculations. Figure 9.5 below shows an example visual basic subroutine that sets variables corresponding to user characteristics from cells within a worksheet.

```
Sub MainRoutine()
        For CellValue = 2 To 13 Step 1
                Resolve = Cells(CellValue, 7).Value
                Issue = Cells(CellValue, 8).Value
                Verify = Cells(CellValue, 9).Value
                ScanRate = Cells(CellValue, 10).Value
                Cells(CellValue, 11).Value = CalcT()
        Next CellValue
End Sub
```

FIGURE 9.5 EXAMPLE SPREADSHEET SUBROUTINE

A function is then called to calculate the estimated task completion to for a specific task (a full listing of the set of visual basic functions is provided in appendix I). This approach allows any task to be described as part of a spreadsheet, allowing the time calculation function (CalcT) to extract the names of menu commands from the spreadsheet, and estimate the time required to select the command with the given user characteristics. The task described for the purpose of the experiment is illustrated in figure 9.6 below.

M. <Pick & Place>				
	<Set Speed>		Med	
	<Goto>		Side	Two
	<Move>		Shoulder	In
	<Move>		Base	Out
	<Move>		Elbow	In
	<Move>		Wrist	Out
	<Move>		Hand	Out
	<Goto>		Front	Mid
	<Move>		Hand	Out
	<Move>		Hand	In
	<Goto>		Home	One
	Report <> Accompilshed			

FIGURE 9.6 SPREADSHEET TASK DESCRIPTION

9.4.3 Estimating user characteristics

An application was developed for the purpose of familiarising users with input devices, and measuring characteristics corresponding to the operators outlined above. For familiarisation with gesture recognition, the application provides a menu of commands and a number associated with each command, allowing for direct menu selection. The user is then provided a cue to select a particular command. The gesture corresponding to the number associated with the command is performed, and classified by the gesture recognition system. Feedback is then provided, informing the user of either successful command selection or the occurrence of an error. The user is required to verify whether the result is correct or not with either a short movement to the left or right of the input device.

Users would undertake training sessions with the program, allowing the mean and standard deviation of the last n command selections (typically 40) to be used to calculate task completion time predictions.

As the values for primitive operators are point estimates, it was interesting to see how the task completion time predictions may vary as a result of error on the primitive operator estimates. This was achieved by constructing confidence intervals for each estimate at a level of $\alpha = 0.05$. The values at the extremes of the confidence intervals were used to allow each task completion time prediction to be represented as a range.

9.4.4 Measuring Task Completion Time

For comparison with the predicted task completion times, subjects were timed undertaking the pick and place task. Six values were measured for each user, from which an average was calculated. As the experiment focused on navigating the interface, the manipulator was disconnected during the trial.

9.4.5 Experiment design.

Subjects were assigned interface configurations and gesture vocabularies as shown in figure 9.7 below: Measurements were taken during two separate phases. Within each phase each subject undertook the pick and place task 6 times with each interface configuration, from which the average task completion

time for the subject was calculated. A familiarisation period of half an hour was undertaken before phase 1 (using the program described in section 9.4.3.). A second familiarisation period of half an hour was undertaken by each subject between phase 1 and phase 2.

Phase 1	**Subject**	**Interface**	**Vocabulary set**
	A	scanning followed by direct	1
	B	direct followed by scanning	1
	C	scanning followed by direct	2
	D	direct followed by scanning	2
	E	scanning followed by direct	3
	F	direct followed by scanning	3

Phase 2	**Subject**	**Interface**	**Vocabulary set**
	A	direct followed by scanning	1
	B	scanning followed by direct	1
	C	direct followed by scanning	2
	D	scanning followed by direct	2
	E	direct followed by scanning	3
	F	scanning followed by direct	3

FIGURE 9.7 EXPERIMENT DESIGN

9.5 Results

Figure 9.8 shows the predicted and measured task completion times for each subject using the scanning system during phase 1. The maximum difference between the predicted and measured values is 9%, with the average difference being 6%. Error bars are used to show the predicted values as a range, computed as the limits of a 95% confidence interval (based on a sample of measured user characteristics n = 40). The measured values fall within the predicted range for each of the six subjects. The average measured value is 214s, with a standard deviation of 5.8.

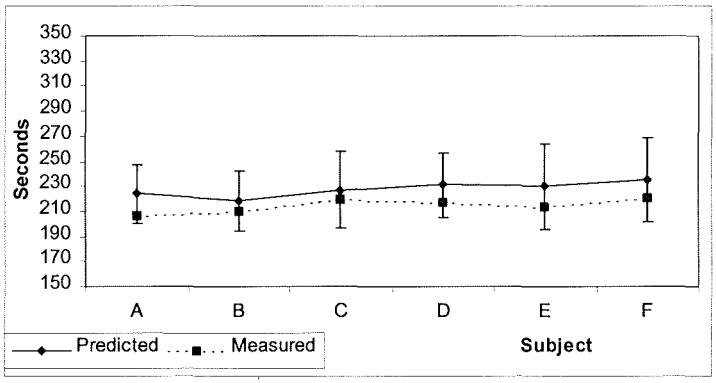


FIGURE 9.8 SCANNING SYSTEM – PHASE 1

Figure 9.9 shows the predicted and measured task completion times for subjects using the direct menu selection system during phase 1. The maximum difference between the predicted and

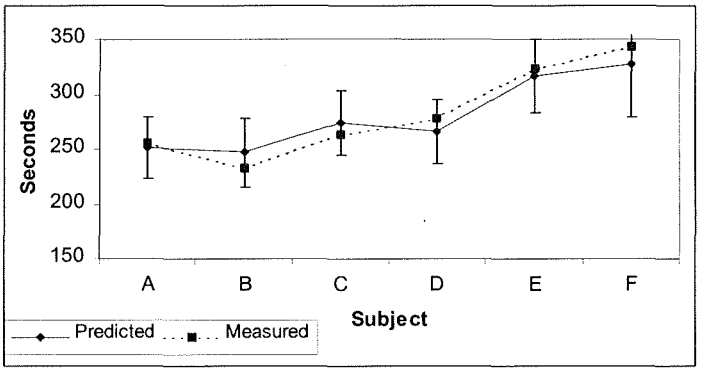


FIGURE 9.9 DIRECT MENU SELECTION – PHASE 1

measured values is 6%, with the average difference being 4%. As above, the measured values fall within the predicted range for each of the six subjects. The average measured value is 283s, with a standard deviation of 42.7. Greater evidence of a difference between gesture vocabularies is shown than with the scanning system. The average measured task completion times for vocabularies 2 and 3 are greater than measurements for vocabulary 1 by 11% and 37% respectively.

Figure 9.10 shows the predicted and measured task completion times for the scanning system during phase 2. The maximum difference between the predicted and measured values is 7%, with the average

difference being 4%. Again, the measured values fall within the predicted ranges. The average measured value is 215s, with a standard deviation of 6.8.

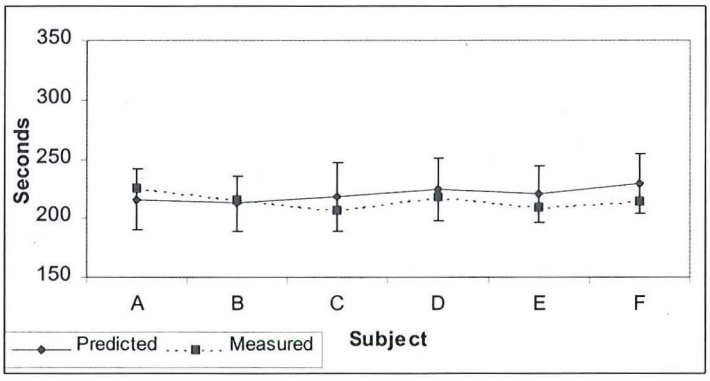


FIGURE 9.10 SCANNING SYSTEM – PHASE 2

Task completion times for the direct menu selection during phase 2 are illustrated in figure 9.11. The maximum difference between predicted and measured values is 6%, with the average difference being 3%. Measured values fall within the predicted range, with the average measured value 197s, and standard deviation of 36.4. As with phase 1, greater evidence of a difference between gesture vocabularies is shown than with the scanning system. The average measured task completion times for vocabularies 2 and 3 are greater than measurements for vocabulary 1 by 14% and 47% respectively.

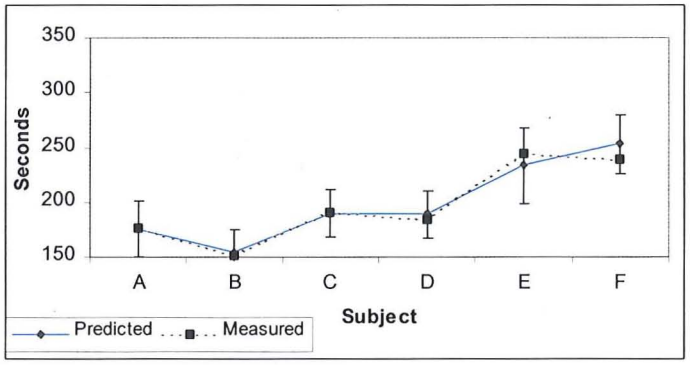


FIGURE 9.11 DIRECT MENU SELECTION PHASE 2

The results show that during phase 1 the scanning system was faster than the direct menu selection system for all six subjects, whereas during phase 2 the direct menu system was measured as being

faster for four of the subjects. The benefit of using a direct menu system in preference to the scanning system in terms of task completion time may be expressed as the difference between the task completion times for the two systems. This value, referred to here as gain, is illustrated for each of the subjects in figures 9.12 and 9.13 below. Predicted and measured gains are shown for both phases.

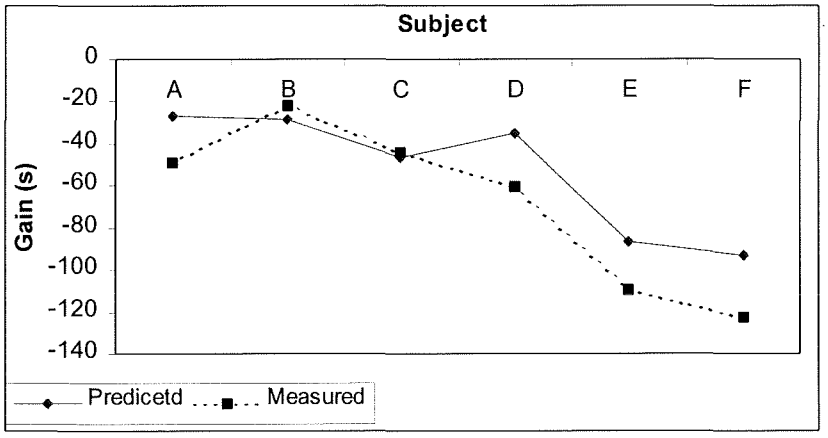


FIGURE 9.12 PREDICTED GAIN – PHASE 1

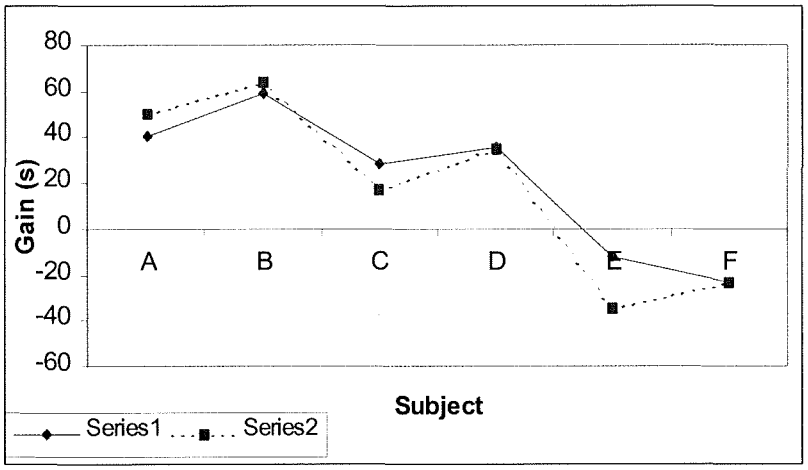


FIGURE 9.13 PREDICTED GAIN – PHASE 2

9.6 Conclusions

The variance recorded for the scanning system is less than that for direct menu selection, indicating that the former is less sensitive to user characteristics. The scanning system provided the faster form of interaction in the initial phase, however four of the subjects (those using the simpler gesture vocabularies) achieved faster interaction with direct selection during the second phase.

Over the two phases of the experiment, the average differences between predicted and measured values for the scanning and direct selection systems were 5% and 4% respectively. These figures suggest that the form of task analysis employed provided an accurate model of user interaction. Additionally, the measured variance in user characteristics, used to express predicted values as a range, are adequate to account for measured and predicted differences. Furthermore, the measured and predicted gains for all subjects over both phases are of the same sign, hence consistently correct predictions of the faster form of interaction were made.

The results demonstrate that user and device characteristics may be captured during a process of user training, and used to form an accurate model of user interaction. Successful predictions of the relative usability of interface configurations were made consistently. The results show that task analysis may be used during the process of configuring an adaptable interface for a specific user, providing objective measures to complement the subjective preferences of the individual.

As discussed in Chapter 5, a factor that has limited the application of task analysis to interface evaluation in commercial environments, is the complexity involved in providing a formal description of interaction for each interface design of interest. An advantage that results from the application of task analysis as described in this chapter, is that much of the process may be automated. The description of interaction for each interface configuration is provided once, following which any number of usability predictions may be generated with the appropriate insertion of specific user and device characteristics.

Section VI

Discussion & Conclusions

Chapter 10

Conclusions and further work

The work reported in this thesis has made a number of practical and theoretical contributions to the field of rehabilitation robotics. These have centered around the development of an adaptable user interface and control system for a novel rehabilitation robotic arm:

- Requirements Analysis

  This work identified general design criteria. It was shown that existing systems did not adequately conform to these criteria, and that levels of conformance could be used to predict or explain the relative levels of success of existing projects.

- Novel Design

  The construction of the Middlesex Manipulator : a prototype implementation of the novel Scariculated Kinematic configuration. This work focused on the design of a highly modular and adaptable low-cost user interface and control system, and included the integration of novel forms of gesture recognition.

- Evaluation

  An extensive user evaluation of the manipulator identified areas of non-conformance with design criteria, and allowed for the prioritization of areas for future work.

- Configuration

  The development and evaluation of a novel form of Task Analysis, that may be used to configure an adaptable user interface based on user's controlling ability.

This chapter discusses these contributions, and outlines possible areas for future work.

10.1 Contributions to research

The initial research objective was to assess whether a prototype implementation of the Scariculated kinematic configuration would conform to design criteria appropriate for rehabilitation robot design. This was addressed through the development of the 'Middlesex Manipulator', which began with a review of the field of rehabilitation robotics, focusing on extant systems and the user feedback that these had elicited. Chapter 2 of this thesis examines a number of systems that are representative of the successes and failures of the field, and argues that a cohesive picture may be drawn from an analysis of user evaluations, and the relative levels of user-acceptance that these systems have achieved. This analysis allowed for a set of ten general design criteria to be specified.

The variety of projects that exists within rehabilitation robotics has allowed for a range of lessons to be learnt. Universally accepted design criteria would help prevent these lessons from being repeated. As discussed in Chapter 2, general design criteria should be expected to evolve in line with technological change and evolving user expectations. However their definition, as attempted within this thesis, will assist the field of rehabilitation robotics in progressing towards the delivery and wide-spread user acceptance of general-purpose robotic devices.

The design criteria identified provided a framework for the development of a control system and user interface for the Middlesex Manipulator. A multi-disciplinary approach was adopted, in which techniques developed within the fields of human-computer interaction, software engineering and artificial intelligence, were adapted and applied. This work was novel, due to the limited formal application of HCI and AI to rehabilitation robot design. The applicability of techniques such as Task Analysis and Heuristic evaluation were assessed, showing that within certain constraints, these techniques could be successfully applied to the design of assistive technology.

The resulting design provided a level of adaptability beyond that of comparable systems, allowing for the use of novel input devices, and prioritising low-cost. The system was evaluated by an individual with spinal-cord injury, and the results were used to assess the Manipulator against the design criteria.

Chapter 8 provides a discussion of the results of the evaluation, detailing which design criteria are conformed to by the manipulator prototype. Figure 10.1 below summarises these findings, and compares the results with the analyses of the HANDY 1 and MANUS systems presented in Chapter 2[1].

	MANUS	HANDY 1	Middlesex
Cost	No	Yes	Yes
Functionality	Yes	No	Yes
Performance	Yes	Yes	Yes
Mobility	Yes	Yes	Yes
Input devices	?	No	Yes
Variety of control modes	?	No	Yes
Adaptability	?	No	Yes
Appearance	Yes	Yes	No
Safety	Yes	Yes	Yes
Design modifications	No	No	No

FIGURE 10.1 DESIGN CRITERIA CONFORMANCE COMPARISONS

Work is currently being undertaken to increase the control modes and adaptability of the MANUS arm, though as discussed in Chapter 2, this has on occasion reduced system functionality. The prominent issue with the MANUS arm is its high cost, and significant design changes would be required to address this. As the current design does not facilitate these modifications, the MANUS arm fails on both the cost and design modifications criteria. The HANDY 1 system is unlikely to achieve success as a general-purpose manipulator comparable to its success as a feeding aide. Without fundamental modifications to the HANDY 1's construction and kinematic configuration, its flexibility will not match that required by systems designed to be general-purpose.

[1] The two systems were selected as they have achieved levels user acceptance greater than most rehabilitation systems, and evaluations are widely reported .

The evaluation of the Middlesex Manipulator prototype demonstrated that functionality required of a general-purpose manipulator was provided. The research reported in this thesis resulted in a system design allowing adaptability at the user interface, and a low-cost control system. The prominent negative issue with the current prototype is poor physical appearance. Whilst the significance of this is reduced for a prototype, it is important that the design allows for the system to be evolved into a product capable of achieving user acceptance. This is not provided by the current design, and consequently the manipulator fails to conform with both the appearance and design modifications criteria.

Figure 10.1 shows a unique profile for each of the manipulators. Researchers within the field would dispute the exact contents of the table, however this thesis argues that the criteria that is clearly not conformed to by all three is the design modifications criteria. This tends to suggest that the design solutions adopted by the three projects, collectively demonstrate the feasibility of successful general-purpose rehabilitation robot design. However, as discussed in Chapter 1, the potential market for a rehabilitation robot is orders of magnitude greater than that currently tapped. It may therefore be concluded from Figure 10.1, that an indispensable attribute of any rehabilitation robotic system, is that it may be easily evolved to meet user requirements as both technology and user expectations progress.

The second objective of the thesis related to the process of configuring an adaptable user-interface to match specific user and device characteristics. As discussed in Chapter 9, the selection and installation of assistive technology relies predominantly on the subjective assessments of clinicians. It was argued that this process may be supported by the inclusion of objective measures, and that the measures of interest should be a combination of user's functional ability with the characteristics of the device of interest. This combination was defined within this thesis as the user's controlling ability. A methodology based on a form of task analysis was developed, that allows estimates of the relevant user characteristics to be included within a model of user interaction.

An experiment was undertaken, to determine the accuracy of usability predictions resulting from the model. Whilst task analysis can address error frequency, interface complexity and the functional completeness of a system, the experiment focused on predicting the speed with which an interface may be navigated. It was demonstrated that the model could consistently predict the relative usability of interface configurations for varying interaction styles and levels of controlling ability.

It was demonstrated that the process of producing usability predictions could be easily automated, and that the results could be used to assist clinicians and end users in selecting input devices and interaction styles for a specific user interface.

10.2 Future work

10.2.1 The manipulator and motor control system

A number of design modifications were identified in Chapter 8 as being necessary for user acceptance of the manipulator. Principal amongst these are the use of 'softer' materials where possible, and a softer or more rounded appearance. This would include covering all motors, gears and leads. An analysis of user tasks indicated a required increase in the speed of the two linear axes. The speed is currently limited by the unacceptable levels of noise generated, a problem that may be resolved if the use of alternative materials was investigated.

The user evaluation described in Chapter 8 employed semi–structured environments, in which objects were adapted to match the functionality of the manipulator. Increasing the manipulator's functionality, as required to undertake the entire set of user tasks identified, would require the inclusion of an appropriate three degree-of-freedom end-effector. A prototype end-effector was developed by an undergraduate student under the supervision of the author (Reynolds B., 1997). The project was successful in achieving the three degrees of freedom, and provides a detachable gripper unit, allowing for grippers to be changed to match task requirements. However, the weight of the prototype end-effector is excessive at around 1 kg, and requires to be reduced through the use of lighter motors than are currently employed, and perhaps alternative materials. Work towards a modified design is planned within the School of Engineering Systems at Middlesex University.

10.2.2 The user interface system

The adaptability inherent in the system's design allows and encourages the development of new forms of interaction. The evaluation reported in Chapter 8 employed a user interface in which control commands were organised into a menu-based system. This is referred to in Chapter 6 as version one of the user interface system. A second version has been developed that employs a Dialog-based graphical user interface. Investigations are required to allow a comparison between the two approaches for the various forms of user input available. A purpose-built LCD display unit has been developed by Donate (1996) under the supervision of the author, and remains to be tested.

Tele-operated control of the Middlesex Manipulator provides a natural interface for those users possessing an appropriate level of controlling ability. The current design of the user interface and control system allows for tele-operation, with the use of an 'intelligent' joystick that can convert joystick movement into the appropriate JUCL commands. A design that adopts this approach was implemented by (Silverio, 1996) under the author's supervision.

The tilt-sensors used for gesture classification exhibit a slow time response, limiting the number of gestures that can be easily generated. During the final stages of project development a two-axis solid-state tilt sensor was identified (Crossbow Technologies, USA), providing greater operating range (±75°) and a faster response than electrolytic tilt sensors. Use of the solid-state tilt sensor as a 'head - mouse' is currently being investigated by the author.

10.2.3 Choice of user interface and control system platform

Recent years have seen a significant reduction in the cost of processors and peripheral equipment as well as improvements in operating systems and software development tools. A continual re-assessment of the state and cost of technology is required to ensure that design solutions employ the appropriate platform for implementation. The user interface for version 1 was developed on a PC running Windows 3.11 at 100MHz. However, the intention was to port the system to an embedded platform, such as an embedded 486. The reduced cost of Pentium machines (industrial or desktop) suggests

that these would now be the appropriate platform for future work. This would allow for the use of the Windows NT or Windows 2000 operating system. These offer a true multi-tasking environment, which may be exploited to increase system safety. Separate processes or threads within an NT system may be used to replicate or replace tasks currently performed by the embedded micro controllers.

10.2.4 User Evaluation

A fundamental requirement for the development of any form of assistive technology is the involvement of potential users throughout the design process. To date, this has been achieved for the Middlesex Manipulator through user surveys, an analysis of the evaluations of existing systems, and a single-user case study of the current prototype. An important component of future work will be the resolution of issues raised within this thesis, and the exposure of a modified prototype to a wider number of potential end-users.

10.3 Concluding remarks

The main practical contribution of the work reported in this thesis, is the production of a working and testable manipulator prototype, from an inherited novel robot design. The lessons learnt from the evaluation will contribute to the pool of collective findings within the field of rehabilitation robotics, from which the original design objectives were drawn, and from which a successful production model will emerge.

Whilst modifications to the manipulator's appearance are required, it has been demonstrated that a purpose built control system based around a low-cost embedded microcontroller, provides adequate functionality for the performance of tasks prioritized by potential end-users. The necessity and benefits of providing an adaptable system was demonstrated, as was the ability to achieve this at low cost.

This thesis has demonstrated that the field of human-computer interaction has remained too isolated from assistive technology, and that techniques from the field can be adapted and successfully applied. It is hoped that such an approach will influence the development of future systems.

References

Anzai Y. (1994) "Human-robot-computer interaction: a new paradigm of research in robotics", Advanced Robotics, Vol. 8, No. 4, pp. 357 - 369.

Batavia A.I., Hammer G.S. (1990) "Toward the development of consumer-based criteria for the evaluation of assistive devices", Journal of Rehabilitation Research and Development, Vol. 27, No 4, pp. 425 - 436.

Birch, G.E. (1993) "Development and methodology for the formal evaluation of the Neil Squire Foundation robotic assistive appliance", Robotica, Vol. 11, pp. 529 - 534.

Birch G.E. Fengler M. Gosine R.G. Schroeder K. Schroeder M., Johnson D.L. (1996). An assessment methodology and its application to a vocational robotic assistive device, Technology and Disability, Vol. 5, No 2, pp. 151 - 165.

Bishop C. M. (1995) "Neural Networks for Pattern Recognition", Oxford University Press, 1995. ISBN 0-19-853864-2.

Bolmsjo, G., Topping M., Heck, H., Hedenborn, P., Olsson, M. (1997) "RAIL - Project status and technical developments", Advancement of Assistive Technology, Assistive Technology Research Series 3, Eds Anogianakis G., Buhler C., Soede M., IOS press, pp. 24 - 28. ISBN 90-5199-3617.

Brelivet, L. (1992) Telemanipulateur MANUS: Rapport d'evaluation, Association Francaise contre les Myopathies (AFM).

Buter S.P., Veltman R.J. (1996) "Modifications of the prototype of a robotic manipulator for disabled wheelchair users", ERASMUS Project Report, Middlesex University.

Card S. K., Moran T. P., Newell A. L. (1983), The Psychology of Human Computer Interaction, Hillsdale, NJ: Erlbaum. ISBN 0898 592 437.

Clay T.P., Hillman M.R. Orpwood R.D., Clarke A.K., (1987) "A survey of the potential disabled users of a robotic aid system", Royal National Hospital for Rheumatic diseases and Bath Institute for Medical Engineering.

Dallaway, J. Timmers P. (1995) "Rehabilitation robotics in Europe", IEEE Transactions on Rehabilitation Engineering. Vol. 3, pp. 35 - 45.

Danielson C., Holmberg L. (1994) "Evaluation of the RAID workstation", International Conference of Rehabilitation Robotics, Wilmington, Delaware, pp. 7 - 11.

Dario P., Guglielmelli E., Allotta B. (1995) "Mobile robots aid the disabled", Service Robot, Vol. 1, N1 1995, pp 14 - 18.

Demers L., Weiss-Lambrou R., Ska B. (1996) "Development of the Quebec user evaluation of satisfaction with assistive technology", Assistive Technology. Vol. 8, No 1, pp. 3 - 15.

Desurvire H. W., Kondziela J.M., Atwood M.E. (1993) "What is gained and lost when using evaluation methods other than empirical testing", NYNEX Science & Technology AI Laboratory, New York, pp. 90 - 101.

Dijkstra N., Fennema A. (1994) "Developing and manufacturing an electrical robot arm", ERASMUS Project Report, Middlesex University.

Donate, A., C. (1996) "Development of an LCD feedback device for use within a robotic manipulator control system", B.Eng. Honours degree in Electronic Engineering project report, Middlesex University, London.

Dowland B., Cipolla R., Clarkson J. (1997) "A prototype for interactive robotics application development", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 71 - 74.

Driessen, B.J.F., Woerden, J.A., Nelisse, M.W., Overboom, G.R. (1997) "Rehabilitation Robotic Concepts - integrating Manus on a mobile platform", Advancement of Assistive Technology, Assistive Technology Research Series 3, Eds Anogianakis G., Buhler C., Soede M., IOS press, pp. 15 - 19. ISBN 90 5199 361 7.

Dym C.L., Levitt R. E. (1991) "Knowledge based systems in Engineering", McGraw-Hill International Editions, ISBN 0-07-100850 - 0

Edwards A.D.N. (1995) "Extra-ordinary human-computer interaction", Cambridge series on human-computer interaction, Ed Edwards A.D.N., Cambridge Univ. Press. ISBN 0521 434 130.

Eftring H., Boschian K., (1999) "Technical Results from MANUS User Trials", ICORR '99: International Conference on Rehabilitation Robotics, Stanford, CA.

Erlandson, R. F., Sant, D., Wiadnyana, K., Rippy, J. Nizio, P. (1995) "Instrumentation of the Handy 1 for Oral-Motor Therapy", Rehabilitation Engineering Society of North America, 1995 conference proceedings.

Finlay P.A. (1988) Applications for Advanced Robotics in Medicine and Healthcare, Feasibility study report, No. R1175/1, January.

Franklin G.F., Powell J.D., (1981) Digital Control of Dynamic Systems, Addison-Wesley, ISBN 0-201-02891-3

Gellrich L., (1995) "Development of a control system for a wheelchair mounted robot arm", ERASMUS project report, Middlesex University, London.

Gelrich L. (1996) "Development of a gesture recognition system", A dissertation in partial fulfillment of the requirement for a Master of Science, Middlesex University, June 1996.

Guittet J., Kwee, H.H. Quetin, N., Yclon J. (1979), The Spartacus telethesis: manipulator control studies. Bull. Prosth. Res., BPR 10-13, 69-105.

Hagen, K., Hagan, S., Hillman, M., Jepson, J. "Design of a wheelchair-mounted robot", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 27 - 30.

Hammel M., Van der Loos, H.F.M. (1992) "Evaluation of a vocational robot with a quadriplegic employee", Arch Phys Med Rehabil. Vol. 73, pp. 683 - 693.

Harwin W.S., Jackson R.D. (1990) "Analysis of intentional head gestures to assist computer access by physically disabled people", J. Biomed Eng, pp. 193 - 198.

Heide K.D., Roorda G.J. (1993) "Design of an electrical robot arm", ERASMUS Project Report, Middlesex University, 1993.

Hillman, M. (1992), "Rehabilitation Robotics", Critical Reviews in Physical & Rehabilitation Medicine, Vol. 4, No. 1, pp. 79 - 103.

Hillman, M., Jepson, J. (1992) "Evaluation of a robotic workstation for the disabled", Journal of Biomedical Engineering, Vol. 14, pp. 187 - 192.

Hillman, M., Jepson, J. (1997) "Evaluation of a trolley mounted robot - a case study", International Conference of Rehabilitation Robotics, Bath University, UK. pp. 95 - 98

Hrycej T. (1991) "Back to Single layer learning principles", International Joint Conference on Neural Networks, Seattle, 1991.

Hush D.R., Horne B.G. (1993) "Progress in supervised neural networks", IEEE Signal Processing Magazine, January 1993, pp. 8 - 39.

Johnson P., (1992) "Human Computer Interaction - Psychology, Task Analysis and Software Engineering", McGraw-Hill, ISBN 0-07-707235-9

Kassler, M. (1993) "Introduction to the special issue on robotics for health care", Robotica, Vol. 11, pp. 493 - 494.

Keates S., Potter R., Perricos C., Robinson P. (1997) "Gesture recognition - research and clinical perspectives", RESNA 97, pp. 333 - 335.

Keates S., Robinson P. (1997) "The role of user modelling in rehabilitation robotics", International Conference of Rehabilitation Robotics, Bath University, UK pp. 75 - 78.

Kieras D.E., and Polson P. (1985) "An approach to the formal analyses of user complexity", International Journal of Man-Machine Studies, Vol. 22, pp. 365 - 394.

Kieras D.E. (1988) "Towards a practical GOMS model methodology for user interface design", Handbook of Human Computer Interaction, Amsterdam: Elsevier. ISBN 0444 705 368.

Kintsch W. (1988) "The use of knowledge in discourse processing : a construction integration model", Psychological review, Vol. 95.

Kwee H. H., Duimel J. J. (1988) The MANUS Wheelchair-Bourne Manipulator: Developments Towards a Production Model, Proceedings of the International Conference of the Association for the Advancement of Rehabilitation Technology. Pp. 440 - 461.

Kwee H.H., Duimel J.J. (1989), The Manus Wheelchair-Borne Manipulator: System Review and First Results, The 2nd Workshop on Medical and Healthcare Robotics, Newcastle Upon Tyne, UK, pp. 1 -11.

Kwee H.H., Cremers G.B., van der Pijl D.J., Aartsen H.A. (1994) "User evaluation of an M3S demonstration platform", International Conference of Rehabilitation Robotics, Wilmington, Delaware, pp. 3 - 5.

MacIntyre, F., Estep K. W., Siebert, J. M. (1990) Cost of User Friendly Programming, Journal of Forth Application and Research, Vol. 6, No. 2, pp. 103 - 115.

Mahoney R., M., Dalloway J. L., Jackson R. D. (1992) "Development of the robot control language - CURL", International Conference of Rehabilitation Robotics, Keele University, Staffordshire, UK, September 1992.

Mahoney, R. M. (1997) "Robotic products for rehabilitation: status and strategy", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 12 - 17.

Makino, H. and Furuya, N. (1982) "SCARA robot and its family", Proceedings of the 3rd International Conference on Assembly Automation, IFS Publications Ltd, pp. 433 - 444.

Martin, J. Meltzer, H. and Elliot, D. (1988) 'The Prevalence of Disability amongst Adults', Office of Population Censuses and Surveys, Social Survey Division, H.M.S.O.

Mattie J., Hannah R. (1994) "Evaluation of the Inventaid Manipulator Arm for a youth with muscular dystrophy", International Conference of Rehabilitation Robotics, Wilmington, Delaware, pp. 13 - 17.

Mattie J., Hannah R. (1995) " Development of an evaluation procedure for wheelchair-mounted manipulator arms", Conference of the Rehabilitation Engineering Society of North America, pp. 499 - 501.

McEachern W., Perricos C. Jackson R., (1994) "Head gesture assisted direct control of a rehabilitation manipulation system", ICORR 94, pp. 49 - 54.

Middendorf, W.H. (1986) "Decisions", Design of Devices and Systems, New York, Dekker, pp.198 - 230.

Milner, M., Naumann, S., King, A. and Verburg, G. (1992) Evaluation of Manus Manipulator Arm in ADL, Vocational and School Settings. Final report to National Health Research and Development Program, Project #6606-4198-59.

Moran T.P. (1981) "The Command Language Grammar: a representation for the user interface of interactive computing systems", International Journal of Man-Machine Studies, Vol. 15, pp. 3 - 50.

Nielsen J. (1992)."Finding usability problems through heuristic evaluation", Human Factors in Computing Systems, CHI 92, pp. 373 - 378.

Nielsen J., Phillips V. (1993) "Estimating the Relative Usability of Two Interfaces: Heuristic, Formal, and Empirical Methods Compared", Human Factors in Computing Systems, Conference Proceedings, pp. 214 - 221.

Neilson J. (1994) "Usability Inspection Methods", John Wiley & sons, 1994, pp 5 -6. ISBN 04710 187 75.

Newell A., Simon H. (1972) "Human Problem Solving", Englewood Cliffs, NJ, Prentice Hall.

Norman D., Draper S., (1986) "User centered system design", Lawrence Erlbaum Associates Inc. ISBN 0898 597 811.

Oderud, T., Bastiansen, J.E. (1992) Integrating a Manus manipulator and an electric Wheelchair. Practical experiences. Proc 15th RESNA Conference, Toronto, pp. 595 - 597.

Oderud, T. (1997) "Experiences from the Evaluation of a Manus Wheelchair-mounted Manipulator", Advancement of Assistive Technology, Assistive Technology Research Series 3, Eds Anogianakis G., Buhler C., Soede M. IOS press, pp. 20 - 23. ISBN 90 5199 361 7.

Overboom G., Nelisse M., van Woerden K. (1997) "Focus on the Central position of Users in integrated Systems", Advancement of Assistive Technology, Assistive Technology Research Series 3, Eds Anogianakis G., Buhler C., Soede M., IOS press, pp. 303 - 313. ISBN 90 5199 361 7.

Parsons B.N., Gellrich L., Warner P.R., Gill R., White A.S. (1996) "Application of a Gesture Classification System to the Control of a Rehabilitation Robotic Manipulator", IEEE Conference on Engineering in Medicine and Biology, Amsterdam. ISBN 90 9010005 9 (CD ROM).

Parsons B., Warner P.R., White A.S., Gill R. (1997a) "An Adaptable User Interface and Controller for a Rehabilitation Robotic Arm", International Conference on Advanced Robotics, California, pp. 919 - 923.

Parsons B.N., Warner P.R., White A., Gill R. (1997b) "Initial Evaluation of the Middlesex Rehabilitation Robotic Arm", RESNA Conference, Pittsburgh, pp. 411 - 413.

Parsons B., Warner P., White A.S., Gill R. (1997c) "An approach to the development of adaptable manipulator controller software", International Conference of Rehabilitation Robotics, Bath University, UK. pp. 67 - 70.

Payne S.J., Green T.R.G., (1986) "Task-Action Grammars : A model of the Mental Representation of Task Languages", Human Computer Interaction, Vol. 2, pp. 93 - 133.

Payne S. J., and Green T.R.G. (1989) "The structure of command languages: An experiment on Task Action Grammar, International Journal of Man-Machine Studies, Vol. 30, pp. 213-234.

Polson P., Lewis C. "Cognitive Walkthroughs: a method for theory based evaluation of user interfaces", Int. J. Man-Machine Studies, Vol. 36, pp. 741 - 773.

Poulson D., Ashby M., Richardson S. (1996) "User fit : a practical handbook on user-centred design for assistive technology", TIDE, ECSC-EC-EAEC, 1995.

Prior S.D. (1989) "A review of world rehabilitation research", Internal Report, Middlesex University, London.

Prior S.D. (1990) "An electric wheelchair-mounted robotic arm - a survey of potential users", Journal of Medical Engineering & Technology, Vol. 14, No 4, pp. 143 - 154.

Prior S.D., Warner P.R., Parsons J.T., White A.S., Oettinger P. (1992) "A hybrid rehabilitation robotic arm for the physically disabled electric wheelchair user", International Conference of Rehabilitation Robotics, Keele University, Staffordshire, UK, September 1992.

Prior S.D. (1993) "Investigations into the design of a wheelchair-mounted rehabilitation robotic manipulator", PhD thesis, Middlesex University, 1993.

Reisner P., (1981) "Formal grammars and Human Factors design of an interactive graphics system", IEEE Transactions of Software Engineering Systems, Vol. 5, pp. 229 -240.

Reynolds B. (1997) " The design of a three degree of freedom end-effector for the Middlesex Manipulator", B.Eng. Honours degree in mechanical engineering project report, Middlesex University, London.

Rubio M., M., (1996) " Development of a microcontroller-based control system for a wheelchair mounted robot arm", B.Eng. Honours degree in electronic engineering project report, Middlesex University, 1996.

Silverio, J. P. (1996) "Intelligent Joystick for a robotic manipulator", B.Eng. Honours degree in electronic engineering project report, Middlesex University, London.

Sheredos S. J., Taylor B., Cobb C., Dann E. E. (1996) "Preliminary evaluation of the helping hand electro-mechanical arm", Technology and Disability, Vol. 5, pp. 229 - 232

Sheredos S. J., Taylor B. (1997) "Clinical evaluation of the helping hand electro-mechanical arm", RESNA 97, Pittsburgh, USA, pp. 378 - 380.

Smith S.L. Mosier J.N. (1984) Design guidelines for the user interface for computer based information systems. Bedford MA: The Mitre Corp.

Smith, J., Topping, M. (1997) "Study to determine the main factors leading to the overall success of the Handy 1 robotic system", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 147 - 150.

Stanger C.A., Annezio C. P., Cawley M.F. (1996) I"Range of Head Motion and Force of High Cervical Spinal Cord Injured Individuals for the Design of a Test-bed Robotic System", ICORR'94 Proceedings of the International Conference on Rehabilitation Robotics, Wilmington, Delaware, USA, pp. 37 - 42.

Stuyt, H. J. A. (1997) "Manus in Europe", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 111 - 115.

Tew A.I., Gray C. J., "A real-time gesture recognizer", Journal of Biomedical Engineering, Vol. 15, pp. 181 - 187.

Topping, M. (1993) "Early experiences in the use of the Handy 1 robotic aid to eating", Robotica, Vol. 11, pp. 525 - 527.

Topping M. (1995) "The Development of HANDY 1 a robotic aid to independence for the severely disabled", IEE Colloquium on Mechatronic Aids for the Disabled, University of Dundee, UK, pp. 1 - 6. Digest No 1995/107.

Topping, M. (1996) "Handy 1, a robotic aid to independence for severely disabled people", Technology and Disability, Vol. 5, pp. 233 - 234.

Topping, M., Heck H., Bolmsjo G. (1997) "An overview of the BIOMED 2 RAIL project", International Conference of Rehabilitation Robotics, Bath University, UK, pp. 23 - 26.

Van der Loos, H.F.M., Hammel J., M. (1990), "Designing rehabilitation robots as office and household equipment", International Conference of Rehabilitation Robotics, Wilmington, Delaware, USA, pp. 121 - 135.

Verburg G., Kwee H., Wisaksana A., Cheetham A., va Woerden J. (1995) "Manus : The evolution of an assistive technology", Technology and Disability, Vol. 5, pp. 217 - 228.

Wasserman P.D. (1989) "Neural computing - theory and practice", Van Nostrand Reinhold, New York. ISBN 0 442 20743 3.

List of media included with thesis

The following items have been produced to supplement the written thesis:

Software (CD ROM)

Source code for motor control software.
Source code for user interface system.

Video

Video footage of the initial evaluation of the Middlesex Manipulator.

Appendix A

Juvo Motor Control Language Opcode Summary

This appendix provides a summary of the Juvo Motor Control Language (JMCL) v 1.0. Opcodes. JMCL defines the command set between a user interface system and a motor control system used to control the Middlesex manipulator.

BRK - sets motor brake for all axes
ERM - indicates motor brake set
ACK - acknowledge
CAN - cancel dialogue
ERT - error in transmission
HLT - stop all axes
Hn - stop axis n
Sk - set max speed for axis k
Vn - set speed of axis n to value passed in next byte
Mnd - move axis n in direction d
Pn - move axis n to absolute position specified by next 2 bytes
WIn - transmit 2 bytes containing position of axis n
RST - reset motor brakes
NXT - request next byte
Lnd - limit of axis n in direction d encountered.
where : $0 \leq n \leq 7$, $0 \leq d \leq 1$, and $0 \leq k \leq 31$.

Appendix B

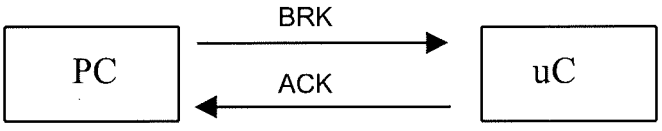
Juvo Motor Control Language Protocol

This appendix provides a summary of the Juvo Motor Control Language (JMCL) v 1.0. JMCL defines the communication protocol between a user interface system and a motor control system used to control the Middlesex manipulator. The protocol defines communication over a serial link between a personal computer (PC) and an embedded microcontroller (uC), at a level above RS232. Thus RS232 handshaking, baud rate and data formats are not defined.

Command descriptions

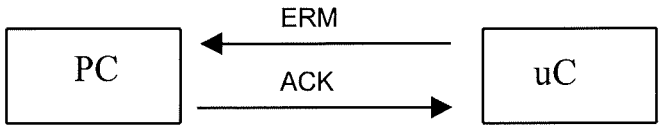
BRK

The break command (BRK) is used to bring the manipulator's motors to an immediate stop, causing all motor drive signals to be set to zero. As the Halt command defined below allows the motors to be stopped less abruptly, the break command should be reserved for emergency scenarios. The motor control system responds to successful execution of the Brk command with an acknowledge command (ACK).



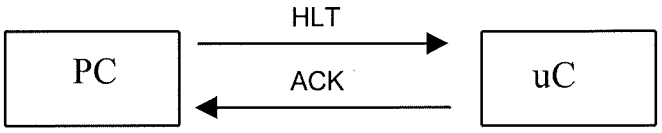
ERM

The motor error command (ERM) is issued by the motor control system to indicate that an error occurred causing motor drive circuitry to be disabled. Typically caused by current limits being exceeded.



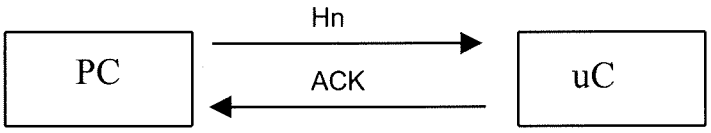
HLT

The halt command (HLT) brings all motors that are currently in motion to a halt, by setting the axes' target positions as the current positions plus a pre-defined constant value.



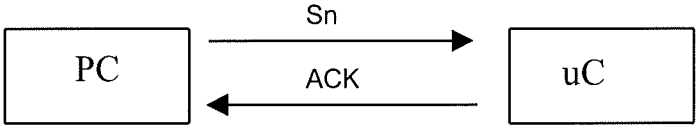
Hn

The halt axis n command (Hn) brings axis n to a halt by setting the target position as described above. As n may have a value from 1 to 8, Hn actually represents a set of command with 8 consecutively numbered opcodes (see opcode listing at end of appendix).



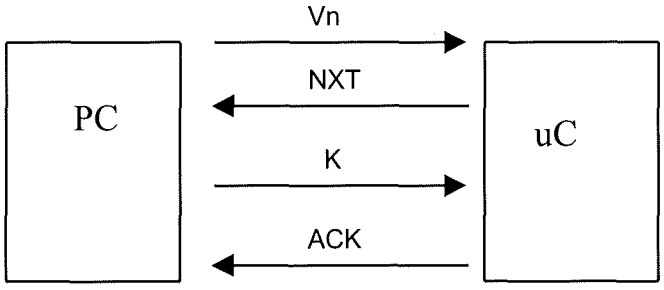
Sn

The Sn command (Sn) sets all axes to speed level n. Eight speed levels are selectable, with the actual motor speeds for each axis configurable at motor controller software and hardware levels.



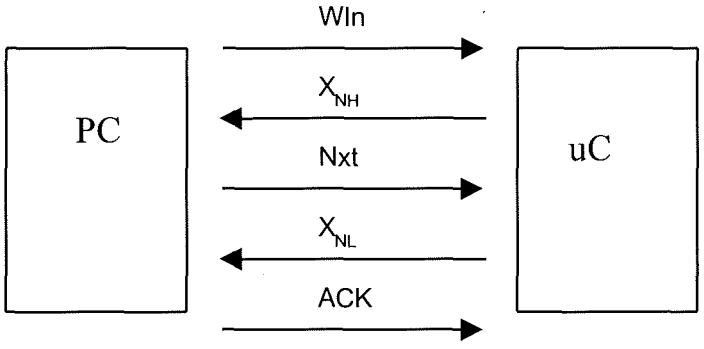
Vn

The velocity of axis n command (Vn) sets the speed of axis n to a percentage of the maximum axis speed. The command is used for fine speed control, and consists of two transmissions to the microcontroller. The first indicates that speed is being set, and which axis is selected. As n can be from one to eight, there are eight actual opcodes for the Vn command. The motor controller responds with the next command (Nxt) which acts as request for the speed value. This value k, is then transmitted by to the microcontroller as one byte, thus 256 speed levels are selectable for each axis. This translates to a selection from 0 to 100% of the hardware setting, with a resolution of 0.4%.



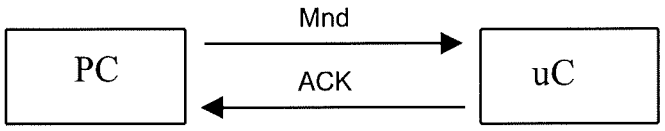
WIn

The where is axis n command (Win) is used to determine the position of axis n, where n can be from one to eight. Receipt of the command by the microcontroller causes transmission of the absolute position of the selected axis in two bytes. The most significant byte is first transmitted, followed by receipt of the Nxt command, after which the least significant byte is transmitted.



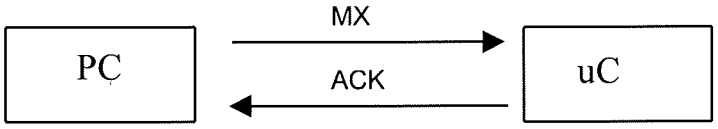
Mnd

The move axis n in direction d command (Mnd) is used to move a selected axis in one of two directions. As n can take on a value of from one to eight, and n can take on a value of one or two, Mnd consists of a set of sixteen consecutive opcodes.



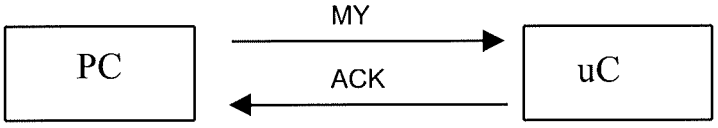
Mx

The move in direction x command (Mx) is used to move the manipulator's end-effector in the x plane in one of 2 directions. Thus x may take on a value of one or two.



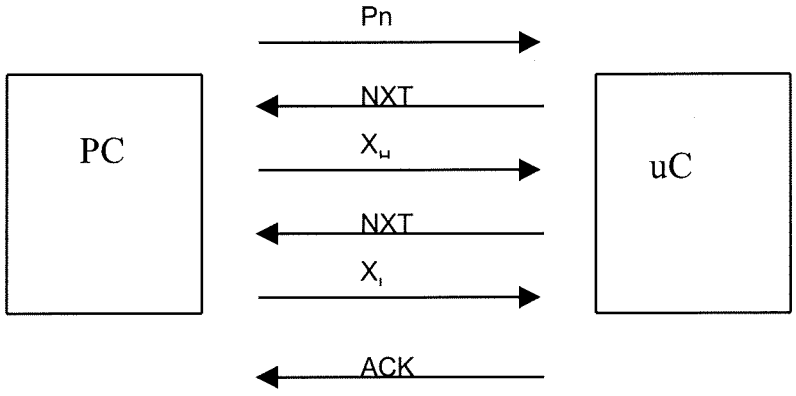
Mx

The move in direction y command (My) is used to move the manipulator's end-effector in the y plane in one of 2 directions. Thus y may take on a value of one or two.



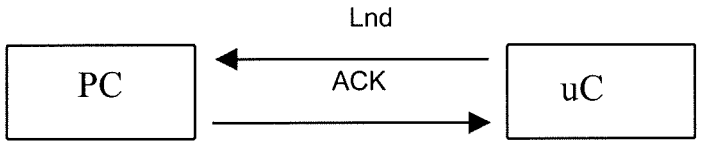
Pn

The move axis n to an absolute position (Pn) is used to move a selected axis to a position encoded in two bytes. On receipt of the Pn command, the microcontroller requests the two bytes by issuing the Nxt command.



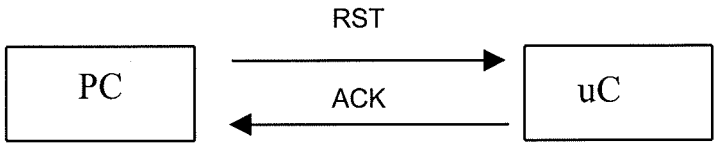
Lnd

The Lnd command is used to indicate that the limit of axis n in direction d was reached. This may be caused by either a software or hardware limit being exceeded, and would have resulted in the microcontroller removing drive signals from all axes (achieved applying the motor brake).



RST

The reset command is used to reset all motor axes. This removes the effect of any previously applied brake signals, and allows a drive signal to be applied to the motors.



Appendix C

Motor Control Code Listings

The following appendix provides listings of the 8051 motor control code written in the C programming language. The listings include:

1 MCMain.c	Main source file containing top level code
2 IO.c	IO routines
3 Serial.c	Serial comms routines
4 Control.h	Hardware dependent constant definitions, Gain constants Global variable definitions


```
The following appendix provides listings of the 8051 motor control code
written in the C programming language. The listings include:

1 MCMain.c  Main source file containing top level code
2 IO.c      IO routines
3 Serial.c  Serial comms routines
4 Control.h Hardware dependent constant definitions, Gain constants
            Global variable definitions
```

1 MCMain.c

```
////////////////////////////////////////////////////////////////////////////
// Headers for functions not defined here
#include "serial.h"
#include "funcs.h"
#include "jmcl.h"

// BRAKECHECKDELAY is used to determine how frequently we check if
// a brake has been set
#define BRAKECHECKDELAY 100
#ifndef FALSE
        #define FALSE 0
        #define TRUE !FALSE
#endif

// Number of axes is 6 till we get a 3 d.o.f.  end-effector
#ifndef NUMBER_OF_AXIS
        #define NUMBER_OF_AXIS 6
#endif

// Variables defined as extern for accessibility in linked modules
// Not exactly structured programming, but faster

// Motor filter values
extern float MotorAlpha;
extern float MotorBeta;

// current axis
extern char Axis;

// Arrays hold current and target positions, error is the dfference
extern int  CurrentPos[];
extern int  TargetPos[];
extern int  Error[];

// These act as software limits for each axis
extern unsigned char  MinPos[];
extern unsigned char  MaxPos[];



// Dynamic error holds acceptable error values (dynamic)
extern unsigned char  DynamicError[];

// Hlt band holds acceptable error values (static)
extern unsigned char  HltBand[];

// Reaced elements set to true for each axis when error acceptable
extern char Reached[] ;

// used to record drive signals, so new drive signals can be increased
// at an acceptable rate
extern float PrevSignal[];

// Current user selected speed setting
extern unsigned char SpeedSetting;

// If there is new user input, the following variables are set to true,
// command name, and arguments respectively
static char NewInput = TRUE;
static int UserInput = BRK;
static int UserInput2;
static int UserInput3;

// Following variables determine whether the brake can be set, how long since
// we last checked, whether there's a forced brake or whether the brake is set
static char CanSetBrake = FALSE;
static char BrakeCheckDelay = BRAKECHECKDELAY;
```

```
static char ForcedBrake = TRUE;
static char BrakeSet = TRUE;




////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
main()
{

//////////////////////////////////////////////////////////
//   Perform hardware initialization, and ensure brake is set

        InitSerialPort();
        ConfigTimerIc();
        InitPort1();
        SetBrakeOn();

//////////////////////////////////////////////////////////
//      Set initial positions to initial target positions

        ReadCurrentPos();
        for (Axis = 0;Axis < NUMBER_OF_AXIS; Axis ++)
                TargetPos[Axis] = CurrentPos[Axis];



//////////////////////////////////////////////////////////
//      Perform main program loop for ever

        do
        {
                //       Check for user input
                if (CharReady())
                {
                        UserInput = SerialIn();
                        NewInput = TRUE;
                }

                //  If three is input, process it
                if (NewInput)
                {
                        switch (UserInput)
                        {
                        // If user has selected brake, set brake and initialize all
                        // associated variables appropriately
                        case BRK :  SetBrakeOn();
                                        ForcedBrake = TRUE;
                                        BrakeSet = TRUE;
                                        SetZeroSpeeds();
                                        for (Axis = 0; Axis < NUMBER_OF_AXIS; Axis++)
                                        {
                                        Reached[Axis] = TRUE;
                                                PrevSignal[Axis] = 0;
                                                TargetPos[Axis] = CurrentPos[Axis];
                                        }
                        // use goto to quit - horrible but quick
                                        goto  EndOfNewInput;

                        //  If halt, add a predefined constant to current position
                        case HLT :      for (Axis = 0;Axis < NUMBER_OF_AXIS; Axis ++)
                                         if(!Reached[Axis])
                                          if(Error[Axis]>0)
                                                TargetPos[Axis] = CurrentPos[Axis] + HltBand[Axis];
                                          else
                                                TargetPos[Axis] = CurrentPos[Axis] - HltBand[Axis];
```

```
                                goto  EndOfNewInput;

                    // If reset, release brakes and set appropriate flags
        case RST :  SetZeroSpeeds();
                                Reset();
                                BrakeSet = FALSE;
                                ForcedBrake = FALSE;
                                goto  EndOfNewInput;
                    }// end of switch


//      The remaining commands can't easily be checked for within a switch, as
//      they consist of ranges of values.
        //      If Win, send data from the CurrentPos array
                if ( (WI0 <= UserInput) && (UserInput <= WI9))
                {
                        SerialWordOut(CurrentPos[UserInput - WI0]);
                        goto  EndOfNewInput;
                }

        //      If Mnd, work out which axis, then set the target as the min or max
        //      for that axis
                if ( (M00 <= UserInput)  && (UserInput <= M91))
                {
                        UserInput -= M00;
                        Axis = UserInput / 2 ;
                        if ( UserInput%2 == 0)
                                TargetPos[Axis] = MinPos[Axis];
                        else
                                TargetPos[Axis] = MaxPos[Axis];

                        Reached[Axis] = FALSE;
                        BrakeCheckDelay = 0;
                        goto  EndOfNewInput;
                }

        //      If Speed, Work out which level, the set corresponding variable
                if ( (S00 <= UserInput)  && (UserInput <= S31))
                {
                        if(UserInput <= S08)
                                SpeedSetting = 0x00;
                        else if(UserInput <= S15)
                                SpeedSetting = 0x01;
                        else
                                SpeedSetting = 0x02;
                        goto  EndOfNewInput;
                }

//      If Pnn, work out which axis, set the target position, then check this does
//      not exceed the axis limit
                if ( (UserInput >= P00) && (UserInput <= P09) )
                        if ( (UserInput2 = GetNextByte()) != ERROR )
                {
                          if ( (UserInput3 = GetNextByte()) != ERROR )
                          {
                                Axis = UserInput - P00;
                                TargetPos[Axis] = UserInput2 << 4;
                        TargetPos[Axis] += UserInput3;
                                if (TargetPos[Axis] > MaxPos[Axis])
                                        TargetPos[Axis] = MaxPos[Axis];
                                if (TargetPos[Axis] < MinPos[Axis])
                                        TargetPos[Axis] = MinPos[Axis];
                                Reached[Axis] = FALSE;
                                BrakeCheckDelay = 0;
                          }
                }
EndOfNewInput :         NewInput = FALSE;
                }    /*   End of new input service  */
```

```
//        Perform check to see which motors need drive signals applied or adjusting
              ReadCurrentPos();

              if( !ForcedBrake    )
              {
                       //     Periodic Check  For Brake set, set a flag if all targets reached
                      if(!(BrakeCheckDelay--))
                      {
                              BrakeCheckDelay = BRAKECHECKDELAY;
                              CanSetBrake = TRUE;
                              for(Axis = 0 ; Axis < NUMBER_OF_AXIS; Axis++)
                                      if(!Reached[Axis])
                                      {
                                              CanSetBrake = FALSE;
                                              break;
                                       }
                              if(CanSetBrake)
                              {
                                      SetBrakeOn();
                                      BrakeSet = TRUE;
                              }
                              else    if (BrakeSet)
                                      Reset();
                      }


                      //   Check for axis move per Axis
                      for ( Axis = 0; Axis < NUMBER_OF_AXIS; Axis ++)
                      {
                      /*  Here we call the move function if the target has not been reached
                              and the current error is greater than the dynamic error    */

                              if ( Mag(Error[Axis]) > DynamicError[Axis] )
                              {
                                      Move(Axis);
                                      Reached[Axis] = FALSE;
                              }

                              else    //             Otherwis classify axis as reached
                              {
                                      Reached[Axis] = TRUE;
                                      OutputDriveSignal( Axis, 0);
                                      PrevSignal[Axis] = 0;
                              }
                      }   //   End of each axis check

               }              //             End of if !Forced Brake

       }while(TRUE == TRUE);  // end of main program loop
}
```

2 IO.C

```
////////////////////////////////////////////////////////////////////////////
//      definitions of lower level input/output functions called by MCMain

#include "controlf.h"
#include "funcs.h"
#include "io51.h"

//  Global variables (see MCMain)
extern unsigned char  ZeroSpeedOffset[];
extern float MotorAlpha;
extern float MotorBeta;
extern char Axis;
extern int  CurrentPos[];
extern int  TargetPos[];
extern int  Error[];
extern unsigned char  MinPos[];
extern unsigned char  MaxPos[];
extern unsigned char  DynamicError[];
extern unsigned char  HltBand[];
extern char Reached[] ;
extern float PrevSignal[];
extern unsigned char SpeedSetting;

//  A to D filter variables
int Sample, LastSample;
unsigned char SampleCount;

//  Function populates CurrentPos array with current axis positions
void ReadCurrentPos()
{
//      Enable A/D
        for(Axis = 0; Axis < NUMBER_OF_AXIS - 1; Axis++)
        {
                // Read value for current axis, use as initial value for filter
                SampleCount = DATA_SAMPLE_LENGTH;
                SetADAddress(Axis) ;
                StartConv();
                StopConv();
                LastSample = read_XDATA(MSB_ADDR);

                // Read another samplecount values and filter
                while(SampleCount--)
                {
                        StartConv();
                        StopConv();
                        Sample = read_XDATA(MSB_ADDR);
                        LastSample *= AD_BETA;
                        LastSample += Sample;
                        LastSample /= 10;
                }
                // Set current pos and target array values
                CurrentPos[Axis] = LastSample;
                Error[Axis] = TargetPos[Axis] - CurrentPos[Axis];
        }
}
```

```
////////////////////////////////////////////////////////////
// Function Calculates required drive signal, and applies to axis
void Move(char Axis)
{
        float DriveSignal;


         /*     Calculate New Drive Signal  */
         /*     Error varies between -255 & +255, thus possible drive signal
                        varies between -127 and +127 for gain = 0.5    */

         /*  Multiply error by gain Kq or Kp dependent on direction  */
         /*  If error is within target band, multiply signal by another constant
         to increase rate of deceleration  */

         if(Error[Axis]>0)
         {
                        DriveSignal = (float ) Error[Axis] * Kp[Axis];
                         /*     Filter Output if Accelerating*/
                        if((DriveSignal>PrevSignal[Axis]) && (Error[Axis] >5))
                        {
                                DriveSignal *= MotorAlpha;
                                DriveSignal += ((float )PrevSignal[Axis] * MotorBeta);
                        }
                        else if (DriveSignal < PosMin[Axis])
                        {
                                DriveSignal = PosMin[Axis];
                                goto FuncEnd;
                        }
          }
          else if (Error[Axis]<0)
          {
                        DriveSignal = (float ) Error[Axis] * Kq[Axis];
                         /*     Filter Output if Accelerating*/
                        if((DriveSignal<PrevSignal[Axis]) && (Error[Axis] < 5))
                        {
                                DriveSignal *= MotorAlpha;
                                DriveSignal += ((float ) PrevSignal[Axis] * MotorBeta);
                        }
                        else if (DriveSignal > -1 * NegMin[Axis])
                        {
                                DriveSignal = -1 * NegMin[Axis];
                                goto FuncEnd;
                        }
          }

        // Check max signals not exceeded
         if(DriveSignal > MaxSig[Axis][SpeedSetting])
                DriveSignal = MaxSig[Axis][SpeedSetting];
         else if(DriveSignal < (-1*MinSig[Axis][SpeedSetting]))
                DriveSignal = (-1*MinSig[Axis][SpeedSetting]);

        //      Store signals & output
FuncEnd :
         PrevSignal[Axis] = DriveSignal;
         OutputDriveSignal( Axis, PrevSignal[Axis]);
}
```

```
////////////////////////////////////////////////////////////////
//      Function sets drive signal output

void OutputDriveSignal( char Axis, int IDriveSignal)
{
        /*      IDriveSignal varies between -32 & 32 and should be
                mapped to 0 to 64 +- offset for current axis.
                result is then assigned to char for output  */

char DriveSignal;
        IDriveSignal+=ZeroSpeedOffset[Axis];

        DriveSignal = IDriveSignal;

        switch (Axis)
        {
                case 0x00       :       write_XDATA(COUNTER_1, DriveSignal);
                                                break;
                case 0x01       :       write_XDATA(COUNTER_2, DriveSignal);
                                                break;
                case 0x02       :       write_XDATA(COUNTER_3, DriveSignal);
                                                break;
                case 0x03       :       write_XDATA(COUNTER_4, DriveSignal);
                                                break;
                case 0x04       :       write_XDATA(COUNTER_5, DriveSignal);
                                                break;
                case 0x05       :       write_XDATA(COUNTER_6, DriveSignal);
                                                break;
         }
}

////////////////////////////////////////////////////////////////
//      Initialize 8254 timer Ics for waveform generation

void ConfigTimerIc()            /*    Configure Timer IC      */
{
        write_XDATA(CTRL_ADDR_1_3,CTRL_WRD_1);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_1_3,CTRL_WRD_2);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_1_3,CTRL_WRD_3);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_4_6,CTRL_WRD_4);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_4_6,CTRL_WRD_5);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_4_6,CTRL_WRD_6);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_7_9,CTRL_WRD_7);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_7_9,CTRL_WRD_8);
        Delay(IO_DELAY);
        write_XDATA(CTRL_ADDR_7_9,CTRL_WRD_9);

}
```

```
//////////////////////////////////////////////////////////
//      Write initial zero speed PWM values

void SetZeroSpeeds()
{
        write_XDATA(COUNTER_1,STOP0);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_2,STOP1);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_3,STOP2);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_4,STOP3);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_5,STOP4);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_6,STOP5);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_7,STOP6);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_8,STOP7);
        Delay(IO_DELAY);
        write_XDATA(COUNTER_9,STOP8);
        Delay(IO_DELAY);
}

//////////////////////////////////////////////////////////
// Ensure brake set, and A/D conversion stopped

void InitPort1()
{
        SetBrakeOn();
        StopConv();
        Delay(IO_DELAY);

}

//////////////////////////////////////////////////////////
//  Following functions call set bit to control brake, and A/D conversion
void Reset()
{
        SetBrakeOff();
        clear_bit(P1_4_bit);
        Delay(RESET_PULSE_WIDTH);
        set_bit(P1_4_bit);
}

void SetBrakeOn()
{
                clear_bit(P1_7_bit);
}

void SetBrakeOff()
{
                set_bit(P1_7_bit);
}


void StartConv()
{
        clear_bit(P1_0_bit);
}

void StopConv()
{
        set_bit(P1_0_bit);
}

void SetADAddress(char Axis)
{
        clear_bit(P1_1_bit);
        clear_bit(P1_2_bit);
        clear_bit(P1_3_bit);
```

```
        switch(Axis)
        {
                case    0x01 :   set_bit(P1_1_bit);
                                                        break;
                case    0x02 :   set_bit(P1_2_bit);
                                                        break;
                case    0x03 :   set_bit(P1_1_bit);
                                                        set_bit(P1_2_bit);
                                                        break;
                case    0x04 : set_bit(P1_3_bit);
                                                        break;
                case    0x05 : set_bit(P1_3_bit);
                                                        set_bit(P1_1_bit);
                                                        break;
                case    0x06 : set_bit(P1_3_bit);
                                                        Delay(IO_DELAY);
                                                        set_bit(P1_2_bit);
                                                        break;
                case    0x07 : set_bit(P1_3_bit);
                                                        Delay(IO_DELAY);
                                                        set_bit(P1_2_bit);
                                                        Delay(IO_DELAY);
                                                        set_bit(P1_1_bit);
                 }

}

void Delay(int t)
{
        while(t--);
}
```

3 Serial.c

```
////////////////////////////////////////////////////////////
// The following routines are called by MCMain to provide serial I/O
// through the 8051 serial port

#include "serial52.h"
#include "io51.h"
#include "jmcl.h"

static int ser_wait;
static long LongWait;
static char data_in;

//      Set baud rate and serial mode
void InitSerialPort()
{
        output(SCON, SERIAL_MODE_1);
        output(TMOD, TIMER_MODE_2);
        output(TH1, TIMER_COUNT);
        set_bit(TR1_bit);
}

// Read a single character
char SerialIn()
{
                clear_bit(RI_bit);
                return(input(SBUF));
}

// Check for character ready
BOOL CharReady()
{
        return(read_bit(RI_bit));
}

//      Check if buffer clear for transmit
BOOL ReadyToTransmit()
{
        return(read_bit(TI_bit));
}

// Send a single character
void Transmit(char data)
{
        while(!ReadyToTransmit());
                /*   wait    */
                        clear_bit(TI_bit);

                        output(SBUF,data);
}
```

```
//      Send a word (2 characters
int SerialWordOut(int Word)
{
        int temp;
        char msb, lsb;

        temp = Word >> 4;
        msb = temp;
        lsb = Word & 0x0F;

        ser_wait = 0;
        LongWait = 100 * SER_TIME_OUT;
        Transmit(msb);

        while(!CharReady() && (ser_wait < LongWait))
                        ser_wait ++;

        if ( ser_wait < LongWait)
        {
                        data_in = SerialIn();
                        if(data_in == NXT)
                                        Transmit(lsb);
                        else
                        {
                                        Transmit(ERS);
                                        return ERROR;
                        }
        }
        else
        {
                        Transmit(ERT);
                        return ERROR;
        }
 }

// Solicit a byte by first transmitting the Nxt command
 int GetNextByte()
 {
        ser_wait = 0;
        LongWait = 100 * SER_TIME_OUT;

        Transmit(NXT);
        while(!CharReady() && (ser_wait < LongWait))
                                ser_wait ++;

        if ( ser_wait < LongWait)
        {
                        data_in = SerialIn();
                        return data_in;
        }
        else
                        return ERROR;
 }
```

```
//   Transmit error command
void Ers()
{
        Transmit(ERS);
}
```

4 Control.H

```
/*              Timer IC Address + Control Words      */

#define _CONTROLF_H

#ifndef FALSE
        #define FALSE 0
        #define TRUE !FALSE
#endif

#ifndef NUMBER_OF_AXIS
        #define NUMBER_OF_AXIS 6
#endif

#define CTRL_ADDR_1_3  0x2003          /* address conltrol-word 1-3 */
#define CTRL_ADDR_4_6  0x4003          /* address conltrol-word 4-6 */
#define CTRL_ADDR_7_9  0x6003          /* address conltrol-word 7-9 */
#define CTRL_WRD_1 0x12                /* define control-word_1     */
#define CTRL_WRD_2 0x52                /* define control-word_2     */
#define CTRL_WRD_3 0x92                /* define control-word_3     */
#define CTRL_WRD_4 0x12                /* define control-word_4     */
#define CTRL_WRD_5 0x52                /* define control-word_5     */
#define CTRL_WRD_6 0x92                /* define control-word_6     */
#define CTRL_WRD_7 0x12                /* define control-word_7     */
#define CTRL_WRD_8 0x52                /* define control-word_8     */
#define CTRL_WRD_9 0x92                /* define control-word_9     */
#define COUNTER_1  0x2000              /* address counter_1         */
#define COUNTER_2  0x2001              /* address counter_2         */
#define COUNTER_3  0x2002              /* address counter_3         */
#define COUNTER_4  0x4000              /* address counter_4         */
#define COUNTER_5  0x4001              /* address counter_5         */
#define COUNTER_6  0x4002              /* address counter_6         */
#define COUNTER_7  0x6000              /* address counter_7         */
#define COUNTER_8  0x6001              /* address counter_8         */
#define COUNTER_9  0x6002              /* address counter_9         */

/* define stop values per axis          */
#define STOP0       0x1D
#define STOP1       0x20
#define STOP2       0x1E
#define STOP3       0x1D
#define STOP4       0x1D
#define STOP5       0x1D
#define STOP6       0x1D
#define STOP7       0x1D
```

```
/*      A/D + Analogue Switch Address + Control Words    */
#define IO_DELAY 1
#define MSB_ADDR 0x8000
#define LSB_ADDR 0xA000

/*      Motor Driver Logic Signal Definitions    */
#define RESET_PULSE_WIDTH 5000

/*     AD Digital filter charcteristics  */
#define DATA_SAMPLE_LENGTH 0x10
#define AD_ALPHA 1
#define AD_BETA 9


/* Gain Constants      */
#define Kp0       1.2  /* Base Down 1.2 */
#define Kp1       4.0
#define Kp2       4.0       /* 4.6 */
#define Kp3       2.0       /* 2.0  */
#define Kp4       3.8
#define Kp5           1.2    /*  wrist up (spring expands)*/
#define Kp6       1.5
#define Kp7       3.8

#define Kq0       3.2  /* Base up  3.2*/
#define Kq1       4.0
#define Kq2       3.6   /* 3.6 */
#define Kq3       4.0
#define Kq4       3.8
#define Kq5           1.2   /*  wrist down  */
#define Kq6       1.5
#define Kq7       3.8

#define Kd0       1.6
#define Kd1       0.5
#define Kd2       1.5
#define Kd3       0.35
#define Kd4       0.8
#define Kd5           0.1
#define Kd6       0.8
#define Kd7       0.8


// Serial I/O constants
#define ERROR -1
#define SERIAL_MODE_1 0x52  /*   8 bit UART  */
#define TIMER_MODE_2 0x20   /*  Timer 1 8 bit auto reload */
#define TIMER_COUNT 0xF3   /* Use for 2400 Baud */
#define FOR_EVER 1
#define BIT_6_MASK 0x80
#define BIT_7_MASK 0x40
#define SER_TIME_OUT 32000

//      Gain arrays
static float Kp[NUMBER_OF_AXIS] = {Kp0,Kp1,Kp2,Kp3,Kp4,Kp5};
static float Kq[NUMBER_OF_AXIS] = {Kq0,Kq1,Kq2,Kq3,Kq4,Kq5};

// Speed limits for each axis
static char MaxSig[NUMBER_OF_AXIS][SPEED_LEVELS] ={0x08,0x09,0x0A,
0x07,0x0A,0x0C,
0x08,0x09,0x0A,
0x04,0x04,0x08,
0x16,0x1A,0x1A,
0x06,0x07,0x08};
static char MinSig[NUMBER_OF_AXIS][SPEED_LEVELS] ={0x0F,0x10,0x11,
0x07,0x0A,0x0C,
0x06,0x07,0x08,
0x07,0x08,0x0A,
0x16,0x1A,0x1A,
0x07,0x08,0x09};

// Position limits
```

```
static char PosMin[NUMBER_OF_AXIS] = {0x07,0x06,0x07,0x02,0x14,0x05};
static char NegMin[NUMBER_OF_AXIS] = {0x0C,0x06,0x06,0x02,0x14,0x05};

// initial Speed setting
unsigned char SpeedSetting = 0x01;
#define SPEED_LEVELS 3

/* Control Data structures   */
unsigned char  ZeroSpeedOffset[NUMBER_OF_AXIS]= {STOP0,STOP1,STOP2,STOP3,STOP4,STOP5};
float MotorAlpha = 0.01;
float MotorBeta = 0.99;
char Axis;
int  CurrentPos[NUMBER_OF_AXIS];
int  TargetPos[NUMBER_OF_AXIS];
int  Error[NUMBER_OF_AXIS];
unsigned char  MinPos[NUMBER_OF_AXIS] = {0x2A,0x32,0x0F,0x0A,0x14,0x00};
unsigned char  MaxPos[NUMBER_OF_AXIS] = { 0xB0,0xFF,0x8C,0xE6,0xDA,0xFF};
unsigned char  DynamicError[NUMBER_OF_AXIS] = {0x01,0x01,0x01,0x01,0x01,0x05};
unsigned char  HltBand[NUMBER_OF_AXIS] = {0x01,0x02,0x01,0x02,0x02,0x0};
char Reached[NUMBER_OF_AXIS] = {TRUE,TRUE,TRUE,TRUE,TRUE,TRUE};
float PrevSignal[] = {0,0,0,0,0,0};
```

Appendix D

Issuing JMCL for Task execution

The following appendix provides code examples that may be used as a template to create an executable. The application may sequence and issue JMCL commands to perform a pre-determined task. Code examples are provided in section 1, with a serial IO library listed in section 2.

1. Code examples for task automation

```
// To set motor brakes call :
      s_putc(BRK);

//    To reset motor brakes call :
      s_putc(RST);

//  To stop all axes call :
      s_putc(HLT);

//    To Check for serial in
      if((pos = s_inchar()) != NOT_READY)

//    To move axis 0 in direction 0 :
      s_putc(M00);

//    To move axis 0 in direction 0 :
      s_putc(M01);

//    To set speed level
      s_putc(S31);


////////////////////////////////////////////////////////////
//    Example code to read  position of axis 0

      s_putc(WI0);
      Timer(TIMER_RESET);

      while(((pos = s_inchar()) == NOT_READY) && Timer(TIMER_INC));

      if(!Timer(TIMER_TEST))
                  return -1;

      s_putc(NXT);

      Timer(TIMER_RESET);

      pos <<= 4;

      while(((temp = s_inchar()) == NOT_READY) && Timer(TIMER_INC));

      if(!Timer(TIMER_TEST))
                  return -1;
      pos += (temp & 0x0F);
      return pos;
////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////
// Settint axis 0 to position PosNo
            s_putc(P00);

            Timer(TIMER_RESET);
            while(((Next = s_inchar()) == NOT_READY) && Timer(TIMER_INC));

            if(!Timer(TIMER_TEST))
                  return ERROR;
            else if (Next == NXT)
                  s_putc(PosNo);

////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//    Moving axis A from x to y

      MoveAxisTo(A,x);

      MoveAxisTo(A,y);



int MoveAxisTo(int Axis, int Pos)
{
      int Next;

      s_putc(P00+Axis);
      Timer(TIMER_RESET);
      while(((Next = s_inchar()) == NOT_READY) && Timer(TIMER_INC));

      if(!Timer(TIMER_TEST))
      {
            printf("\n\tTimed out on Nxt ");
            delay(500);
            return 0;
      }
      else if (Next == NXT)
                  s_putc(Pos);

      return 1;
}
////////////////////////////////////////

////////////////////////////////////////////////////////////////////
```

2 Serial IO routines

```
////////////////////////////////////////////////////////////////////
// Serial I/O functions

struct serial
      {
      int uart_base;
      int data_off;
      int status_off;
      unsigned rcvmask;
      unsigned xmitmask;
      };


static struct serial sio =
      {
            COMM2,
            DP_OFF,
            SP_OFF,
            RCV_MASK,
            XMIT_MASK,
      };

BYTE s_rcv()
{
      return( inportb(sio.uart_base + sio.data_off));
}


BYTE s_rcvstat()
{
      return( inportb(sio.uart_base + sio.status_off) & sio.rcvmask);
}


void s_xmit(BYTE c)
{
            outportb(sio.uart_base + sio.data_off, c);
}


BYTE s_xmitstat()
{
      return( inportb(sio.uart_base + sio.data_off) & sio.xmitmask);
}


int s_inchar()
{
      return(s_rcvstat() == NULL ? NOT_READY : s_rcv());
}
```

```
void s_putc(BYTE c)
{
      while(s_xmitstat()== NULL);
      s_xmit(c);
}


int Timer(BOOL Call)
{
      static long count;
      int i;

      for(i=0; i<1000; i++);

      if(Call == TIMER_RESET)
            count = 0;
      else if (Call == TIMER_INC)
            count++;

      return ((count <TIME_OUT) ? 1 : 0);
}
```

Appendix E

Juvo User Control Language

The following appendix provides a summary of the commands that may be issued with the Middlesex Manipulator's interface system. As the interface system is designed to be adaptable, the configurations of the interface may vary in the order and number of commands presented to the user.

Level 0

Branch	A	B	C	D	E
	DoTask	Goto	Speed	MoveArm	Move
Branch	F	G	I		
	HereIs	TeachTask	Power		

Branch A DoTask

Stem	A	B	C	D	E	F
Level						
1	†Task	End_{L0}				
2	Go	Speed	End_{L0}			
1	†Joint					
2	†$Level_{L2}$					
3	Stop					

Branch B **Goto**

Stem Level	A	B	C
1	†Position	End_{L0}	
2	Go	Speed †$Level_{L2}$	End_{L0}
1			
3	Stop		
4	$Continue_{L3}$	Speed †$Level_{L4}$	End_{L0}

Branch C **Speed**

Stem Level	A	B	C
1	†Joint	End_{L0}	
2	†$Level_{L0}$	End_{L0}	

Branch D **MoveArm**

Stem Level	A	B	C
1	†C_Dir	End_{L0}	
2	Go	Speed †$Level_{L2}$	End_{L0}
1			
3	Stop		
4	$Continue_{L3}$	Speed †$Level_{L4}$	End_{L0}
1			

Branch E **Move**

Stem Level	A	B	C
1	†Joint	End_{L0}	
2	†C_Dir	End_{L0}	
3 1	Go	Speed †Level $_{L3}$	End_{L0}
4	Stop		
5 1	Continue $_{L4}$	Speed †Level $_{L5}$	End_{L0}

Branch F **HereIs**

Stem Level	A	B	C
1	†Position †Confirm $_{L0}$	Where	End_{L0}

Branch G **Teach Task**

Stem Level	A	B	C	D	E	F
1	†Task	End_{L0}				
2 1 2	GoTo †Position	Speed Level $_{L2}$	MoveArm †C_Dir	Move †Joint †J_Dir	Wait $_{L2}$	End_{L0}
3	Go	End_{L0}				
4	Stop $_{L2}$					

Branch H **Home**

Stem Level	A	B
1	†Go	End_{L0}
2	Stop	
3	$Continue_{L2}$	End_{L0}

Branch I **Power**

Stem Level	A	B	
1	On_{L0}	Off_{L0}	End_{L0}

Branch J **Confirm**

Stem Level	A	B
1	Yes_{L0}	No_{L0}

†	Indicates List
L2	Read as return to level 2
.....	Indicates items occur as options

Appendix F

User Interface Code Listings

The following appendix provides example code listings of the user interface system. The module provided corresponds to the Dialogue Manager component of the user interface system. For complete code listings of all Modal Logic Units and Input Device Modules refer to the Disk 3 included with the thesis.

1. DMAN.CPP

```
/*
                                DMAN.CPP

Source code for dialogue manager application, part of a suite of applications
that combine to form a user interface and control system for the JUVO
manipulator.

DMAN acts as a client and communicates with a number of servers via DDE.
Conversations are established with at least one Input Device Module (IDM)
and one Feedback Device Module (FDM). The IDM receives commands from the
user witch are dispatched to DMAN. DMAN responds by despatching status
information to the FDM, and sending the command code to the appropriate
Modal Logic Unit (MLU), this may involve first activating the MLU (establishing a DDE
conversation).
The MLU will respond with a list of command codes, which are then dispatched
to the IDM and FDM.
*/

#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\framewin.h>
#include <owl\dc.h>
#include <owl\menu.h>
#include <owl\inputdia.h>
#include "dman.rh"
#include <ddeml.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "JUCL.h"

// Forward declaration of main window
class TDMLClWnd;

// Main application class
class TDMLClApp : public TApplication {
  public:
        TDMLClApp() : TApplication(),CallBackProc((FARPROC)CallBack) {
              InstId = 0;
              }
        void InitMainWindow();
        void InitInstance();
        int    TermInstance(int status);

        DWORD   InstId;

        // Call back function used for DDE comms
        static HDDEDATA FAR PASCAL _export CallBack(WORD, WORD, HCONV, HSZ, HSZ,

                                HDDEDATA, DWORD, DWORD);

        TProcInstance CallBackProc;
};


class TDMLClWnd : public TFrameWindow {
  public:
        TDMLClWnd(TWindow*, const char*);
        virtual ~TDMLClWnd();

        virtual void SetupWindow();

        void EvInitMenu(HMENU);
```

```
        int    Connect();  // Establish IDM & FDM conversations
        int  Advise();              // Establish advise loops with IDM & FDM
        void IRequest();      // Request Data from IDM
        void FRequest();      // FDM
        void MLURequest();//  MLU

        void IPoke(unsigned char *);         // Poke Data to IDM
        void FPoke(unsigned char *Data);     // Poke Data to FDM
        void MLUPoke(unsigned char *);              // Poke Data to MLU

        void IReceivedData(HDDEDATA);
        void FReceivedData(HDDEDATA);
        void MLUReceivedData(HDDEDATA);

//  Establish and terminate conversation with MLU named by char *
        int    ActivateMLU(unsigned char *);
        void   DeActivateMLU();

//      Array holds JUCL commands
        unsigned char FCommandList[25];

// Menu option calls connect and advise, and issues initial
// command list to IDM
        void CmInit();


        DWORD   InstId() {
              return ((TDMLClApp*)GetApplication())->InstId;
              }

        BOOL          Initialised; // True if CmInit has been called

// Standard DDE variables for IDM, FDM and MLU

        HCONV   HIConv;
        BOOL    ILoop;
        HSZ     IService;
        HSZ     ITopic;
        HSZ     IItem;

        HCONV   HFConv;
        BOOL    FLoop;
        HSZ     FService;
        HSZ     FTopic;
        HSZ     FItem;

        HCONV   HMLUConv;
        BOOL    MLULoop;
        HSZ     MLUService;
        HSZ     MLUTopic;
        HSZ     MLUItem;

        unsigned char    IData[25]; // Data recieved from IDM
        unsigned char    FData[25]; // Data recieved from FDM
        unsigned char    MLUData[25];//  Data recieved from MLU
        unsigned char        CommandList[25];// If MLUData is a command list
                                                          // its copied into
CommandList
                                                          // which is poked to IDM

        char                         MLUName[20];  // Name of active MLU for paint
        BOOL                         MLUActive;       // True if an MLU is currently active

        DECLARE_RESPONSE_TABLE(TDMLClWnd);
};

DEFINE_RESPONSE_TABLE1(TDMLClWnd, TFrameWindow)
  EV_WM_INITMENU,
  EV_COMMAND(CM_INIT, CmInit),
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
END_RESPONSE_TABLE;
```

```
static TDMLClWnd* This = 0;

TDMLClWnd::TDMLClWnd(TWindow* parent, const char* title)
  : TFrameWindow(parent, title),
        TWindow(parent, title)
{
  // Initialize DDE variables
  IData[0] = 0;
  HIConv = 0;
  ILoop = 0;

  FData[0] = 0;
  HFConv = 0;
  FLoop = 0;

  MLUData[0] = 0;
  HMLUConv = 0;
  MLULoop = 0;

  MLUActive = 0;
  Initialised = 0;

// Window attributes
  Attr.X = 75;
  Attr.Y = 105;
  Attr.W = 600;
  Attr.H = 45;

}


TDMLClWnd::~TDMLClWnd()
{
  // This clean up is required for those resources that were allocated during
  // the DDEML conversation.
  //
  if (HIConv)
        DdeDisconnect(HIConv);      // Let the other party know we are leaving

  //if (HFConv)
        //DdeDisconnect(HFConv);

  if (HMLUConv)
        DdeDisconnect(HMLUConv);


  // Free allocated DDE memory.

  if (InstId())
  {
        DdeFreeStringHandle(InstId(), IService);
        DdeFreeStringHandle(InstId(), ITopic);
        DdeFreeStringHandle(InstId(), IItem);

        DdeFreeStringHandle(InstId(), FService);
        DdeFreeStringHandle(InstId(), FTopic);
        DdeFreeStringHandle(InstId(), FItem);

        if(MLUActive)
              DeActivateMLU();
  }
}

void
TDMLClWnd::SetupWindow()
{
  This = this;  // Requied because callback function not passed this
  TFrameWindow::SetupWindow();

  AssignMenu(TDMLClWnd_MENU);
```

```
  // create resources for IDM and FDM conversations

  IService = ITopic = IItem = 0;
  FService = FTopic = FItem = 0;
  MLUService = MLUTopic = MLUItem = 0;
  MLUActive = 0;

  IService = DdeCreateStringHandle(InstId(), "JUVO", CP_WINANSI);
  ITopic = DdeCreateStringHandle(InstId(), "IDM1", CP_WINANSI);
  IItem = DdeCreateStringHandle(InstId(), "JUCL", CP_WINANSI);
  if (!IService || !ITopic || !IItem)
  {
        MessageBox("Creation of strings for IDM1 failed.", Title, MB_ICONSTOP);
        PostQuitMessage(0);
  }

  FService = DdeCreateStringHandle(InstId(), "JUVO", CP_WINANSI);
  FTopic = DdeCreateStringHandle(InstId(), "FDM1", CP_WINANSI);
  FItem = DdeCreateStringHandle(InstId(), "JUCL", CP_WINANSI);
  if (!FService || !FTopic || !FItem) {
        MessageBox("Creation of strings for FDM1 failed .", Title, MB_ICONSTOP);
        PostQuitMessage(0);
  }

}

// Acitvate an MLU named by MLUID, and enter and advise loop
int
TDMLClWnd::ActivateMLU(unsigned char *MLUID)
{
        DWORD temp;

        MLUService = MLUTopic = MLUItem = 0;

        MLUService = DdeCreateStringHandle(InstId(), "JUVO", CP_WINANSI);
        MLUTopic = DdeCreateStringHandle(InstId(), (char *)MLUID, CP_WINANSI);
        MLUItem = DdeCreateStringHandle(InstId(), "JUCL", CP_WINANSI);
        if (!MLUService || !MLUTopic || !MLUItem)
                return 0;

        HMLUConv = DdeConnect(InstId(),MLUService, MLUTopic, 0);
        if (HMLUConv)
                if(DdeClientTransaction(0, 0, HMLUConv, MLUItem, CF_TEXT, XTYP_ADVSTART |
XTYPF_ACKREQ, 1000, &temp))
                        {
                                MLUActive = 1;
                                return 1;
                        }

        return 0;
}

// Deactivate the currently active  MLU
void
TDMLClWnd::DeActivateMLU()
{
        if (HMLUConv)
                DdeDisconnect(HMLUConv);

         DdeFreeStringHandle(InstId(), MLUService);
         DdeFreeStringHandle(InstId(), MLUTopic);
         DdeFreeStringHandle(InstId(), MLUItem);

         MLUData[0] = 0;
         HMLUConv = 0;
         MLULoop = 0;
         MLUActive = 0;
}

void
```

```
TDMLClWnd::EvInitMenu(HMENU menuHandle)
{
  TMenu menu(menuHandle);
  DrawMenuBar();
}

// Establish conversations with IDM and FDM
int
TDMLClWnd::Connect()
{
  HIConv = DdeConnect(InstId(),IService, ITopic, 0);
  if (HIConv)
        PostMessage(WM_INITMENU, WPARAM(GetMenu()));
  else
  {
        MessageBox("Can't connect to IDM1 ",Title,MB_ICONSTOP);
        return 0;
  }

  HFConv = DdeConnect(InstId(),FService, FTopic, 0);
  if (HFConv)
        PostMessage(WM_INITMENU, WPARAM(GetMenu()));
  else
  {
        MessageBox("Can't connect to FDM1 ",Title,MB_ICONSTOP);
        return 0;
  }

  return 1;
}


// Start continuous advise loops with IDM and FDM

int
TDMLClWnd::Advise()
{
        DWORD temp;
        if(!ILoop)
        {
                if(DdeClientTransaction(0, 0, HIConv, IItem, CF_TEXT, XTYP_ADVSTART |
XTYPF_ACKREQ,           1000, &temp))
                        ILoop = TRUE;
                else
                        MessageBox("Could not start advise loop for IDM1", Title, MB_ICONSTOP);
        }

        if(!FLoop)
        {
                if(DdeClientTransaction(0, 0, HFConv, FItem, CF_TEXT, XTYP_ADVSTART |
XTYPF_ACKREQ,           1000, &temp))
                        FLoop = TRUE;
                else
                        MessageBox("Could not start advise loop for FDM1", Title, MB_ICONSTOP);
        }

        return (ILoop && FLoop);
}

// Request a data item from the IDM (usually triggered by advise loop).
// ReceiveData will be called asynchronously by the callback.
void
TDMLClWnd::IRequest()
{
  DdeClientTransaction(0, 0, HIConv, IItem, CF_TEXT, XTYP_REQUEST, TIMEOUT_ASYNC, 0);
}

void
TDMLClWnd::FRequest()
{
  DdeClientTransaction(0, 0, HFConv, FItem, CF_TEXT, XTYP_REQUEST, TIMEOUT_ASYNC, 0);
```

```
}

// Poke a string over to IDM1
void
TDMLClWnd::IPoke(unsigned char *Data)
{
  DdeClientTransaction(Data, strlen((char *)Data)+1, HIConv, IItem, CF_TEXT, XTYP_POKE, 1000,
       0);
}
// Poke a string over to FDM
void
TDMLClWnd::FPoke(unsigned char *Data)
{
  DdeClientTransaction(Data, strlen((char *)Data)+1, HFConv, FItem, CF_TEXT, XTYP_POKE, 1000,
       0);
}

// Poke a string over to MLU
void
TDMLClWnd::MLUPoke(unsigned char *Data)
{
  DdeClientTransaction(Data, strlen((char *)Data)+1, HMLUConv, MLUItem, CF_TEXT, XTYP_POKE,
       1000, 0);
}


void
TDMLClWnd::CmHelpAbout()
{
  MessageBox("DMAN written by B. Parsons "
                              "JUVO Controller Software",
                              "About DMAN", MB_ICONINFORMATION);
}

//
// This function is called when the callback function is notified of
// available data from the IDM.
// The function checks to see if an MLU is active, and activates one
// if not, using the code from the IDM as the MLU code.
// The Code from the IDM is the dispatched to the MLU.
// A string is copied into MLUName for paint info

void
TDMLClWnd::IReceivedData(HDDEDATA hData)
{
       if (hData)
       {
              DdeGetData(hData, IData, sizeof IData, 0);

              if(!MLUActive)
                     if(ActivateMLU(IData))
                     {
                            FCommandList[0] = CommandListBegin;
                            FCommandList[1] = IData[0];
                            FCommandList[2] = CommandListEnd;
                            FCommandList[3] = 0x00;
                            FPoke(FCommandList);
                     }

              if(MLUActive)
                     MLUPoke(IData);
       }
}

void
TDMLClWnd::FReceivedData(HDDEDATA hData)
{
  if (hData)
  {
        DdeGetData(hData, (unsigned char*)FData, sizeof FData, 0);
  }
```

```
}

// Data recieved from MLU. If data is a command List, data is copied
// into CommandList and dispatched to IDM.
// If data is END, MLU is deactivated, and start-up command list is
// dispatched to IDM.

void
TDMLClWnd::MLUReceivedData(HDDEDATA hData)
{
  int i = 0;
  if (hData)
  {
        DdeGetData(hData, (unsigned char*)MLUData, sizeof MLUData, 0);

        if(MLUData[0] == CommandListBegin)
        {
              while(MLUData[i])
              {
                    CommandList[i] = MLUData[i];
                    i++;
              }
              CommandList[i] = 0x00;
              IPoke(CommandList);
        }

        if(MLUData[0] == END)
        {
                    DeActivateMLU();
                    CommandList[0] = CommandListBegin;
                    CommandList[1] = DOTASK;
                    CommandList[2] = GOTO;
                    CommandList[3] = SPEED;
                    CommandList[4] = MOVEARM;
                    CommandList[5] = MOVE;
                    CommandList[6] = HEREIS;
                    CommandList[7] = TEACHTASK;
                    CommandList[8] = HOME;
                    CommandList[9] = POWER;
                    CommandList[10] = CommandListEnd;
                    CommandList[11] = 0x00;
                    IPoke(CommandList);

                    FCommandList[0] = CommandListBegin;
                    FCommandList[1] = END;
                    FCommandList[2] = CommandListEnd;
                    FCommandList[3] = 0x00;
                    FPoke(FCommandList);

        }

  }
}

// Establish IDM and FDM conversations and advise loops.
// dispatch start-up command list to IDM

void TDMLClWnd::CmInit()
{
       if(Connect() && Advise())
       {
             CommandList[0] = CommandListBegin;
             CommandList[1] = DOTASK;
             CommandList[2] = GOTO;
             CommandList[3] = SPEED;
             CommandList[4] = MOVEARM;
             CommandList[5] = MOVE;
             CommandList[6] = HEREIS;
             CommandList[7] = TEACHTASK;
             CommandList[8] = HOME;
             CommandList[9] = POWER;
```

```
                CommandList[10] = CommandListEnd;
                CommandList[11] = 0x00;
                IPoke(CommandList);
                Initialised = 1;
        }
}

// Call back procedure handles DDE messages from DDEML
HDDEDATA FAR PASCAL _export
TDMLClApp::CallBack(WORD type, WORD, HCONV hConv, HSZ, HSZ, HDDEDATA hData,
                                                    DWORD, DWORD)
{
  switch (type) {
        case XTYP_ADVDATA:
                if (hConv == This->HIConv)
                  This->IReceivedData(hData);
                if (hConv == This->HFConv)
                  This->FReceivedData(hData);
                if (hConv == This->HMLUConv)
                  This->MLUReceivedData(hData);
                return (HDDEDATA)DDE_FACK;

        case XTYP_XACT_COMPLETE:
                if (hConv == This->HIConv)
                  This->IReceivedData(hData);
                if (hConv == This->HFConv)
                  This->FReceivedData(hData);
                if (hConv == This->HMLUConv)
                  This->MLUReceivedData(hData);
                break;

        // Potential problems here !!

        case XTYP_DISCONNECT:
                This->MessageBox("Disconnected.", This->Title, MB_ICONINFORMATION);
                This->HIConv = 0;
                This->ILoop = 0;
                This->HFConv = 0;
                This->FLoop = 0;

                This->PostMessage(WM_INITMENU, WPARAM(This->GetMenu()));
                break;

        case XTYP_ERROR:
                This->MessageBox("A critical DDE error has occured.", This->Title,
                                MB_ICONINFORMATION);
  }
  return 0;
}


void
TDMLClApp::InitMainWindow()
{
  MainWindow = new TDMLClWnd(0, "JUVO Dialogue Manager");
}

void
TDMLClApp::InitInstance()
{
  // The code below sets up the DDEML call back function that is used by the
  // DDE Management Library to carry out data transfers between
  // applications.
  //
  if (DdeInitialize(&InstId, (PFNCALLBACK)(FARPROC)CallBackProc, APPCMD_CLIENTONLY, 0) !=
                        DMLERR_NO_ERROR) {
                ::MessageBox(0,"Initialization failed.", "DDEML Client",
MB_ICONSTOP|MB_TASKMODAL);
                PostQuitMessage(0);
        }
```

```
  // Must come after we've initialized DDE since InitInstance will trigger
  // SetupWindow
  TApplication::InitInstance();
}

int
TDMLClApp::TermInstance(int status)
{
  if (InstId) {
        DdeUninitialize(InstId);
  }
  return TApplication::TermInstance(status);
}


int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TDMLClApp().Run();
}
```

Appendix G

Neural Network Code Listings

BP.CPP provides an implementation of a backpropagation artificial neural network. This file may be linked with an application's main program, and provides an API allowing a neural network to be configured, trained, and used for classification. The API provides the following interface.

```
// BP.CPP


#include "bp2.h"
#include "matrix.h"

#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include "Memman.h"

#define FALSE 0
#define TRUE !FALSE
#define MAX_FILENAME 25
#define LRMin 0.1
#define PI 3.14159265359

typedef unsigned char BOOL;

// Length of inputvector;
static int VectorLength = DEFAULT_VECTOR_LENGTH;

// Lengths of input, hidden and output layers
static int K = DEFAULT_K;
static int J = DEFAULT_J;
static int I = DEFAULT_I;

// network initialised flag
static int Initialised = 0;

// Arrays of input and target vectors for network training
static float *XN[MAX_VECTORS]; // Input Vectors
static float *TM[MAX_VECTORS]; // Target Vectors


static int N;       // Number of input vectors
static int M;       // Number of target vectors
static float *X;    // Current Input Vector
static float *T;    // Current Target Vector
static float *U;    // Cluster centre outputs
static float *WH[MAX_J]; // Array of hidden layer weights
static float *WHChange[MAX_J]; // Last weight change
static float *Y;           // Output Layer node outputs
static float *E;           // Network Error
static float *WO[MAX_I]; // Array of output layer weights
static float *WOChange[MAX_I];  // Last weight change

//     Network traning variables
static float *DeltaOut;
static float *DeltaHidden;
static float *WeightedDeltaSum;
static float LearningRate = 0.9;
static float LRDescentRate = 0.005;
static float Momentum = 0.8;
static long NumTrainingCycles = DEF_TRAINING_CYCLES;
int i,j,k;    // Loop count variables
long double Temp;


//     Returns current size of input layer
int GetInputNodes()
{
```

```
        return K;
}

//      Sets current size of input layer
void SetInputNodes(int k)
{
        K = k;
}

//      Returns current number of input vectors
int GetN()
{
        return N;
}

//      Returns current number of hidden nodes
int GetHiddenNodes()
{
        return J;
}

//      Sets current number of hidden nodes
void SetHiddenNodes(int j)
{
        J = j;
        if(J > MAX_J)
                J = MAX_J;
}

//      Returns current number of output nodes
int GetOutputNodes()
{
        return I;
}

//      Sets current number of output nodes
void SetOutputNodes(int i)
{
        I = i;

        if(I > MAX_I)
                I = MAX_I;
}

//      Returns current length of input vector
int GetVectorLength()
{
        return VectorLength;
}

//      Sets current length of input vector
void SetVectorLength(int L)
{
        VectorLength = L;
}

//      Returns current learning rate
int GetLearningRate()
{
        return (int ) (LearningRate * 100);
}

//      Sets current learning rate
void SetLearningRate(int L)
```

```
{
	LearningRate = ((float )L) /100.00;
}

//	Returns descent rate
int GetDescentRate()
{
	return (int ) (LRDescentRate * 10000);
}

//	Sets descent rate
void SetDescentRate(int L)
{
	LRDescentRate = ((float )L) /10000.00;
}

//	Sets number of training cycles
void SetNoTrainingCycles(int N)
{
	NumTrainingCycles = N;
	NumTrainingCycles *= 1000;
}

//	Returns number of training cycles
int GetNoTrainingCycles()
{
	return (int ) (NumTrainingCycles / 1000);
}

//	Allocates memory for network
int InitDataStructures()
{
	if(Initialised)
	{
		if (X) delete [] X;
		if (T) delete [] T;
		if (U) delete [] U;
		if (WH) delete [] WH;
		if (WHChange) delete [] WHChange;
		if (Y) delete [] Y;
		if (E) delete [] E;
		if (WO) delete [] WO;
		if (WOChange) delete [] WOChange;
		if (DeltaOut) delete [] DeltaOut;
		if (DeltaHidden) delete [] DeltaHidden;
		if (WeightedDeltaSum) delete [] WeightedDeltaSum;
		Initialised = 0;
	}


	X = Newfloat1(K);	// Current Input Vector
		if(!X)
			return 0;

	T = Newfloat1(I);	// Current Target Vector
		if(!T)
			return 0;

	U = Newfloat1(J); // Cluster centre outputs
		if(!U)
			return 0;

	for(int j = 0; j < J; j++)
	{
```

```
		WH[j] = Newfloat2(K); // Array of hidden layer weights
		if(!WH[j])
			return 0;
	}

	for(j = 0; j<J; j++)
	{
		WHChange[j] = Newfloat3(K); // Last weight change
		if(!WHChange[j])
			return 0;
		else
			for(int k = 0; k<K; k++)
				WHChange[j][k] = 0.0;
	}

	Y = Newfloat1(I);		// Output Layer node outputs
		if(!Y)
			return 0;

	E = Newfloat1(I);		// Network Error
		if(!E)
			return 0;

	for(int i = 0; i<I; i++)
	{
		WO[i] = Newfloat4(J); // Array of output layer weights
		if(!WO[i])
			return 0;
	}

	for(i = 0; i<I; i++)
	{
		WOChange[i] = Newfloat5(J);  // Last weight change
		if(!WOChange[i])
			return 0;
		else
			for(int j = 0; j<J; j++)
				WOChange[i][j] = 0.0;
	}

	DeltaOut = Newfloat1(I);
		if(!DeltaOut)
			return 0;

	DeltaHidden = Newfloat1(J);
		if(!DeltaHidden)
			return 0;

	WeightedDeltaSum = Newfloat1(J);
		if(!WeightedDeltaSum)
			return 0;

	Initialised = 1;
	return 1;
}


// Calculates network output, populating array Y
void ComputeNetOut()
{
	for(i=0; i<I; i++)
	{
		Y[i] = VectorMult(&WO[i][0], U, J);
		Y[i] = 1/(1+(exp(-1*Y[i])));
```

```
        }
}


// Calculates hidden output, populating array U
void ComputeHiddenOut()
{
        for(j=0; j<J; j++)
        {
                U[j] = VectorMult(&WH[j][0], X, K);
                Temp = expl((long double)(-1*U[j]));
                U[j] = 1/(1+ (float)Temp);
        }
}

// Adjust weights in the output layer
void UpdateOutWeights()
{
        // Compute Delta Out
        for(i = 0 ; i<I; i++)
                        DeltaOut[i] = Y[i]*(1 - Y[i])*E[i];

        // Adjust weights
        for(i = 0; i<I; i++)
         for(j = 0; j<J; j++)
         {
          WOChange[i][j] =(LearningRate*DeltaOut[i]*U[j])+(Momentum*WOChange[i][j]);
          WO[i][j] += WOChange[i][j];
         }
}

// Adjust weights in the hidden layer
void UpdateHiddenWeights()
{

        // Compute Delta Hidden
        for(j = 0;j<J;j++)
                WeightedDeltaSum[j] = 0;

        for(j = 0;j<J;j++)
        {
                for(i = 0; i<I; i++)
                                WeightedDeltaSum[j] += DeltaOut[i]* WO[i][j];
                DeltaHidden[j] = U[j] * (1 - U[j]) * WeightedDeltaSum[j];
        }

        // Update Weights
        for(j = 0; j<J; j++)
         for(k = 0; k<K; k++)
         {
          WHChange[j][k]=(LearningRate*DeltaHidden[j]*X[k])+(WHChange[j][k]*Momentum);
          WH[j][k] += WHChange[j][k];
         }

}

// Randomize weight values, called before training
void RandomizeWeights()
{

        time_t t;
        srand((unsigned) time(&t));

        // Randomize weights
```

```
        for(i = 0; i<I; i++)
                for(j = 0; j<J; j++)
                        WO[i][j] = ((float )random(99)) / 10000.00;

        for(j = 0; j<J; j++)
                for(k = 0; k<K; k++)
                        WH[j][k] = ((float )random(99)) / 10000.00;
}


// Train the neural network using exsisting contents of inputs X
// and targets T
int Train()
{

        int n = 0;
        int Cycles = 0, Loops  = 0;

        RandomizeWeights();

        do{
                // Choose next training pair
                n = random(N);
                CopyVector(T, TM[n], I);
                CopyVector(X, XN[n], K);

                ComputeHiddenOut();
                ComputeNetOut();

                // Calc Error
                VectorSub(T, Y , I, E);
                UpdateOutWeights();
                UpdateHiddenWeights();

                Loops++;
                if(!(Loops%N))
                {
                        Loops = 0;
                        Cycles++;
                        //if(ErrorAcceptable())
                          //  //   break;
                        LearningRate -= LRDescentRate/250.00;
                        LearningRate = (LearningRate < LRMin ) ? LRMin : LearningRate;
                }
        }while(Cycles < NumTrainingCycles);

        if(Cycles < NumTrainingCycles)
                return 1;
        else
                return 0;
}

// Determine whether current network errors are acceptable
int ErrorAcceptable()
{
        for(i = 0; i< I; i++)
                if(Mag(E[i]) > ACCEPTABLE_ERROR)
                        return 0;

        return 1;
}

// Function classifies an input vector, populating Netout array
// with the network output
```

```
int Classify(int Vector, float *NetOut)
{
	// Create X from gesture
	float MaxOutVal  = -9999;
	int WinningNode = -1;


	for(int k =0; k<K; k++)
		X[k] = (float ) XN[Vector][k];

	ComputeHiddenOut();
	ComputeNetOut();

	// find maximum output

	for(int i = 0; i<I; i++)
	{
		NetOut[i] = Y[i];
		if(Y[i] > MaxOutVal)
		{
			MaxOutVal = Y[i];
			WinningNode = i;
		}
	}
	return WinningNode + 1;
}

// Function Reads a set of input vectors from a file, populating
// the array XN
int ReadInputVectors(char *FileName)
{
	FILE *fp;
	char VectorFileName[MAX_FILENAME];

	float Buffer[MAX_K];

	strcpy(VectorFileName, FileName);
	strcat(VectorFileName,".pvt");

	if((fp = fopen(VectorFileName,"r")) ==  NULL)
		return 0;

	N = 0;
	while(!feof(fp))
	{
		for(int i = 0; i< K; i++)
		{
			fscanf(fp, "%f", &Buffer[i]);
			Buffer[i]/=100;
		}

		XN[N] = new float [K];
		if(!XN[N])
		{
			delete XN;
			return 0;
		}

		CopyVector(XN[N], Buffer, K);
		N++;
	}
	fclose(fp);
	N--;
	return 1;
```

```
}

// Function Reads a set of target vectors from a file, populating
// the array T
int ReadTargetVectors(char *FileName)
{
	FILE *fp;
	float Buffer[MAX_I];

	char TargetFileName[MAX_FILENAME];

	strcpy(TargetFileName,FileName);
	strcat(TargetFileName,".tvt");

	if((fp = fopen(TargetFileName,"r")) ==  NULL)
		return 0;

	M = 0;
	while(!feof(fp))
	{
		for(int i = 0; i< I; i++)
			fscanf(fp, "%f", &Buffer[i]);

		TM[M] = new float [I];
		if(!TM[M])
		{
			delete TM;
			return 0;
		}

		CopyVector(TM[M], Buffer, I);
		M++;
	}
	fclose(fp);
	return 1;
}
// Function saves a set of network weights stored in WH and WO
// to a file
int SaveWeights(char *WeightFileName)
{
	FILE *fp;


	if((fp = fopen(WeightFileName,"w")) ==  NULL)
		return 0;


	// Output Hidden weigths
	for(int j = 0; j< J; j++)
	{
		for(int k = 0; k < K; k++)
			fprintf(fp, "%f ", (float )WH[j][k]);
		fprintf(fp, "\n");
	}

	// Output output layer weights

	for(int i = 0; i< I; i++)
	{
		for(j = 0; j< J; j++)
			fprintf(fp, "%f ", (float )WO[i][j]);
		fprintf(fp, "\n");
	}
```

```
        fclose(fp);
        return 1;
}
// Function loads a set of network weights to store in WH and WO
// from a file
int LoadWeights(char *WeightFileName)
{
        FILE *fp;

        if((fp = fopen(WeightFileName,"r")) ==  NULL)
                return 0;

        // Read hidden weights

        for(int j = 0; j< J; j++)
                for(int k = 0; k < K; k++)
                        fscanf(fp, "%f", &WH[j][k]);

        // Read output layer weigths

        for(int i = 0; i< I; i++)
                for(j = 0; j< J; j++)
                        fscanf(fp, "%f ", &WO[i][j]);
        fclose(fp);
        return 1;
}
```

Appendix H

Neural Network Test Application

The BPTEST windows application captures user input from a mouse or trackball device, storing an array of a and y coordinates as a 2 dimensional gesture. These are then classified using the neural network functions provided in BP.CPP

```
// BPTEST.CPP

#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\framewin.h>
#include <owl\menu.h>
#include <owl/inputdia.h>
#include <ddeml.h>
#include <owl\static.h>
#include <string.h>

#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <owl/dc.h>

#include "bp2.h"
#include "bptst3.rc"
#include "bptst3.rh"

#define DELAY 2
#define FALSE 0
#define TRUE !FALSE
int Filter = 5;
#define THRESHOLD 0.15
#define MAXGESTURES 8

// Application's main wiindow class
class TDMLSrWnd : public TFrameWindow {
  public:

        BOOL START_REC;             //  Start recording flag set true by Left Button
        int StartX, StartY;          //  Mouse Pos at start of recording
        int TraceElement;               //  counter variable

        BOOL Initialised;               // Network initialized ?
        BOOL ReadyToRecord;             // Ready to record ?
        TGesture Gesture;               // A gesture class holds x and y coordinates
        BOOL Running;                         // Curently recording
        int Classification;             // Variable holds classification value from
network
        int CurrentVector;              // Counter Variable
        float NetOut[MAX_I];            // Array holds network output

        int LastRecord;
        int RUNLENGTH;
        float *GestureArray;            // Array of coordinates
        int *Buffer;
        int GestureID;

        // Gesture timing variables
        clock_t Now, LastTime;
        clock_t GestureStartTime[200],GestureEndTime[200];
        int Run, ScoreCard[200], Target;


        char FileName[30];
        FILE *fp;                   //  Used to save templates

        TDMLSrWnd(TWindow *Parent, const char*);

        // member functions are defined below
        void StartRec();
```

```
        void Capture();
        void Test();
        void Evaluate();
        void InitNetwork();
        void CmAutoTest();
        virtual void SetupWindow();
        void EvLButtonDown(uint, TPoint&);
        void EvRButtonDown(uint, TPoint&);
        void EvMouseMove(uint, TPoint&);
        bool IdleAction(long );
        void CmSetGest();
        void CmSetRunLength();
        void TDMLSrWnd::EraseBox();
        void TDMLSrWnd::DrawBox();

  DECLARE_RESPONSE_TABLE(TDMLSrWnd);
  DECLARE_CASTABLE;
};

DEFINE_RESPONSE_TABLE1(TDMLSrWnd, TWindow)
       EV_COMMAND(CM_AUTOTEST, CmAutoTest),
       EV_COMMAND(CM_SETGEST, CmSetGest),
       EV_COMMAND(CM_SETRUNLENGTH, CmSetRunLength),
       EV_WM_LBUTTONDOWN,
       EV_WM_MOUSEMOVE,
       EV_WM_RBUTTONDOWN,

END_RESPONSE_TABLE;

IMPLEMENT_CASTABLE1(TDMLSrWnd, TWindow);



TDMLSrWnd::TDMLSrWnd(TWindow* parent, const char* title)
  : TFrameWindow(parent, title),
        TWindow(parent, title)
{
//     Initialize all flags, attributes, and netwrk
  START_REC = FALSE;
  Running = FALSE;
  ReadyToRecord = FALSE;
  InitNetwork();
  LastRecord = 0;
  GestureID = 0;
  RUNLENGTH  = 20;
  randomize();
  Attr.X = 0;
  Attr.Y = 0;
  Attr.W = 600;
  Attr.H = 600;

}

// Assign menu in Setup
void
TDMLSrWnd::SetupWindow()
{
  TWindow::SetupWindow();
  TMenu Menu(GetMenu());
  AssignMenu(BPTESTMENU);

}

// Initialze network data structures and load weights
```

```
void
TDMLSrWnd::InitNetwork()
{
	if(!InitDataStructures())
		MessageBox("Network too large !!", Title, MB_ICONSTOP);
	else if(!LoadWeights("Track8.wgt"))
		MessageBox("Error Loading Weights", Title, MB_ICONSTOP);

}


// Gesture coordinates are recorded in response to Windows mouse move
//     messages
void TDMLSrWnd::EvMouseMove(uint, TPoint& point)
{

	HDC DC;
	char s1[40];
	char *blanks="            ";
	int x,y;


	if(TCAPTURE && START_REC)  //  Store Current Mouse Pos (relative to
				   //  to start) in Template of currently
				   //  selected command
	{
		if(TraceElement == 0)
		{
			StartX = point.x;
			StartY = point.y;
		}
		if(TraceElement<(GetVectorLength()/2)) // If all elements not yet
recorded
		{
			x = point.x;		//	Current Mouse Pos
			y = point.y;

			x -= StartX;		//	Adjust relative to start
			y -= StartY;
			y *= -1;

			Gesture.x[TraceElement] = x;
			Gesture.y[TraceElement] = y;

			LastRecord++;
			if(LastRecord >= Filter)
			{
				TraceElement++;
				LastRecord = 0;
			}

			DC = GetDC(HWindow);

		//	Output Coords to provide feedback

			strcpy(s1,"                        ");
			TextOut(DC, 50, 100, s1, strlen(s1));

			sprintf(s1,"%d %d ", x, y);
			TextOut(DC, 50, 400, s1, strlen(s1));
			ReleaseDC(HWindow, DC);

		}
```

```
            else    //   End of Template Record
            {
                  char s[30] = "                                    ";
                  DC = GetDC(HWindow);
                  TCAPTURE = FALSE;
                  START_REC = FALSE;
                  GestureEndTime[Run] = clock();
                  TextOut(DC, 50, 30, s, strlen(s));
                  TextOut(DC, 50, 400, blanks, strlen(blanks));
                  Test();
                  ReleaseDC(HWindow, DC);
                  Run++;
                  EraseBox();
                  TextOut(DC, 50, 10, blanks, strlen(blanks));
                  LastTime = clock();
                  Now = clock();
            }
      }
}

// Function to capture user gesture, requests gesture to be performed
void TDMLSrWnd::Capture()
{
      char TargetString[3];
      char * blanks = "                                                ";

      HDC DC;

        char s[20]="Perform gesture ";
            if(GestureID == 0)
            {
                  Target = random(MAXGESTURES);
                  Target ++;
            }
            else
                  Target = GestureID;

            itoa(Target, TargetString, 10);
            strcat(s,TargetString);

            DC = GetDC(HWindow);
            TextOut(DC, 50, 10, s, strlen(s));
            TextOut(DC, 50, 60, blanks, strlen(blanks));
            TextOut(DC, 50, 400, blanks, strlen(blanks));

            strcpy(s, "X");
            TextOut(DC, 225, 225, s, strlen(s));
            ReleaseDC(HWindow, DC);


            TraceElement = 0;
            LastRecord = 0;
            TCAPTURE = TRUE;
            START_REC = TRUE;
            GestureStartTime[Run] = clock();
            DrawBox();
}


// Test routine captures and classifies a user gesture
void TDMLSrWnd::Test()
{
      char *Class = "    ";
      char Message[35];
```

```
        HDC DC;
        char s1[]="                                                    ";
        char Buffer[10];
        float Uncertainty = 0;

        Classification = Classify(Gesture, NetOut);
        itoa(Classification, Class, 10);
        strcpy(Message,"Classified as : ");
        strcat(Message,Class);

        DC = GetDC(HWindow);

        for(int i = 0; i<GetOutputNodes(); i++)
                if(i != Classification -1)
                        Uncertainty += NetOut[i];

        TextOut(DC, 50, 60, s1, strlen(s1));

        if((Classification == Target) && Uncertainty < THRESHOLD)
        {
                ScoreCard[Run] = 1;
                TextOut(DC, 50, 60, Message, strlen(Message));
        }

        else
        {
                ScoreCard[Run] = 0;
                MessageBeep(-1);
                strcpy(Message,"Not recognised !");
                TextOut(DC, 50, 60, Message, strlen(Message));
                MessageBeep(-1);
        }

    TextOut(DC, 50, 10, s1, strlen(s1));

        ReleaseDC(HWindow, DC);

}

// If we are currently running, call capture routine periodically from
// idle action
bool TDMLSrWnd::IdleAction(long l)
{
        if(Running)
        {
                if(Run == RUNLENGTH)
                {
                        Running = FALSE;
                        Evaluate();
                }
                if(!TCAPTURE && Running)
                {
                        Now = clock();
                        if(Now - LastTime < 0)
                                LastTime = Now;
                        if((Now - LastTime)/CLK_TCK > DELAY)
                        {
                                LastTime = Now;
                                Capture();
                        }
                }

                }
        return 1;
```

```
}

// At the end of a run, evaluate and output results
void TDMLSrWnd::Evaluate()
{

        HDC DC;
        float AverageTime = 0, Slowest = 0 ,Fastest = 1000;
        float Time;
        int Errors = 0;
        char ResultString[200];
        char SSlowest[8], SFastest[8], SAverageTime[8], SErrors[6];

        for(int i = 0; i<RUNLENGTH; i++)
        {
                Time = (GestureEndTime[i]-GestureStartTime[i])/ CLK_TCK;
                AverageTime+= Time;

                if(Time >Slowest)
                        Slowest = Time;

                if(Time<Fastest)
                        Fastest = Time;

                if(!ScoreCard[i])
                        Errors++;
        }

        AverageTime /= RUNLENGTH;

        sprintf(SSlowest,"%.2f",Slowest);
        sprintf(SFastest,"%.2f",Fastest);
        sprintf(SAverageTime,"%.2f",AverageTime);
        itoa(Errors, SErrors, 10);

        strcpy(ResultString, "Average Time = ");
        strcat(ResultString, SAverageTime);
        strcat(ResultString, ",  Slowest = ");
        strcat(ResultString, SSlowest);
        strcat(ResultString, ",  Fastest = ");
        strcat(ResultString, SFastest);
        strcat(ResultString, ",  Number of Errors = ");
        strcat(ResultString, SErrors);


        DC = GetDC(HWindow);

        TextOut(DC, 50, 100, ResultString, strlen(ResultString));

        ReleaseDC(HWindow, DC);

}

// Ste the length of a test run
void TDMLSrWnd::CmSetRunLength()

{
        char InputText[6];

        wsprintf(InputText, "%d", RUNLENGTH);
        if ((TInputDialog(this, "Run Length",
                                                "Set Run Length :",
                                                InputText,
```

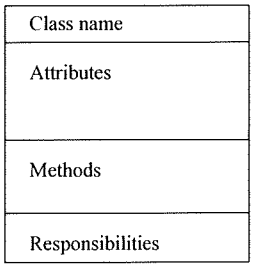
```
                                                        sizeof(InputText))).Execute() ==
IDOK)
        {
                RUNLENGTH= atoi(InputText);
                if (RUNLENGTH < 1)
                        RUNLENGTH = 1;
        }

}

// Function allows a specific gesture to be continuously tested
void TDMLSrWnd::CmSetGest()

{
        char InputText[6];

        wsprintf(InputText, "%d", GestureID);
        if ((TInputDialog(this, "Gesture Number",
                                                        "Set Gesture Number ( 0 for
Random ) :",
                                                        InputText,
                                                        sizeof(InputText))).Execute() ==
IDOK)
        {
                GestureID = atoi(InputText);
                if (GestureID < 0)
                        GestureID = 0;

                if (GestureID > MAX_GESTURES)
                        GestureID = MAX_GESTURES;

        }

}

// A box Draw and erase function exist to provide an area for gesture input
void TDMLSrWnd::DrawBox()
{
        int XGap = 75, YGap = 75;
        int X, Y;

        HDC DC;

        DC = GetDC(HWindow);
        char OutString[2];

        OutString [0] = 127;
        OutString [1] = 0x00;


        X = 150;
        Y = 150;

        for(int i = 0; i<3; i++)
        {
                for(int j = 0; j<3; j++)
                {
                        TextOut(DC, X, Y, OutString, strlen(OutString));
                        X += XGap;
                }
                Y += YGap;
                X = 150;
        }
```

```
        ReleaseDC(HWindow, DC);
}

void TDMLSrWnd::EraseBox()
{
        int XGap = 75, YGap = 75;
        int X, Y;
        int BoxTLX,BoxTLY;

        HDC DC;

        DC = GetDC(HWindow);
        char OutString[2];

        OutString [0] =  ' ';
        OutString [1] = 0x00;

        X = 150;
        Y = 150;

        for(int i = 0; i<3; i++)
        {
                for(int j = 0; j<3; j++)
                {
                        TextOut(DC, X, Y, OutString, strlen(OutString));
                        X += XGap;
                }
                Y += YGap;
                X = 150;
        }

                OutString[0] = 'X';
                TextOut(DC, 225, 225, OutString, strlen(OutString));


        ReleaseDC(HWindow, DC);
}

// Main application class
class TDMLSrApp : public TApplication {
  public:

                TDMLSrApp(const char FAR*  AName = 0):TApplication(AName){};

         virtual void InitMainWindow();
};


void
TDMLSrApp::InitMainWindow()
{
  MainWindow = new TDMLSrWnd(0, "Gesture Performance Test");
}

int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TDMLSrApp("Performance Test").Run();
}
```

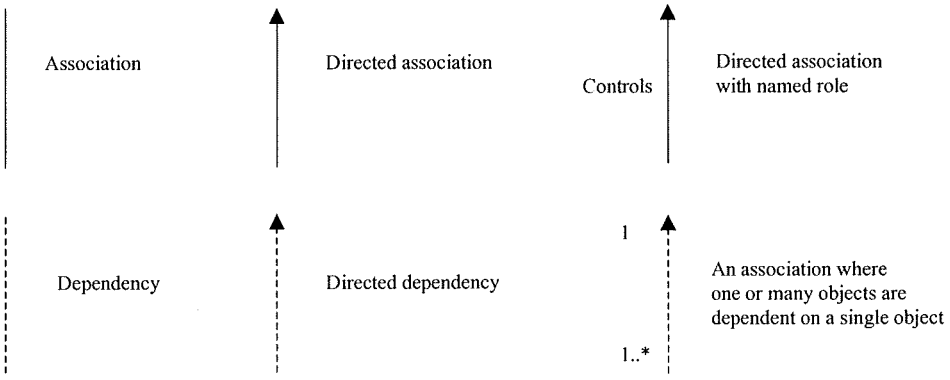
Appendix I

Unified Modeling Language Notation (UML)

Class diagram



Relationships



Appendix J

Evaluation Video Contents

As discussed in chapter 8, video footage was made of the manipulator evaluation. The thesis includes as accompanying material a video containing sample footage. The video shows the author and an evaluator undertaking feeding and drinking tasks with the manipulator.

The manipulator's characteristics as discussed in chapter 8 may be observed in the video, including its appearance, the generation of acoustic noise, and the slow speed of the linear axes. The manipulator does not include the three-axis end-effector discussed in chapter 3. However, the video demonstrates that the tasks addressed were successfully undertaken.

Section 1

Start - 2 minutes

The first section of the video contains footage of the author simulating a feeding task. The semi-structured environment contains a food plate and the arm mounted on a fixed platform. The positioning of these items is not optimized for the feeding task (the distance between is far greater than necessary), but facilitates the video recording.

A modified spoon is attached to the manipulator, and may move through the vertical plane. The task is undertaken using a number of pre-taught positions. The manipulator is controlled using a mouse for direct menu selection, and feedback is provided by a VDU (a window on the VDU is created to simulate the feedback LCD screen).

The feeding task consists of repeatedly acquiring scoops of food from the plate using predominantly the pre-taught positions, with joint movement for fine adjustment if required.

Section 2

2 minutes - 3 minutes 25 seconds

This section shows footage of the feeding task being undertaken by the author using voice control. The footage highlights the slow movement exhibited by the manipulator's linear axes.

Section 3

3 minutes 25 seconds - 5 minutes

A repeat of the feeding task, using simple finger movement monitored by an electrolytic tilt sensor. The input device now employs a scanning system, as opposed to the direct-menu selection used above.

The acoustic noise generated by the linear axes is evident during task completion.

Section 4

5 minutes - 6 minutes

The evaluator discussed in chapter 8 is shown undertaking a feeding task, using the manipulator to feed a pot of yogurt emptied into the plate. Control is achieved through wrist movement activating a scanning system.

Section 5

6 minutes - 6 minutes 30 seconds

The evaluator is shown undertaking the feeding task using voice control.

Section 6

6 minutes 30 seconds - 15 minutes 15 seconds

The evaluator is shown undertaking pick and place and drinking tasks. A semi-structured environment was created including a straw holder, and a wine box with a lever attached to its tap.

The evaluator was required to pick up a straw and place the straw in a cup. Pick up the cup and place the cup below a tap. Pour the drink, and finally pick up the cup. This was achieved using a mouse and direct-menu selection. Joint control as opposed to pre-taught positions was selected.

A temporary two-finger gripper was attached to the manipulator.

Appendix K

Spreadsheet automated Task Analysis

The following visual basic routines act as macros attached to a Microsoft excel worksheet. The worksheet is used to provide a representation of a user task as shown in section 1. The VB routines perform calculations to estimate task completion time by extracting user characteristics and task details from the sheet, these acting as inputs to the functions defined.

1. Spreadsheet Task description

M. <Pick & Place>					Resolve	Issue	Verify	Scan Rate	
	<Set Speed>	Med		av	2.3	2	1.1	0	183.60
	<Goto>	Side	Two	max	2.7	2.1	1.3	0	207.40
	<Move>	Shoulder	In	min	1.95	1.9	0.9	0	161.50
	<Move>	Base	Out	av	1.98	2.02	0.72	0	160.48
	<Move>	Elbow	In	max	2.32	2.15	1.64	0	207.75
	<Move>	Wrist	Out	min	0.85	1.89	0.58	0	112.80
	<Move>	Hand	Out	av	3	1.933	2.516		238.51
	<Goto>	Front	Mid	av	2.313	2.005	1.091		183.91
	<Move>	Hand	Out	av	3.36	2.061	1		233.988
	<Move>	Hand	In	av	1.945	1.945	1.077		168.878
	<Goto>	Home	One	av	1.856	2.263	0.824		168.062
	Report <> Accompilshed			av	1.978	2.02	0.716		160.276

Example spreadsheet representation of a user task completion time estimates are shown for different sets of user characteristics.

2. Visual Basic Routines

```
Dim Resolve, ScanRate, Issue, Verify As Single
Dim Opt1, Opt2 As String

Sub MainRoutine()
   Sheets("Main").Select

   For CellValue = 2 To 13 Step 1

      Resolve = Cells(CellValue, 7).Value
      Issue = Cells(CellValue, 8).Value
      Verify = Cells(CellValue, 9).Value
      ScanRate = Cells(CellValue, 10).Value
      Cells(CellValue, 11).Value = CalcT()

   Next CellValue

End Sub

Function CalcT()
   Dim Step As Single

   Opt1 = Range("D2").Value
   Step = SetSpeed(Opt1)

   Opt1 = Range("D3").Value
   Opt2 = Range("E3").Value
   Step = Step + GotoPos(Opt1, Opt2)

   Opt1 = Range("D4").Value
   Opt2 = Range("E4").Value
   Step = Step + Move(Opt1, Opt2)

   Opt1 = Range("D5").Value
   Opt2 = Range("E5").Value
   Step = Step + JointMove(Opt1, Opt2)
```

```
    Opt1 = Range("D6").Value
    Opt2 = Range("E6").Value
    Step = Step + JointMove(Opt1, Opt2)

    Opt1 = Range("D7").Value
    Opt2 = Range("E7").Value
    Step = Step + JointMove(Opt1, Opt2)

    Opt1 = Range("D8").Value
    Opt2 = Range("E8").Value
    Step = Step + JointMove(Opt1, Opt2)
    Step = Step + EndMove()

    Opt1 = Range("D9").Value
    Opt2 = Range("E9").Value
    Step = Step + GotoPos(Opt1, Opt2)

    Opt1 = Range("D10").Value
    Opt2 = Range("E10").Value
    Step = Step + Move(Opt1, Opt2)

    Opt1 = Range("D11").Value
    Opt2 = Range("E11").Value
    Step = Step + JointMove(Opt1, Opt2)
    Step = Step + EndMove()

    Opt1 = Range("D12").Value
    Opt2 = Range("E12").Value
    Step = Step + GotoPos(Opt1, Opt2)
    CalcT = Step

End Function

Function SetSpeed(SpeedLevel)
    T = SelectCommand("Speed")
    T = T + SelectCommand(SpeedLevel)
    SetSpeed = T
End Function

Function Move(Joint, Direction)
    T = SelectCommand("Move")
    T = T + SelectCommand(Joint)
    T = T + SelectCommand(Direction)
    Move = T
End Function

Function JointMove(Joint, Direction)
    T = SelectCommand(Joint)
    T = T + SelectCommand(Direction)
    T = T + SelectCommand("Stop")
    JointMove = T
End Function

Function EndMove()
    T = SelectCommand("End")
    EndMove = T
```

```
End Function

Function GotoPos(Sector, Level)
   T = SelectCommand("Goto")
   T = T + SelectCommand(Sector)
   T = T + SelectCommand(Level)
   GotoPos = T
End Function


Function SelectCommand(Command)
   T = T + Resolve
   n = SystemDelay(Command)
   T = T + (n * ScanRate)
Rem   If (ScanRate > 0) Then
Rem    T = T + 2
Rem   End If
   T = T + Issue
   T = T + Verify
   SelectCommand = T
End Function

Function SystemDelay(Item)
   Dim n As Single

   If (Item = "Speed") Then
      n = 2.5
   ElseIf (Item = "Med") Then
      n = 3
   ElseIf (Item = "Move") Then
      n = 2.5
   ElseIf (Item = "Base") Then
      n = 3
   ElseIf (Item = "Arm") Then
      n = 3
   ElseIf (Item = "Shoulder") Then
      n = 3
   ElseIf (Item = "Elbow") Then
      n = 3
   ElseIf (Item = "Hand") Then
      n = 3
   ElseIf (Item = "Wrist") Then
      n = 3
   ElseIf (Item = "Out") Then
      n = 2
   ElseIf (Item = "In") Then
      n = 2
   ElseIf (Item = "Home") Then
      n = 2
   ElseIf (Item = "Front") Then
      n = 2
   ElseIf (Item = "Side") Then
      n = 2
   ElseIf (Item = "One") Then
      n = 2
   ElseIf (Item = "Two") Then
      n = 2
```

```
    ElseIf (Item = "Three") Then
      n = 2
    ElseIf (Item = "Stop") Then
      n = 0
    ElseIf (Item = "End") Then
      n = 3
    End If

    SystemDelay = n

End Function



Dim Resolve, ScanRate, Issue, Verify As Single
Dim Opt1, Opt2 As String

Sub CalcT()
    Dim Step As Single
    Sheets("Main").Select
    Resolve = Range("H2").Value
    ScanRate = Range("K2").Value
    Issue = Range("I2").Value
    Verify = Range("J2").Value

    Opt1 = Range("D2").Value
    Step = SetSpeed(Opt1)
    Range("F2").Value = Step

    Opt1 = Range("D3").Value
    Opt1 = Range("E3").Value
    Step = GotoPos(Opt1, Opt2)
    Range("F3").Value = Step


    Opt1 = Range("D4").Value
    Opt2 = Range("E4").Value
    Step = Move(Opt1, Opt2)
    Range("F4").Value = Step

    Opt1 = Range("D5").Value
    Opt2 = Range("E5").Value
    Step = JointMove(Opt1, Opt2)
    Range("F5").Value = Step

    Opt1 = Range("D6").Value
    Opt2 = Range("E6").Value
    Step = JointMove(Opt1, Opt2)
    Range("F6").Value = Step

    Opt1 = Range("D7").Value
    Opt2 = Range("E7").Value
    Step = JointMove(Opt1, Opt2)
    Range("F7").Value = Step

    Opt1 = Range("D8").Value
```

```
    Opt2 = Range("E8").Value
    Step = JointMove(Opt1, Opt2)
    Step = Step + EndMove()
    Range("F8").Value = Step


    Opt1 = Range("D9").Value
    Opt1 = Range("E9").Value
    Step = GotoPos(Opt1, Opt2)
    Range("F9").Value = Step

    Opt1 = Range("D10").Value
    Opt2 = Range("E10").Value
    Step = Move(Opt1, Opt2)
    Range("F10").Value = Step

    Opt1 = Range("D11").Value
    Opt2 = Range("E11").Value
    Step = JointMove(Opt1, Opt2)
    Step = Step + EndMove()
    Range("F11").Value = Step

    Opt1 = Range("D12").Value
    Opt1 = Range("E12").Value
    Step = GotoPos(Opt1, Opt2)
    Cells(10, 9).Value = Step


End Sub

Function SetSpeed(SpeedLevel)
    T = SelectCommand("Speed")
    T = T + SelectCommand(SpeedLevel)
    SetSpeed = T
End Function

Function Move(Joint, Direction)
    T = SelectCommand("Move")
    T = T + SelectCommand(Joint)
    T = T + SelectCommand(Direction)
    T = T + SelectCommand("Stop")
    Move = T
End Function

Function JointMove(Joint, Direction)
    T = SelectCommand(Joint)
    T = T + SelectCommand(Direction)
    T = T + SelectCommand("Stop")
    JointMove = T
End Function

Function EndMove()
    T = SelectCommand("End")
    EndMove = T
End Function
```

```
Function GotoPos(Sector, Level)
   T = SelectCommand("Goto")
   T = T + SelectCommand(Sector)
   T = T + SelectCommand(Level)
   GotoPos = T
End Function


Function SelectCommand(Command)
   T = T + Resolve
   n = SystemDelay(Command)
   T = T + (n * ScanRate)
   If (ScanRate > 0) Then
      T = T + 1
   End If
   T = T + Issue
   T = T + Verify
   SelectCommand = T
End Function

Function SystemDelay(Item)
   If (Item = "Speed") Then
      n = 2
   ElseIf (Item = "Med") Then
      n = 2
   ElseIf (Item = "Move") Then
      n = 0
   ElseIf (Item = "Base") Then
      n = 0
   ElseIf (Item = "Arm") Then
      n = 1
   ElseIf (Item = "Shoulder") Then
      n = 2
   ElseIf (Item = "Elbow") Then
      n = 3
   ElseIf (Item = "Hand") Then
      n = 4
   ElseIf (Item = "Wrist") Then
      n = 5
   ElseIf (Item = "Out") Then
      n = 0
   ElseIf (Item = "In") Then
      n = 1
   ElseIf (Item = "Home") Then
      n = 0
   ElseIf (Item = "Front") Then
      n = 1
   ElseIf (Item = "Side") Then
      n = 2
   ElseIf (Item = "One") Then
      n = 0
   ElseIf (Item = "Two") Then
      n = 1
   ElseIf (Item = "Three") Then
      n = 2
   ElseIf (Item = "Stop") Then
```

```
    n = 0
  ElseIf (Item = "End") Then
    n = 6
  End If

  SystemDelay = n

End Function
```

Appendix L

Publications