MODEL-BASED ROBOT CONTROL AND MULTIPROCESSOR IMPLEMENTATION

A DISSERTATION

SUBMITTED TO THE FACULTY OF ENGINEERING

OF GLASGOW UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

HAMID MIRAB

September 1990

To My Parents

Abstract

Model-based control of robot manipulators has been gaining momentum in recent years. Unfortunately there are very few experimental validations to accompany simulation results and as such majority of conclusions drawn lack the credibility associated with the real control implementation.

In this thesis the main theme is to demonstrate the potential enhancements that are brought about by way of implementing this class of controllers as far as manipulator trajectory tracking is concerned. Particular emphasis is on experimental evaluation to render valid conclusions and to give through appreciation of the merits of various controllers.

The approach taken is to recognise that the realistic role of model based controllers at the initial stage is to reduce rather than eliminate the nonlinear and coupling effects inherent in manipulator dynamics. The level of reduction of these effects is obviously dependent on the modelling effort and the accuracy of the developed model.

Model inaccuracies have two forms: parameter uncertainties that arise from limitations in specification of numerical values for the kinematic and dynamic robot parameters or payload variations; and unmodeled dynamics possibly arising from simplified models.

In particular, two aspects of such controllers are presented and experimentally verified: firstly estimating the mass of the load carried by the gripper to overcome performance degradation due to payload variations and secondly incorporating a suitable self-tuning method in the control system to account for other modelling

inaccuracies.

Consideration is given to the requirements regarding computational powers for real-time implementation of the controllers. These suggest that, to achieve the sampling rates needed, parallel processing and the use of fast processors such as the INMOS TRANSPUTER are appealing. Such processors are used for the experimental implementation of the control algorithms reported here.

The suitability of multiprocessing approach for a robot programming system is also discussed.

In the context of the modelling, the importance of including drive system dynamics in the system model is emphasised and use of CAD-based modelling is addressed.

The use of symbolic algebraic manipulation for obtaining efficient and/or customised dynamic equations for reducing the amount of computational effort and/or formulating the model in a certain manner such as linear in the parameter (for estimation purposes) is presented. Experimental validation of an obtained model and a load mass estimation method developed are included.

Acknowledgements

The author wishes to sincerely express his deepest gratitude to his supervisor Professor P. J. Gawthrop, for the encouragement, guidance, and inspiration conveyed to him during the preparation of this thesis.

Deepest appreciation is also expressed to Professor B. F. Scott and Mr. J. Howell who offered helpful suggestions.

The continuous support and encouragement of all the members of the Control Group, especially Dr. H. Demircioglu, Dr. D. Ballance, Mr. G. Worship, Dr. R. Jones, Y. Mather and D. Sbarbaro, served to lighten the burden of this work.

Thanks are also due to Mr. I. Russell.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The interdisciplinary field of robotics entails the interaction of various subject areas. From an engineering research viewpoint, in addition to each discipline requiring an indepth study, the effect of other areas also need to be considered.
An important issue in this context is the need for the robots to interact with their environment as well as other systems. As a result treating the robot as an isolated unit is simplistic and ignores the importance of communication and coordination.
The determining factors in the operational requirements of a robot are the tasks and applications that the robot is employed to carry out (e.g. assembly, materials handling, welding) which in turn specify the required capabilities such as trajectory tracking, obstacle avoidance, compliant motion etc.
Once these specifications are available, all constituents of what is referred to as robotics are combined, each with a certain level of complexity depending on the specified requirements, to perform the task.
The *manipulator* type is chosen so that its configuration, basic motion capabilities, degrees of freedom, and other physical characteristics match the specifications. Also the measure of performance of the manipulator in terms of load carrying capacity, precision movement (spatial resolution, accuracy, repeatability), speed of movement, environmental requirements (e.g. temperature), operating envelope etc. are suitable for the application.

The *power unit* which also includes the actuators then needs to produce the required movements according to the specifications of the *robot controller* in terms of accuracy, smoothness, speed through command signals that are produced using the informations from *sensors* in regular intervals.
It should be mentioned that the controller itself usually has to be synchronised and coordinated with other systems by a higher level supervisory controller.
Large elements of research interest are directed towards each aforementioned stages and individual areas with the kind and depth of the research being dictated by the needs of the future market.

1.1 Objectives

Although intelligent, mobile, autonomous, and human like robots that are capable of operating in space have their attractions, for more down to earth applications one emphasis seems to be on the production of light weight fast and accurate manipulators that can be used in flexible manufacturing environments with the ability to utilise sensory data of various kinds.
One of the implications of this is the need to further investigate the following:

- Programming and languages for robots with capabilities that extend to cater for the demands of flexible manufacturing systems and complex applications.
- Kinematics and dynamics of manipulators with considerations given to redundancies and flexibility of joints and linkages.
- Real-time novel control schemes for smoother, accurate and fast motions and better performance.
- Sensor technology and how sensors can be effectively utilised for robotic applications.

- Faster data processing schemes and utilisation of mutiprocessor environments for reduction of computation time.

Having identified the above as being potentially high priority research areas, the objectives of the work presented here are: with the resources and time available, first of all, study some of these in detail, secondly after gaining insight, propose, describe, and suggest ways of getting round the problems or introduce new approaches that overcome the difficulties faced, and finally by way of simulation and/or implementation verify the credibility of the schemes and methods.
The approach favours use of a real robot where possible for validation and verification, as attempts to do this in the literature are very few, although it is very valuable to move away from mere simulation and consider and observe factors that can only be studied by actual implementation.

1.2 Motivations

Programming systems and languages in the field of robotics are still far from ideal. Despite the improvements that have been made by robot manufacturers and research organisations to accommodate the demands imposed by the nature of robotics as opposed to applications that general purpose programming languages cater for, there remains much to be desired.
Communications, standards, user friendliness, appropriate semantics, suitable front ends, application specific extensions, use of computer graphics and animation, integration with other systems and finally the question of "a cell language" all need careful detailed examination.
These are the subject of one area presented here.
Moving on from robot languages and programming, both kinematics and dynamic modelling of manipulators are of great interest for research purposes. The dynamic equations of the manipulator are a set of highly nonlinear coupled second order differential equations.

The need for fast, accurate and versatile manipulators has brought about research challenges into the dynamic effects.

Kinematic and inertial parameters of robot manipulators that are part of the dynamic equations, can be obtained by various methods. Although the accuracy of these parameters is of paramount importance in schemes that utilise dynamic equations of the manipulator, comparisons of the methods used have not been effectively carried out and in particular the benefits of employing CAD approaches, have been greatly undermined. This approach will be employed to see its effectiveness.

Efficiency of robot dynamic equations have been looked at and symbolic manipulation has specifically been used to achieve efficiency, but it can be used to gain additional benefits. This will also be addressed in the work presented.

Usually the effects of the mechanical transmission systems, friction, and actuation dynamics are not included in robot dynamic equations for simplicity, however it has been shown that they can be significant. These effects are considered and taken into account in this study.

Only a few research studies have presented model validation using real data extracted form an actual manipulator, this is done in this work by comparing the behaviour of the model obtained with the actual robot.

Research into manipulator controllers in majority of cases have been based on the available theory in the field of control systems engineering with no accounts of requirements specific to robots, and the familiarity of the designers with a particular method has been the only factor in the employment of a scheme in a great number of occasions. Above all, most of the results are based on simulation and very rarely actual implementation has been successfully attempted.

The controller operates at various levels of hierarchy. At the top level it performs planning and coordination, communicates with other devices, and carries out transformation of sensory information. A programming system, sometimes with a degree of intelligence is used to initialise and carry out the commands.

The second level is motion and/or force control. Information from sensors is used in order to carry out the control with a prespecified performance criteria in either joint or cartesian space.

Majority, if not all industrial robot manipulators at this point are controlled by conventional fixed gain controllers based on single-input single-output models for each joint.

These controllers usually provide an acceptable level of performance, as the demand on their speed and accuracy is not high and in addition they are driven indirectly through a gear mechanism by a DC motor, which means that the effect of inertial variations are reduced by the square of the gear ratio.

The adequacy of these schemes is challenged by the emergence of light weight manipulators as well as increased demand on speed and high performance.

A number of *dynamic control* schemes have been proposed, but non has been successfully implemented on a general purpose industrial robot. The main reasons behind this are

- High level of computational requirements makes the real-time implementation on a present day reasonably priced single processor impossible (with one or two exceptions).
- In practice obtaining an accurate dynamic model of the manipulator is not easy.

The measures for responding to the computational demands are, to utilise the specific and usually simple geometric structure of present day manipulators, resulting in simplified dynamic models, and/or enhance the computational efficiency by exploiting the parallelism of the algorithm. In the work presented here both of these issues have been taken into consideration. Multiprocessing powers of the Transputer is utilised as well as its capabilities as a fast single processing unit at a low cost, to implement model based control algorithms that have high computational requirements, and their real-time implementation would have been unlikely

otherwise.

1.3 Structure of the thesis

In chapter 2 *Robot Control Programming Systems*, a comprehensive definition of robot control programming systems is given and a way of evaluating manipulator languages based on this is suggested.

The requirements of the next generation of robot programming systems and proposing that the Transputers, being a fast processor and ideal for multiprocessor implementation of algorithms, and OCCAM as its natural programming system, form a suitable combination for fulfilling these requirements, are also covered in this chapter.

In chapter 3 *Robot Modelling and Validation*, direct and inverse kinematics, and dynamic modelling of an MA3000 robot is presented. The emphasis is on the inclusion of the actuation system (DC motors) models in the modelling process. A CAD approach is taken to obtain kinematic and inertial parameters, and a symbolic manipulation scheme is used to simplify the dynamic equations of the robot.

The behaviour of the model obtained is then compared to that of the real robot, based on real input/output measurements, hence validating the model.

In chapter 4 *Model-Based Identification of Robots*, following a literature review of dynamic parameter estimation of robot manipultors, a method of estimating the mass of the load held by the gripper of the manipulator, based on the state variable filter approach for estimation of continuous time transfer functions is proposed.

Symbolic manipulation is used to obtain a linear in the parameters and computationally efficient form of the dynamic equations of the robot and, least squares is used for estimation.

In addition to simulation, experimental results on the MA3000 robot are obtained

which verify the effectiveness of the method.

Chapter 5 *Transputer-Robot Interface*, describes the transputer systems, the electronic interface units (both voltage and current driven) between these systems and the robot, and the low level interface software.

The interface was made to enable transputer implementation of the controllers that are introduced in chapter 6 on a real robot, as well as for data acquisition for validation of the model of the robot developed and to show the effectiveness of the load mass estimation method proposed earlier.

The fact that the literature contains very few results based on the data obtained from a real robot as opposed to simulation, and the importance of additional factors introduced when a controller is actually implemented on robots, were the main motivations behind taking the approach of carrying out experiments on the MA3000 robot and hence the need to interface it to a fast computer system which allows real time implementation of the controller cost effectively.

In chapter 6 *Control of Robot manipulators*, the methods used for manipulator control both in the commercial scene and the research circles are reviewed and their short-comings as well as strengths are highlighted. Against this background, two new model-based adaptive schemes are proposed, one based on estimation of the load mass and the other a model based variable structure continuous time self tuning controller.

Chapter 7 *Control Implementation, using Transputers*, points out the advantages of using transputers for manipulator control implementation and gives a brief introduction to parallel computation before presenting a detailed account of how a controller based on the computed torque method, and the two new adaptive controllers proposed in chapter 6, are implemented on the MA3000 robot, using a network of transputers. Then a discussion which includes a comparative evaluation of these methods concluded this chapter.

Finally chapter 8 draws some conclusions from the entire work and includes suggestions for future research.

One area that has not been looked at in addition to flexibility of joints and linkages and redundancy issues of robot manipulators in the work here, is sensory devices and their functions in robotics. Due to its importance and the fact that multiprocessing can play a substantial role in alleviating the problems of computationally intensive data processing in three dimensional vision and even its suitability for modular nature of data extraction from various sensors to combine (sensor fusion), a brief outline is given in the next section.

1.4 Sensory devices

A distinction ought to be made between sensors that are employed for manipulator control such as position sensors, and the ones used for object identification and robot guidance such as vision systems. The research into the former is usually in the form of finding ways to improve the accuracy of devices as well as lowering the cost. In the latter case however, the issue is interaction with three dimensional objects and as such to fully capture the information from the workspace, a sensory system capable of sensing and processing large amounts of three dimensional data is required. As a result the computational requirements to process this information in small time scales become prohibitive. A great deal of research effort has been directed towards the use of parallel processing and in particular use of Transputers for their power and cost-effectiveness, to achieve fast processing of vast amounts of data.

Often though, in the context of robot vision for example, the nature of the part or novel illumination or constraints on the part and sensor placement combine to allow a two dimensional rendition to suffice.

In addition for many applications that only require discrimination of similar or nonsimilar objects or information about planar orientation of objects, two dimensional vision based on visible band luminance data is effective enough.

Another interesting area of research is in the field of tactile perception in the study

of different materials and techniques.

For material handling for instance, determination of the position and orientation of objects is essential, plus monitoring of this information possibly requiring incremental analysis and verification of delivery being necessary, and in some assembly operations the requirements go even further including the information about surface features, texture, frictional characteristics and temperature, hence in addition to vision, tactile sensors are needed to cope with these complex applications. Tactile sensors consisting of pressure sensor arrays can provide information about grasped objects and can locate surface features.

The ultimate aim in these studies is to enable the implementation of a closed loop control system for assembly and other applications that includes vision and tactile sensory information.

A whole new area of research known as sensor fusion, deals with combining these sensor information. Applicability of parallel processing and multiprocessor based data acquisition seems natural in this field.

1.5 Contributions of this work

The contributions of the work presented in this thesis are as follow

- Presentation of a comprehensive definition of Robot Control Programming Systems which provide suitable grounds for both evaluation and comparison of robot programming languages. In addition, justification as to why the Transputer/OCCAM combination is suitable for the next generation of robot programming systems.

- Use of CAD modelling and a Symbolic Manipulation scheme to obtain the kinematic and inertial parameters, and develop a dynamic model for a robot. Also validation of the model, based on real data extracted from the actual robot.

- Proposition of a new computationally efficient load mass estimation method and verification of its effectiveness by means of experimental data extracted from a real robot manipulator.

- Interfacing an MA3000 robot to a network of transputers, requiring electronic units and low level software. As a result, real time implementation of control, and data acquisition from the robot were made possible.

- Introduction of two new control schemes for robot manipulators, namely: a model-based adaptive controller with load mass estimation, and a model-based variable structure self tuning controller in a continuous-time framework.

- Transputer implementation of manipulator controllers on a real robot and comparison and evaluation of the results.

Chapter 2

Robot Control Programming Systems

SUMMARY

A robot control programming system is defined in a comprehensive manner. This is useful for evaluation purposes and also for requirement specification of robotic applications. A combination of the Transputer and OCCAM is proposed as being ideal for the next generation of robot programming systems after considering their requirements.

2.1 Introduction

Generally there are two types of approach to Robot Programming, teach pendant and off-line. Teach pendant programming in which the motion of the robot is controlled with a series of push buttons on a hand held device (pendant) is easy to learn. It is also easy to program the robot in more complex geometric situations and when the robot needs to perform a task under load. However the robot remains unproductive while it is being programmed, modular program development is not possible, general purpose library of subroutines cannot be used and the operator is in danger during development phase of the program. As a result

for a highly automated and flexible factory environment, off-line programming seems more suitable. There are also additional advantages in using off-line programming methods, sensors such as vision and force can easily be incorporated, efficient handling of synchronization with external equipment is achieved, repetitive tasks such as palletizing can be programmed with less effort using macros. In off-line programming, the program is developed based on a simulated environment and robots. This results in a number of disadvantages, firstly visualizing a robot path in a three-dimensional space is quite difficult and reachability, collision-free paths and correct orientation are not easy to determine. Also the affect of configuration, controller action, dynamics and inaccuracies need to be anticipated. These drawbacks can be compensated by robot calibration, use of teach pendants to fine tune the programs, run-time sensing and "adaptive" programs, plus use of CAD/Graphics techniques. The problem of learning these languages, being considerably more complex still remains, although attempts have been made to design user friendly front ends, having menu driven commands.

Most of the current robot programming systems are based on a dedicated programming language and consist of a high priority trajectory generator which computes the sequence of joint variables and a language interpreter. The flow of the robot program is synchronized with the actual motion of the manipulator. Three major categories form the basis on which current approaches to robot programming can be classified, although other categories have also been specified. These are:

- *Servo Level* consisting of a series of end points, represented as a group of joint coordinates, speeds and input/output commands.
- *Robot Level* in which a sequence of robot motions are described. Each statement of the program roughly corresponds to one action of the robot.
- *Object Oriented* where a sequence of positional goals of the object to be manipulated or sequence of tasks to be performed are programmed. A task

planner consults a database (world models) and transforms the task specification into a robot-level program.

Three schools of thought have influenced the way in which off-line robot programming languages have evolved over the years. One which argues that for various reasons, it is more appropriate to extend an already existing general purpose programming language and include the constructs needed specifically by robots, the second which favours using an APT-like language used for Numerical Control machine tools. And finally a totally new language which is purpose designed for Robotic Applications. Essentially the issue lies on the emphasis as to the level of experience that the robot programmer ought to possess. Whichever approach is taken, it is widely accepted that the programs need to be

1. capable of allowing computation from sensors and interacting with other devices.

2. able to work in a real-time environment and check conditions to synchronize events or obtain data when it is needed.

3. general purpose and independent of the physical configuration and kinematic features of a particular arm.

4. capable of easy expression of manipulator positions in space and support debugging and testing.

Furthermore the application for which the programming language is to be used, determines the type of attributes that it should have. As robot tasks become more complex and more expensive to implement, the cost effectiveness of interactive graphic simulation and assisted programming will become apparent. The evolution of CAD/CAM integration with robotics and incorporation of ever larger data bases will lead to more advanced systems. The simulation of sensors and generation of sensor programs should be included in the graphical simulation. Also

program utilities should allow planning of such variables as camera angles, position and orientation constraints on objects to be located by vision and timing of vision in relation to robot motion. Most efforts today involving the development of dynamic models are motivated by improving the control or structural performance of the actual robot. However, there is no reason why this could not be included in an interactive graphic simulation to generate robot programs.

In this chapter a description is given as to what constitutes a *Robot Programming System* and based on this, existing languages are evaluated. Then the requirements of the next generation of languages are outlined. Finally a Transputer approach is suggested as a step towards meeting these requirements.

2.2 Description of a Robot Programming System

A Robot Programming System essentially consists of a number of different interacting elements. Each element's position in levels of hierarchy depends on how it can affect and be affected by other elements. In general every robot application requires certain capabilities to be handled by the robot language. Each of these, together with the nature of the robot application will influence the type of programming and operating environment, as well as implementation and language features required. These are the issues addressed in this section. After description of individual levels of robot programming systems and the elements that form them, evaluation of existing languages become meaningful. Also this is useful for establishing the requirements for the next generation of robot languages. At the top-level the Characteristics of the programming system are considered.

2.2.1 Characteristics of the Programming System

There are certain issues which come under this category and each one provides certain capabilities suitable for various robot applications. Hierarchical decomposition of tasks and modular program development, for example necessitate the ability of a programming system to create abstract data structures which represent elements of the problem. This is usually referred to as *Extensibility*. The syntactic issues and semantic power which is available to the programmer to represent his application reflects the *Flexibility* of the programming system. This is a measure of the range of applications for which a language can be used. Another characteristic ties in with the depth to which alternative run-time conditions are handled, for example since syntactic and semantic errors can potentially be identified and corrected prior to execution by the controller in compiled programs as opposed to interpreted ones, they are said to be more *Reliable* in some ways. To cope with the nondeterministic nature of physical interactions with the real world, conditional branching facilities or *Decision Making* can be used. Faster execution speeds can be achieved in some applications, when robot language is able to represent frequently used robotic functions which results in better *Efficiency*. A robot language which is not dependent on a particular hardware has many advantages, as developed programs could be used for different robots. This issue represents the *Portability* of a programming system and can be at odds with the naturalness of a language to express the problem using language features which are specific to the application area. Other characteristics of robot programming systems are: *Usability* of a language in terms of whether the development meets the guidelines with respect to cost and *Maintainability*, and finally how *Sensor Support* is catered for. These are but some of the characteristics that in the view of the author, seem to be important in areas of robot application. Any application requiring some or all of the above, will also put demands on the rest of the elements in the lower levels of the system. The next level is the Programming and

Operating Surfaces.

2.2.2 Programming and Operating Surfaces

This refers to the hardware and software for generation of a robot *Program* and also the *Operating* environment for its execution. As was mentioned in the introduction to this chapter, an important ingredient of off-line program development is the *Simulation* of the robot behaviour. In a larger scale, other parts of a workcell that interact with the robot should also be simulated. Load kinematics, tolerances, sensor delays and flexibility should ideally be represented, as well as analysis of data and material flow within the cell. A graphics system can be used for collision detection, reach testing and reconfiguration of the workcell and task planning so as to reduce the cycle time of a particular operation. Information necessary for the modelling of robots and parts for simulation can be held in a *CAD Database*. Another issue in the programming environment is the level of *Programmer Experience*. That is, whether due to the complexity of programming, a well trained programmer should be employed so that the programming tools and techniques available can be effectively used, or have a simple, easy to use language that an ordinary machine operator can program in. Alternatively a special *Editor* can be used that simplifies the task of the programmer by means of introducing templates of a particular construct or introducing generic commands for a particular operation. In order to solve frequently occurring programming problems, a *Library* of subroutines can be used. Also a *Pre-processor* can be used to convert an extended syntax front end to the normal output language. *Off-line debugging* and various *Programming Techniques* also are elements of the Programming surface. Once the program is generated, it is the responsibility of the Operating Surface to execute it. Different elements can be identified within this level. The scheduling of the robot program statements is mainly provided by the architecture of the hardware and the program itself only partially controls the implementation. The *Sequence*

of Execution of the program is an attribute of the Operating environment. One example of this is *Parallel Processing* in which different layers of control such as supervisory, trajectory processors, I/O handlers etc. act in parallel, or when two or more program statements or procedures are executed simultaneously ie. *Concurrent Execution.* Unplanned events that might occur during the execution of the program, and need immediate attention, such as exceeding the limit of a joint movement, signaled by a sensor can be handled by *Interrupts.* The program itself may up to a point specify *Process Synchronization.* For data transmission, a standard *Networking* system can be used such as Ethernet or MAP, which can be transparent to the robot language. Other elements of this category are *Diagnostics* and *Peripheral Support.*

2.2.3 Implementation of the Language

One of the key issues which is classed in this level is the *Type of Language* used. This is to distinguish between various ways of implementation, and also different representations based on distinctive syntactic and semantic features. Interpreted, compiled or a combination of these. When an *Interpreter* is used, it runs on the target controller and program statements at source level are read one by one and executed directly. The greatest benefits of the languages that can be run with an Interpreter are their power and ease of debugging. Much of the power comes from the notion of delayed binding and dynamic scoping. However, syntactic errors may be encountered at run time, as there is no off-line translation. Also they are slower compared with compiled programs, due to the fact that before execution, interpretation of a program statement requires significant overhead in contrast with when a *Compiler* is used that reads in the higher order language and puts out an object code. Other issues are, whether the language is procedurally oriented or like AML/X and LISP, expression oriented, object oriented (like SMALL TALK) or goal driven (like PROLOG), which are suitable for Task-level

programming approach for Robotic Applications. After issues related to how the language is processed, questions as to how the actual configuration enables certain kinds of *Run-time Debugging* and *System Access* become important. Then the remaining semantic and process synchronization errors can be pin pointed using for example tracing, single stepping, breaking and run-time editor. In work-cell controllers that orchestrate the communication and control of robots, sensors and other devices, access to the system supervisor program is essential. When programming *Multi-Robots*, coordination of the control becomes complicated due to the extra degrees of freedom introduced, as well as timing constraints and in some applications, simultaneous control of force due to contact of two robots. It is important that an accurate simulation of these kinds of operations are carried out before hand, based on complex models incorporating forces and sensors.

2.2.4 Language Attributes

This is at the bottom level of the Robot Programming System and represents a good measure of comparison for Robot Programming Languages. The important elements of this category are shown in the following tables. Individual elements of each feature represented in the tables can influence the elements of the previous levels discussed. As an example, whether static or dynamic scoping (in Declarations and Variables) in table 2.1, is used can affect the reliability of a programming language, ie. static scoping tends to create optimizable and more reliable execution. The need for these elements in a Robot Programming Language is primarily determined by the type of application used. Generally there is a need for representing angles, coordinates, forces, velocities represented in a vector form (in Array) and hence vector operations ie. dot product and cross product etc. Input/Output feature elements are also shown in table 2.1 . In some applications where coordinated control of motion and sensors is necessary, Timers are needed to make this possible. Control Structures are shown in table 2.2 , again empha-

Declarations and Variables	*Array*	*Input/Output*
variable type	vector	binary
variable	frame	analogue
identifier	matrix	vision
label	coordinate system	compliance
constant		guarded motion
declaration		timers
scope		wait
assignment		text i/o
		file i/o

Table 2.1: Some language attributes

sizing the application dependence of the elements. When more than one robot is involved in an operation, a structure for controlling multiple robot arms is needed and perhaps need for parallel processing and parallel execution. Another feature shown in the table is Sub-Programs. Macros can be utilized by means of viewing a large piece of detailed code, as a template with slots that can be filled in according to the invocation for example in palletizing applications. Subroutines can be useful for various activities such as motion, sensing, transformation calculations, or specified actions such as move, rotate, change speed, grab camera image. When large programs are to be executed or the program uses a given subroutine for more than one purpose, Parameter passing becomes a prerequisite. In table2.3 , One of

Sub-Programs	*Control Structures*	*Operators*
macro	branching	arithmetic operators
subroutines	looping	boolean operators
nesting	iteration	relational operators
parameters	multiple arm control	transformation matrix
		frame affixment

Table 2.2: Some language attributes

the features shown is Data Types. In Robotic applications Geometric data types are very important for referencing part features such as holes and so on. Therefore a data type to represent points, lines, planes, curves and surfaces is necessary. Also shown in the table is Motion in terms of move (grasp, stop) with parameters specifying speeds, acceleration, forces etc. and path (straight line, circular, or

along some other geometric curve) possibly in more than one coordinate system. Tool Statements include Effector Command which references grasping, open and closure with defined force for example and Tool Command which specifies a move relative to the gripper held tool frame of reference.

Data Types	*Motion*	*Tool Statement*
elementary	move	effector command
structured	path	tool command
geometric		

Table 2.3: Some language attributes

The elements of a Robot Programming System described above, form a comprehensive representation of the entities, which are important, for evaluation of individual systems, as well as comparative studies of different ones. These will be dealt with, in the next section.

2.3 A comparative study of Robot Programming Systems

Detailed comparison of Robot Programming Systems is a complex task. There are numerous different ways that this comparison can be carried out. In the literature, usually comparisons are either inadequate or unfair. Inadequacy stems from lack of a comprehensive measure and unfairness is a result of comparing systems which are meant to be used for different applications. To overcome the former, all elements of a Robot Control Programming System described in the previous section ought to be used, not just some and also the interaction of different levels should be taken into account. One way of tackling the latter problem is establishing a benchmark for a particular application and then the systems which are designed for the operation can be compared. There has been some effort in this direction, for example the development of a European Benchmark for the comparison of assembly robot programming systems [15] . An insight to the whole issue can be

gained by looking at how off-line languages for robots evolved over time.

First Generation:

The early robot programming systems were created by extending BASIC to provide for robot motion which was usually straight-line or circular or along some arbitrary curve, elementary sensor usage (binary), basic coordination and a minimal operating system which were designed to be self contained. Also they were designed to be usable by operators without computer science training. VAL is an example of this generation. Due to their limitations, they are restricted to certain applications which do not need complex computations or interface to complex sensors. Communication with other computers and controllers are limited and they are not extensible in a way that allows one to add commands and build capabilities into the language. Although these languages remove the rigidity of augmented teach box software, they provide very little new capabilities.

Second Generation:

To overcome the limitations of the first generation, computational complexity of a modern structured computer language were built into the second generation Robot Programming Systems. RAIL for example looks very much like its parent PASCAL, while AML embody new concepts. They have extensions for motion, sensor communication, improved control and operating systems with more powerful editors and file handling capabilities. Nevertheless many of them are supplied as closed systems which restricts their capabilities for computer communication and coordination. The main advantage lies in their extensibility. Through the use of subroutines and functions, new commands can be added to the language that give capabilities transparent to the user, which then means operators can easily use them. The second generation languages also have some weakness. They cannot yet deal with part or cell geometry as is necessary in the work environment (with the exception of APT-like robot languages). There is still no way to safely debug a robot program. Robotic applications should be able to access the data

that resides in a CAD data base, but so far very few can handle the communications, the different data structures, or the interpretation of geometrical data. Similarly vast majority of them do not facilitate the integration of robots into Flexible Manufacturing Systems (FMS) which use computer control to coordinate combination of NC tools, robots, inspection stations, material handling systems. In addition significant improvements are needed in the use of robotic software.

Third Generation:

These are usually referred to as task-level or object level languages. Instead of step-by-step specification, simulation and graphics are combined to direct a robot to perform a series of tasks, which the software would then determine how to carry out in detail. The statements correspond to high-level tasks which the program decomposes into executable actions. Although desirable, they have not yet matured. There are two fundamentally different theoretical approaches to the development of task-level programming. One relies on the artificial intelligence for an expert system solution that will perform tasks according to formal rules gained from human experts. In the other approach, the task is analyzed from the top down and is hierarchically decomposed into subtasks that the system knows how to perform. The above classification can at least help in the choice of languages with similar capabilities for the purpose of comparison. The usual comparisons which have been carried out are based on the features of the programming system, for example [105]. These features usually include: Type of Geometric Modeler (Solid etc.), Robot Modeler (kinematic etc.), Programming Language(Manipulator level, textual ...), Other Features (Reach Testing, Cycle-time etc.) and Graphic Simulation. It should be noted that comparisons based on language features can easily be done by referring to language manuals, where as when we move further to include all levels of the programming system for example operating environments, it is somewhat dependent on the actual implementation. Detailed comparisons of fourteen languages were made by means of developing a sample program and using different languages by [10]. The intention was to derive some measure of

programmability. They included a number of elements of what we defined as Programming System in the previous section, but not all. One of the measures of programmability was the number of instructions in the program, excluding comments, which is quantitative. Other measures were quantified against defined scales, These measures were: Development time, Readability, Understandability of Instructions, Structured Format, Flexibility of choosing variables, Ease of extension, Range of users, Programming complex tasks, Necessary support facilities, Computer power, Sensing ability, Availability. A different work which again included a portion of the categories which form a Programming System were carried out by [28]. A study to define the characteristics of a good programming system was carried out by [11]. They came up with a list of quality attributes almost identical to language capabilities (explained in the previous section) that affect the life cycle cost of a robot program. For assembly applications in particular [15] developed an assembly application which they executed on different robots and concluded what types of entities are desirable to support such applications. One outstanding result was that different levels of programming system can not be considered in isolation. They also made a remark that comparison should not be made in conjunction with a representative range of manipulators.

After studying the literature, and seeing the shortcomings of the comparisons, an effective evaluation of Robot Programming Systems, in author's opinion is that, once an application is specified, its requirements in terms of sensors, decision making, communications, motion, world modeling etc. ought to be detailed, and then a class of systems with the inherent required features, capabilities etc., should be selected for comparison. A comprehensive pseudo code which entails all the requirements should be prepared and then used for programming the application. A scaled quantitative measure against each of the elements of the Robot Control Programming System (RCPS), will then be indicative of the suitability of each system. As an example, consider a typical assembly operation, such as assembly of a flange. The components which should be provided, to obtain a fast and flexible

programming system for the above operation are pointed out in [9]. These are:

- Installation of sensors for vision, force-torque, slip, proximity, etc., into the robot.
- Control of the robot by a run-time system which is able to adapt itself to on-line changes during assembly.
- The compiler to translate the programmed workcycle should have a component which can automatically generate missing information.

The tasks that have to be explicitly programmed, which require the above components, can easily be identified, by noting what instructions is carried out, if a human operator were to assemble the flange. Firstly, dimensions are transferred from the drawing to the actual assembly object, using vision, and translating the information. Secondly, past experience is used to supplement missing information, such as information about the position of the insertion holes. Thirdly, a sensor controlled positioning operation is performed, for example, to insert a screw into a hole, vision is used for coarse positioning, and touch for guidance and fine positioning, hence corrective action, if the thread of the screw does not engage with that of the hole. Fourthly, missing assembly element tasks are supplemented, for example, use of a screw driver for insertion of screws. Finally, fixturing which is needed during assembly may automatically be done without explicit instructions from the drawing. Now the requirements are known, robots and their associated programming systems that are suitable to fulfill the requirements are chosen for comparison. For simplicity in this case, take only two programming systems, VAL II and AML/X. Unimation Inc.'s VAL II is used for operation of UNIMATE and PUMA robots and is referred to by Unimation, as a robot language and control system with expanded computer logic and advanced communication capabilities. Three levels of robot control systems exist within VAL II: First level offers manual teaching capabilities and front-panel programming, second, allows

development of robot programs written in simple robot programming languages, and third, allows modification of the arm's path from data transmitted through external sensing devices. VAL II can be interfaced to a supervisory system through the Digital Data Communications Message Protocol (DDCMP) used by DEC in its network communication. This protocol provides error checking and automatic message transmission, as needed for factory communications. VAL II also allows a second application program to run concurrently with the main robot control program for the purpose of process control. AML/X is a major revision of AML and is a general purpose programming language for manufacturing and computer aided design, by IBM. It has features for conventional data processing and also object-oriented. An AML/X program is a series of textual expressions evaluated by an interpreter at run-time according to specific rules. Programming languages such as LISP and SMALLTALK have influenced its design. Various application layers can be developed residing on top of AML/X, as well as each other, hence the end user only sees a very small application specific language. Both of the above languages should now be compared on the basis of individual elements of the Robot Control Programming System (RCPS) shown in table 2.4. This could be done, both by looking at the documentations available for each system and also programming the assembly operation. An order of preference can then be used to describe the merits, for example ranging from non-existent ($\times$), available ($\surd$), poor, acceptable, good, very good to excellent. This is shown for some elements in table 2.4. As far as *Flexibility* is concerned, although VAL II has considerable flexibility in motion control with guarded moves, real-time trajectory updating, and watchdog monitors over sensors, AML/X seems superior. This is due to its unique data abstraction, and its debugging procedures for handling exceptions, which result from unplanned events at run-time, etc.. Also lack of built-in motion or sensor primitives, means that it can conveniently work for a great range of robots and sensors. These primitives can be defined according to the robot application and there are some developed definition and implementations, for example

RCPS elements	*VAL II*	*AML/X*
Extensibility	good	very good
Flexibility	acceptable	excellent
Reliability	good	very good
Decision Making	good	very good
Efficiency	acceptable	poor
Portability	poor	very good
Usability	acceptable	very good
Maintainability	very good	very good
Sensor Support	good	good
Simulation	×	×
CAD Database	×	√
Programmer Experience	implementation needed	implementation needed
Editor	good	good
Subroutines Library	acceptable	very good
Pre-Processor	×	×
Off-line Debugging	good	poor
Programming Techniques	implementation needed	implementation needed
Parallel Processing	×	×
Concurrent Execution	×	×
Process Synchronization	√	√
Networking	acceptable	acceptable
Interrupts	√	√
Diagnostics	√	√
Peripheral Support	√	√
Interpreter	×	√
Compiler	√	×
Run-time Debugging	poor	very good
System Access	√	√
Multi-Robots	?	?
Variables	implementation needed	implementation needed
Data Types	implementation needed	implementation needed
Input/Output	implementation needed	implementation needed
Sub-programs	implementation needed	implementation needed
Control Structures	implementation needed	implementation needed
Operators	implementation needed	implementation needed
Motion	implementation needed	implementation needed
Tool Statement	implementation needed	implementation needed

Table 2.4: Comparison of AML/X and VAL II

AML/2 for IBM 7575 and 7576 robots. For this element, VAL II is acceptable, but AML/X seems to be Excellent. In the case of *Extensibility*, in AML/X, user-oriented front-ends can be used to extend the capabilities, due to the freedom for data abstraction. VAL II is capable of controlling various robots from ones used for clean-room applications and ones for harsh environments, and with the introduction of VAL DATA Products, CATVAL postprocessor for interface to CATIA and PCVAL supervisor in 1987, its extensibility was demonstrated. In this case AML/X can be said to be very good and VAL II, good. For *Reliability* comparison, VAL II's combination of teach mode and textual programming, makes run-time debugging more precise, but the extremely sophisticated exception handling capabilities of AML/X makes it much more reliable. The lack of sophisticated conditional branching, makes explicit error handling difficult and for VAL II, run-time failures, will cause problems. considering *Efficiency*, VAL II is reasonably fast and allows for multiprocessing. The limitation of AML/X is that having to go through an interpreter, time-critical applications need to be written in C and called through its C interface. Regarding *Maintainability*, they both score well, AML/X with its data abstraction and self documenting style and VAL II with its motion sequencing being kept away from the location and trajectory data. In terms of *Portability*, AML is very good since the interpreter is written in the general purpose C language, whereas VAL II is not portable as such.

Similar type of evaluation can also be applied to other elements. It should however be noted that, language attributes and some elements of other levels, are best assessed, if programs are actually implemented, so that aspects like ease of programming can be taken account of.

Other languages can also be compared, in the same sprit, based on the requirements of the application, or even a universal application characterized by a benchmark, and evaluation of individual elements of RCPS. Now an effective approach for comparison of programming systems, has been suggested, specification of requirements and future trends will be dealt with in the next section.

2.4 Requirements, Future Trends

It will be too simplistic to assume that a generalization can be made as to the requirements of Robot Programming Systems. Some applications may essentially require certain capabilities and attributes, that might only be desirable or additional to others. For example for a complex assembly automation, the robot system must be equipped with sensors and interfaces to other machines or tools. Therefore the robot language should include facilities to define data structures and input/output actions. However, assembly work covers a wide range of tasks, varying from relatively easy tasks such as transfer of a part which do not necessarily require sensors and interfaces, to precision work such as tasks that involve interaction of more than one robot with different parts at the same time. Decision making, Sensing, Communication, World modelling are some of the categories that make up application features. The increasing complexity of robot applications, and as a result, the need to include complex sensors and integration with CAD/CAM systems, plus the move towards Flexible Manufacturing Systems for small and medium batch productions, make off-line programming as opposed to lead-through programming essential. The off-line programming system requires the existence of a theoretical 3-D model of the robot and its environment, so that real-life behaviour of the robot can be simulated. In addition to this, knowledge of the task or process to be programmed and verification of it is necessary. A user friendly programming interface is also needed to allow efficient development of the programs. Although application is really the driving force behind the language requirements that need to be met, sometimes cost effective versatility can be achieved by inclusion of extra features that improve the performance of the system. It is generally agreed that a robot which is capable of Straight line motion in one direction, controlling the gripper, responding to external signals and originating output signal is able to perform basic assembly operations [15]. The robot assembly performance can significantly be improved in terms of being able

to perform a range of assembly operations, by addition of some desirable features, such as tactile sensing, servo control of the gripper etc.. Other features can also be added to assist the programmer in making full use of the machine, such as diagnostics, computational ability, software maintenance, CAM compatibility. There are three main areas that are worth considering when a needs analysis for a programming system has been carried out and requirements for a particular application have been specified. These are : Interfaces and Standards, Simulation and Data Bases.

Interfaces and Standards

Both issues of a *Robot Independent* programming system and the *Communication* between the robots, sensors and other constituents of a manufacturing cell, make standardization desirable for the sake of efficiency. The main problem that robot industry has faced, is one of deciding at what point the language instructions are to be translated from robot independent instructions to movement instructions, required for execution by a particular robot. In addition, robots of different configurations and geometries that are employed in non-similar workplaces, will need programs that are specifically written for them, even though the commands might be the same. Complex and subtle geometrical reasoning is demanded by any robot independent programming scheme to take into account the location of different parts of the arm, fixtures, permanent parts of the workcell and the temporary locations of the parts in process. One suggested approach to deal with this, is to capture the geometry of the robot and the workspace in tables which the task program can then access at compilation. The solution to the interface problem has also been hindered, due to lack of agreement for definition and format of data elements (such as commands, feedback variables, sensory data parameters) that need to flow between computing modules. A great deal can be learned from the standards that already exist in the field of Manufacturing and CAD/CAM, although they are not directly suitable for robotics. MAP and Initial Graphics

Exchange Specification (IGES) are but to name two. The specification established by IGES, permits the compatible exchange of product definition data used by various CAD/CAM systems. The methodology for representing the data is extensible and independent of the geometric modeling methods used. Geometric, topological and non-geometric (e.g. data organization) product definition data are represented in file structure and language formats. An IGES file contains five subsections which must appear in order. These are: start section, global section, directory entry, parameter data and terminate section. In this standard, the communications file structure and format are defined, but the specific features and protocols for communications media are not included. There has been some investigations into the implementations and requirements of Programming System standards, by a European effort and also a Japanese proposal working within the Computer Aided Manufacturing - International (CAM-I) framework with some initial agreements.

Benefits can also be gained by standardizing Control Systems, but again to what levels of control this should be applied to, is still to be agreed upon. For sophisticated control implementations, feedback data is needed at a variety of abstraction levels. Control loops can have variety of loop delays and predictive intervals, so for example force and velocity data used in servo loops for high speed or high precision motions can be processed and introduced into the control system very fast, whereas decisions at higher levels (based on vision data for example) need to be made less frequently, and therefore the greater amount of sensory processing can be tolerated. The control level dependence of decision making and the uniqueness of procedures executed at each level by the computing modules, necessitates separate subsets of a programming language at each level which could well have the same logical structures. For *Program Format Standards*, a view held by many working in this field is that the useful precedence set by designers of Cutter Tool DATA (CLDATA) in Numerical Control (NC) ought to be followed,

as the standard will be based on existing NC technology with which manufacturing engineers are familiar. However, there are vast differences between machine tools and robots, which have made the expansion of CLDATA a formidable task. A proposal by the working committee of the German Engineering Association VDI seems to be the only applicable standard based on CLDATA. It is called Industrial Robot DATA (IRDATA) and its general structure, record types and transmission definitions are documented in [95]. The idea is that a user program can be compiled and translated into IRDATA-code and after transferring the program as an IRDATA-text to the robot controller, it will be executed by an IRDATA-interpreter. Therefore the language can be used for any robot control which is supported by an IRDATA interpreter. This is illustrated in Figure 2.1 . It allows for description of workspace, coordinate types, position, velocity, ac-

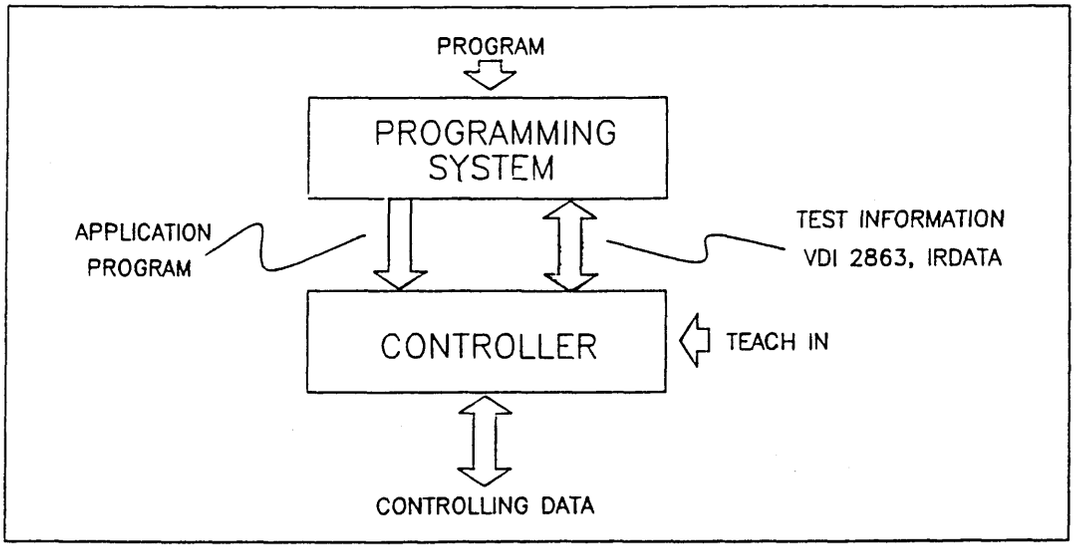


Figure 2.1: IRDATA within a Robot Programming System

celeration, limits, time of motion, guarded motion, synchronization, and modular activities. Parallelism which reflects the predicted change from the use of traditional Von Neumann Computer architecture is also supported.

The general consensus seems to be that experience must be gained over more applications, and technology should be more mature in order to be able to incorporate appropriate requirements into any standard, being at task level or servo

level. One point to note is that attempts to create a standard programming system which is independent of both robot and applications, can end up being so complex that might inhibit the functional use of the system.

Simulation

When programs are developed off-line, in effect models of robots and the workcell environment are assumed, in order to be able to input location values, and movement commands. A graphic simulation support which includes animation, can be a valuable tool for the programmer that provides robot program development, program editing and debugging. In this way, danger to the operator is minimized, while the robot can carry on performing a task during program development. It is obvious that, the more capabilities built in to the system, the more realistic and closer to the real life it will be. There are a number of additional benefits that could be gained, by using a graphic support with the relevant features, some of these are:

- Collision prevention
- Checking for motion constraints and reach testing
- Shortening of an operation's cycle time by utilizing the position of parts and equipment within the workspace
- Analysing the effect of employing different types of grippers

There are graphic simulation packages that include the above capabilities.

GRASP (Graphical Robot Applications Simulation Package by BYG SYSTEMS LTD. Nottingham, UK) for example is a computer graphics simulator designed for robotic cells, with wide variety of robots working in many different situations. Kinematic behaviour of individual robots can be analysed, as well as their interaction with each other and the environment. A high level GRASP program can be created by defining the joint structures, constraints and other data associated

with the robot, as well as the tool path, and then be converted to a robot control language (e.g. VAL II) using a postprocessor.

The Robotics module of CATIA (by Dassault systemes, Suresnes, France) is also an interactive graphics tool for robotic workcell designing and off-line robot programming, with similar capabilities. Although using these graphic simulators, Robot Performance can be studied on the basis of workcell characteristics, for a more realistic model of the robots and the environment as a whole, to allow a more practical simulation to be carried out, the following need to be included:

- complex models of different types of sensors (tactile, vision, etc.)
- complex robot dynamic models which vary even with each copy of the same robot and include coupling, gravitational and inertial effects
- models of joint actuators and control systems used
- non-infinite acceleration and deceleration (to show overshoot errors)
- models of backlash and slop in joints and compliance of links
- model of deformation due to collision etc.

There are simulation packages that include some of the above. I-DEAS Mechanism Design and Excitation Definition Module (by SDRC), allows inclusion of inertia properties and auxiliary functions with motion definition of other joints as a function of one joint, as well as obtaining and inputing load data. It is based on Mechanism Design Theory [79] and the mechanism is described by a set of algebraic equations which must be satisfied during all phases of motion and are based on the identification of all independent loops in the system. Qualitative data is generated in the form of function response (XY) plots and forces within individual joints can be calculated.

ROSI(RObot dynamic SImulator by Cambridge Control Ltd. England) includes simulation of actuators and control. Its Dynamic engine uses Walker-Orin

and recursive Newton-Euler algorithms to perform its dynamic computations. Both the above have limited graphics capabilities, no inverse kinematics and they produce no output code for use by another user written program.

Academic research has also been active in this field, Robot Arm Dynamic Simulation Package (RADSP) by Mech. Eng. Dept. Surrey University is an example of such activity. This is a simulation program for static and dynamic analysis of the performance of a multi-arm robot. Free body method is used for generating the dynamic equations for the manipulator system. Inverse kinematic, path profile planning of the load and torque generation are included as different modules. Either lumped or distributed parameter models of the robot structure may be accommodated as well as linear or non-linear friction effects at the joints. Using this package, steady state characteristics of robots can be assessed. Another academic oriented robot programming system, which includes graphical simulation is RAPT by University of Edinburgh, which is outlined in [1].

Although these are first steps towards creation of a comprehensive graphics simulation package, there is still a lot to be done, before this can be a reality. Even when all the shortcomings pointed out above are dealt with, for the simulated code to be used as a control program, a great deal of sensory information is needed to compensate for the errors. It is however true that the direction is towards creation of an object level programming environment, which graphics simulation and animation play an important role.

Data Bases

The type of information that the robot controller can use to enhance its capabilities and also allow it to be efficiently included within an integrated manufacturing control system, are: Three Dimensional CAD data, information about parts' materials, weight and density, flexibility or rigidity, the forces they can withstand, sensitivity etc.. A part data base can be created to withhold these information and from there, data can be ported to the robot software system, when required.

Artificial Intelligence methods can be used to describe objects, structures and action sequences, in order to prevent time consuming definition of every robot movement with the attributes and attribute values. The working sphere of the robot, the obstacles and other features of the robot environment can be included in a Knowledge base implementation, based on a special relational database. These data bases can be updated at run time with the help of sensors. Therefore structure and data manipulation, evaluation, error monitoring and correction can be achieved.

2.4.1 Future Trends

To discuss the *Future Trends* for programming systems, it is valuable to look at the demands of likely future applications, the deficiencies in the existing off-line programming methods and the feasibility of further enhancements. As the demand for employing robots in more complex applications grows, attention must be paid to increasing the capabilities of robots in terms of sensory information, decision making and more accurate control strategies. As a result, within the programming system context, provisions must be made to accommodate complex sensory data processing, use of artificial intelligence techniques, and control algorithms that are capable of utilising the sensory data to achieve an improved implementation of complex tasks. Off-line programming becomes essential, especially for small or medium batch production and when emphasis is on flexible manufacturing, hence complex geometric and robot modelers ought to be used for benefits discussed earlier.

A general representation of various modules, and their interactions within a robotic manufacturing cell in Figure 2.2, illustrates the layout of the future cells, in terms of programming system needs (e.g. on-line world model etc.) and employment of multi-robots, complex sensory devices, various machine tools, and means of transportation. It seems that the emphasis is towards more autonomy

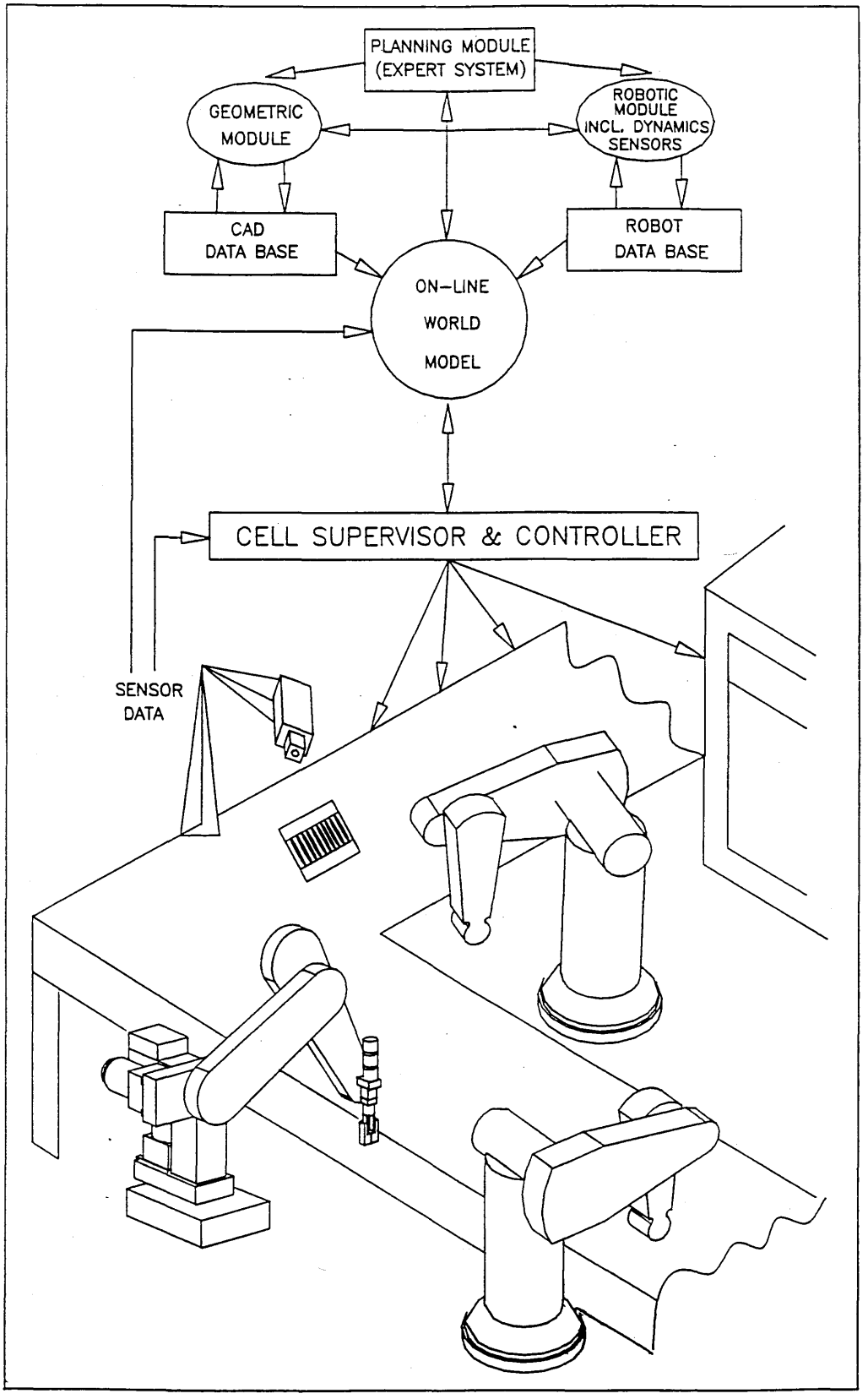


Figure 2.2: Robotic Flexible Manufacturing Cell

with the aid of AI techniques, and improvement of individual modules. In essence improvements can be made in data capturing methods of Geometric modelers, away from the error prone and time consuming manual approaches. Existing data stored within a CAD system can be utilised more efficiently and algorithms used by the robot modeler can be incorporated.

It is difficult to make a general decision as to what kind of approach is most appropriate for the Robot modeler. A robot specific approach, leads to a simple implementation, but limits the scope of application. Alternatively, if the approach is extended to a limited class of structures or even further, to more general complex manipulator types, the complexity resulted specially when manipulator dynamics and other desirable capabilities outlined earlier is incorporated, certainly makes the issue of likely applications, a relevant one. The same dilemma applies in the case of programming methods, but due to the differences in the functional requirements and robot techniques for various applications (compare arc welding, spray painting, and assembly). Modularization can be a possible answer to allow for the efficiency of the whole system.

Logical movement sequence definition (incorporating robot commands, robot functions, and cycle logic) and storage within a programming method framework, can be based on the robot control data within a world model provided by the geometric and robot modeler. An important issue which is worthy of much attention, is dealing with the implicit differences between an idealized theoretical model and the real world. Although incorporation of sensor technology to an extent alleviates the discrepancies, these differences which stem from various sources, ought to be minimized. The robot manipulators should be constructed with tight tolerances and where appropriate rigid structures or allow for compliance within the model. Controllers need to have sufficient resolution and numerical accuracy which can be achieved by efficient algorithms and long word length microprocessors. Numerical accuracy of the modelling and programming system and high quality of real-world model data should also be ensured. Furthermore, the environmental effects within

the workspace such as temperature variations should be taken into account. Finally, multi-robot support, time-based programming with communication between different program modules and system elements, improved human interface, are also some enhancements foreseen for future robot programming systems.

2.5 A Transputer Approach

In an integrated manufacturing system, with multiple robots, complex sensors, machine tools, AGVs etc. interacting and communicating with various computing modules which carry out task scheduling, data processing, complex computations, and so on, the inherent system parallelism can easily be seen at various levels. As with any other parallel system, issues of modular software development, synchronization, communication, efficient data transmission need to be addressed. Within the context robot programming languages, three levels of concurrency can be defined:

1. High level synchronization of both manipulators and programs
2. Response to asynchronous events from the environment
3. Monitoring of sensors

In a wider approach, within the computational hierarchy of a robotic cell, while on the one hand, task decomposition is being carried out at various levels in a top-down fashion, on the other hand sensory data needs to be processed at the same levels, but from bottom to the top. Meanwhile, both of these tasks need to exchange data with the no-line world model. Therefore a minimum of three parallel tasks exist, which can then be further divided into subtasks according to the levels of control hierarchy. There are three control levels, task level, manipulator level and servo level. At the top of the hierarchy, functional and object oriented programs can be used to input robot commands to the plan generator. Then at the manipulator level, a good approach has been to use procedural programms

for calculation of inverse kinematics and dynamics, plus trajectory generation. Finally assembly programs at the hardware interface level, have normally been used for servo control. At this low-level, sensory data processing consists of filtering and scaling joint variables, which are then combined to give data, in required frames and relative to the desired trajectories at manipulator level. At the task level, sensory data tend to hold information about surfaces, volumes, and position and orientation of objects. Once all the subtasks of sensory processing, on-line world model and task decomposition at various levels, are specified in a parallel manner, they need to be coordinated and relevant subtasks need to communicate with each other and other modules to perform motion planning, factory level coordination, task monitoring, error recovery, etc.. To implement this parallel set up, in a cost effective way, the use of INMOS Transputers with their general purpose architecture for multi processing environment, and OCCAM language that makes use of the power of the transputers in a multi processor systems, seem very appropriate. The current top of the range T800 Transputer is a 32 bit processor, designed with a RISC-like architecture. At peak, it can operate at 20MIP instruction rate. The overall throughput is however quoted at 10MIP, due to the fact that some instructions can take longer than a single cycle (50ns) to execute. There is 4K bytes of on chip memory which is matched to the speed of the processor and can be accessed in 50ns. An external memory interface allows access to up to 4G bytes of addressable off-chip RAM. There are 4 serial links which allow data from one transputer either in internal or external memory be DMA'ed into another transputer's memory, in parallel with program execution. OCCAM allows, parallel link communications, using channels, and both parallel and sequential process execution. The existence of natural constructs, such as input (?) and output (!), on the synchronous unbuffered channels, means that there is no need for constructs such as semaphores, and process synchronisation and mutual exclusion. The processing power, the memory and the communication links, plus

the on chip floating point unit, internal timers, support for run-time error diagnostics, high performance graphics support, and external event interrupts, as well as its superior cost/performance characteristic, make the Transputer particularly attractive. The computational speeds achievable, allow the possibility of more time-critical, demanding advanced control algorithms to be implemented too, as will be illustrated later in the thesis. All computational needs of a workcell could be fulfilled, both in areas where large volumes of data need to be processed very fast (e.g. CAD, where independent sets of data are processed), and also where communication, monitoring and synchronisation is required (e.g. sensors). Device interfaces can be implemented, using the event pin of the Transputer and memory mapping. Definition of communication with memory mapped devices is provided by OCCAM through the use of PORT which is used as a channel. Signals can be sent from devices, using the EVENT channel of the Transputer. Transputer Graphics Systems, are available in the market and they can be utilized for simulation, modelling and user friendly front ends for program development. All the robot programming system requirements outlined in the previous section, can suitably be met, by employing a Transputer based system. Within the comparison scheme framework suggested, including all the elements of RCPS, the benefits of a Transputer approach can also be seen easily. For instance, *Programmability* of the system is improved compared to sequential approaches, and generally other systems, for the following reasons: Using the Transputer and OCCAM, gives the benefit of modularity, and can reduce the relative complexity of the software and cost of production. OCCAM is a very expressive language and with its features designed for concurrent systems, is very suitable for expressions of control. *Extensibility* is also improved, as it is easy to add processes, using high-level OCCAM constructs, and extend the processor power, by adding more Transputers. The ease of changing the topology of a Transputer network contributes to *Flexibility* of the system. The existence of powerful debuggers, means that detection of faults are easy, which helps *Maintainability*. In an overall comparison, regarding other

elements such as Efficiency, Process Synchronization, Networking, Subroutines Library, Peripheral Support, etc., the Tranputer OCCAM combination will also score higher.

Further discussion on the Transputer and occam is presented in chapters 5 and 7.

Chapter 3

Robot Modelling and Validation

SUMMARY

An efficient and reasonably accurate dynamic model is developed for an MA3000 robot that includes the actuation systems. A CAD system is used to obtain the kinematic and inertial parameters of the model accurately. Using data extracted from the actual robot, the behaviour of the model is compared to that of the robot and results show a close match between the two.

3.1 Introduction

A series of links connected together by joints, form the building blocks of Robot Manipulators. Link movements are caused by actuation systems, under the robot's controller command, using transmission elements from the actuators to individual joints. There are various types of robots, classified on the basis of their anatomy, arm geometry, actuation types, etc. A Robot model, if created accurately enough, can help minimise the trajectory errors that might occur in executing a desired motion. This can be achieved by predicting the actuator commands from the robot model, given a particular trajectory, and then feeding the information forward in a control loop.

The model of the robot includes the characteristics of the actuation system, the

kinematic parameters, the inertial parameters and finally the dynamic equations of the robot which includes the joint interactions and all the forces (torques) involved.

Although there are various ways to carry out the modelling for each component, using different techniques, it is not always clear which method is the most suitable. To a certain extent, it depends on the availability of resources, and the level of accuracy required. The kinematic parameters, can easily be found from the blueprints of the robot manipulator. For finding the inertial parameters however, either manual methods, parameter estimation, using appropriate sensors for acquiring measurements, or CAD approaches, are the alternative methods.

The dynamic equations can be based upon Newtonian mechanics or Energy approaches. The formulation of these equations will have the same final results; but the representation of these equations and their structures will vary, making them suitable for different applications. Derivation of these equations by hand, when the number of linkage elements exceeds two, is time consuming and prone to human error. Computer assisted formulation of these equations make the task substantially easier and the equations more accurate. Symbolic manipulation programs can be used, together with simplification techniques, to create efficient formulation of manipulator dynamics. These equations are very appropriate for use in real-time.

The final component which needs to be modeled is dynamics of the actuation system. This is not as complicated as the link dynamics modelling, due to the fact that there is no cross couplings between actuation systems; but it is just as important. The actuation is either by means of electric motors, hydraulic, or pneumatic drives. Although data from the manufactures that can be used in parameterised models which have been developed for various actuation systems make it possible to model them fairly accurately, this data is not always available. Hence different techniques are used to obtain the parameters, usually based on the input and output of the actuation system. Frictional effects, gear backlash and

other nonlinearities associated with the actuation and transmission drive train, complicate the task of creating an accurate model.

Once the model of a robot is created, it needs to be validated before it can be used for simulation or control purposes. The process of model validation, is not only an iterative one, but it always helps to originally divide the complete task of validation in to components, which can then be combined to get the total result. By doing this, errors can be pin pointed to a specific area, and as a result, it speeds up the elimination process.

In this chapter, both direct and inverse kinematics, will be discussed, and then manipulator dynamics are considered. Since for the experimental work, a rigid, serial open-chained rotational manipulator, namely a TecQuipment MA3000 Robot is used, the kinematics and dynamics aspects considered, will mainly concentrate on this type of manipulator. Kinematic and inertial parameters of the MA3000 robot are obtained, for the first three links, using a CAD system. A Symbolic manipulation approach is used to generate the dynamic equations of the robot. Based on the particular geometry of the manipulator, algebraic simplifications, and eliminating repetitive computations, make these equations efficient. Then permanent magnet dc motors are looked at, and a model of each joint's motor is developed. The value for the torque constants, not available in the manufacturers catalogue, was found, using steady speed characteristics of the motors and the joint angle data acquired from the robot, as a result of step voltage inputs. Although, models of friction and backlash are not included, a number of methods that can be used for compensation of these nonlinearities, are discussed and the appropriate method can be used when model-based control is implemented.

Finally behaviour comparisons of the model developed and the real robot is included in the final section (Model Validation), and a close match for the waist and the elbow of the robot is shown. Some explanations are offered, for the discrepancies in the case of the shoulder.

3.2 Robot Kinematics

3.2.1 Direct Kinematics

Direct kinematics constitutes the problem of finding the position and orientation of the end effector of a robot manipulator, given the joint variables. The following approach is taken in order to define the kinematic and inertial parameters for the MA3000 and also find their values. First of all, a fixed reference coordinate frame is assumed at the base of the manipulator and then individual body-attached coordinate frames are established. A convention is used for the principal axes, whereby the z_{i-1} axis is positioned such that it goes through the rotational axis of the i^{th} joint, the x_i axis pointing away along the length of the z_{i-1}, axis and the y_i axis is located so that, a right-handed coordinate system is formed. The relative translation and rotation between the coordinate systems attached to each link is described by a 4 $\times$ 4 matrix which combines a 3 $\times$ 3 rotation matrix (RM), a 3 $\times$ 1 position vector (PV) and a 1 $\times$ 3 perspective transformation (PT), plus an element for global scaling factor (E).

$$\begin{bmatrix} RM & | & PV \\ - & - & - \\ PT & | & E \end{bmatrix}$$

In this case $PT = 0$ and $E = 1$. This represents the relative position of one link with respect to an adjacent one. The position and orientation of any of the links including the gripper can then be described in the three-dimensional space of the manipulator's work envelope with respect to the base coordinate system, using matrix products. Using the Denavit-Hartenberg (D-H) notation, four geometric parameters, (α, d, a, θ) describe the kinematic relationship between two bodies in a serial chain mechanism connected by uniaxial joints:

- The twist angle, α_{i-1}, is defined as the angle between the projection of the z_i and z_{i-1} about x_i axis. This is constant for rotary joint robots.

- The length parameter, a_{i-1}, is defined as the mutually orthogonal distance between z_i and z_{i-1} . This is also constant in the case of MA3000.
- d_i is the distance from the origin of the $(i-1)^{th}$ coordinate frame to the intersection of the z_{i-1} axis with the x_i axis along z_{i-1} axis. Also constant.
- The joint variable θ_i about z_{i-1}, from x_{i-1} to x_i.

Now a homogeneous transformation matrix A_i, that represents a rotation of (α) angle about x_i, followed by a translation of (a) units along the same axis, followed by a further translation of (d) units along z_i and a rotation of (θ) angle about z_i, will give the D-H transformation matrix for adjacent coordinate frames i and i-1.

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha & \sin\theta_i\sin\alpha & a\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha & -\cos\theta_i\sin\alpha & a\sin\theta_i \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

To specify the position of the center of the gripper P, three degrees of freedom are required, and three more, to specify the orientation of it. The components of P in the principal axes are p_x, p_y, p_z. Figure 3.1 shows the coordinate frames, the D-H parameters and the vectors representing the gripper orientation for the MA3000 robot.

The transformation from base to the gripper will then be:

$${}^0T_n = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

Vectors $\underline{a}$, $\underline{s}$, $\underline{n}$ represent the approach vector, sliding vector in the direction of finger closure, and normal vector to form a right handed system, respectively. These vectors describe the gripper orientation. n_x, n_y, n_z are the components of the $\underline{n}$. The MA3000 arm link D-H coordinate parameters for the first three

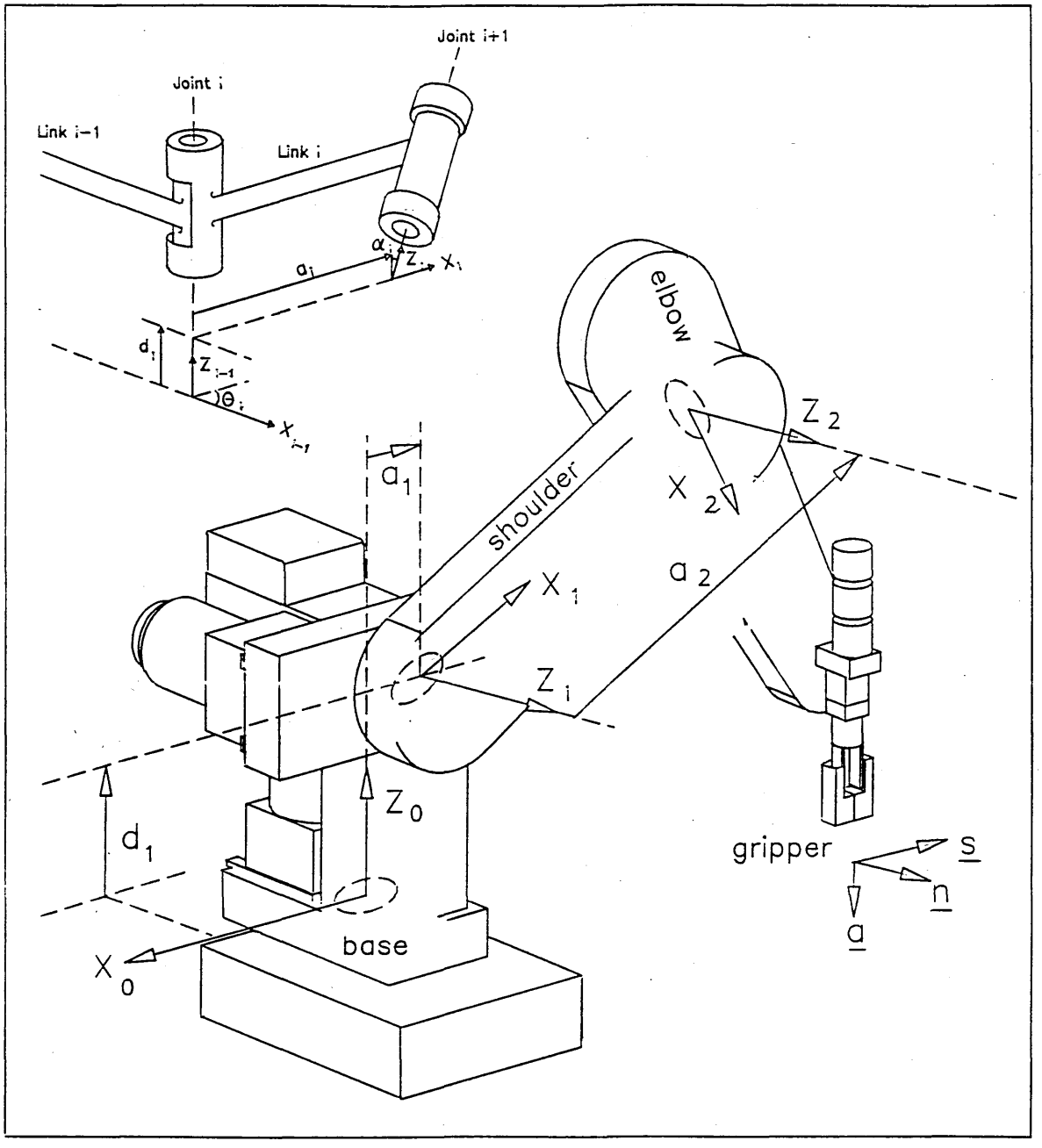


Figure 3.1: Link coordinate system and orientation vectors for an MA3000 Robot

Joint i	θ_i	α_i $(deg.)$	a_i (m)	d_i (m)	Joint range $(deg.)$
1	θ_1	-90	105×10^{-3}	304.5×10^{-3}	0 - 270
2	θ_2	0	402×10^{-3}	0	0 - 270
3	θ_3	0	352.5×10^{-3}	0	0 - 270

Table 3.1: Robot Arm link coord. parameters

links are tabulated in table 3.1. The values of a_i and d_i, along with other link parameters were obtained using a CAD approach, which will be explained in the next section. Note that values for d_2 and d_3 are both zero due to the fact that the origin of the coordinate frames for shoulder, elbow and pitch are assumed to be on the same line, to satisfy the sufficient condition of, 3 adjacent joint axes intersecting, for a closed form solution of the inverse kinematics, which will be discussed later.

Kinematic and Inertial Parameters of the MA3000

The kinematic and inertial parameters of even identical robots are very rarely the same, due to errors which occur during the manufacturing process. Even if this was the case, manufacturers are reluctant to measure the inertial parameters, as presently this is of no benefit to them, for the type of controllers that they employ. An experimental way of determining these parameters, would be to take the components apart, measure them to obtain the mass, counter balance to get the center of mass and swing the pieces to find the inertias.

Parameter estimation has been used to find both kinematic and inertial parameters, treating them as different and independent parameter sets that are to be identified. To estimate the kinematic set, coordinate frames are assigned to each link of the manipulator and using homogeneous transformations relating the coordinate frames, then linear equations are formed, in terms of the joint variables, unknown parameters (eg. length of each link) and the position and orientation of the end effector. Assuming the joint variable values and the position and orientation of the end effector are known, least squares is used to estimate the unknown parameters. For the inertial parameters, nonlinear kinematic equations of the manipulator, are linearized about the unknown parameters by ignoring the higher order differential changes and then again least squares is used to estimate the unknown parameters. Yonghong et al [106] present these methods.

Another Parameter identification approach for inertial parameters is presented in

[4]. In their approach, each link is seen as a load by the adjacent joint. Measurements of the joint torques about the joint axis are made and inertial parameters that appear linearly in the parameters in the dynamic equations are estimated. They point out that, since joint torques are not influenced by all the inertial parameters, they can not all be identified, and the ones that can be identified, can only be determined in linear coefficients. Nevertheless they conclude that the parameters that can not be estimated, are unimportant for control, because they do not effect the torques necessary to drive the robot.

The approach taken here is a CAD approach. As the advantages of using CAD/CAM become apparent to manufacturers of various robot components, it can in the very near future, be the case, whereby the geometric model of individual components become available in a standard CAD specific data format. Then these can be used for very accurate geometric modelling. In the case of MA3000, unfortunately this is not the case. Although a bit of effort is required to build the 3-D model from blue prints, there are many advantages in employing this method. One advantage is that, as model validation is an iterative process, once the model is created, alterations can easily be accommodated. Another advantage is that there is no need to employ force or other type of sensors. However one disadvantage can be the problem with accuracy, but inaccuracies also exist when sensors are used. The approximations made were mainly in the inertias of some components, for example motors. The weight and the volume of the motors were known and an appropriate density is attached to the whole motor, which means that it assumes that mass is evenly distributed. However the resulting inaccuracies are not that great, as the volume to weight ratio is quite small, compared to the overall ratio.

The CAD approach, Using CAM-X to calculate the parameters

CAM-X, the Ferranti Infographics integrated CAD/CAM system, consists primarily of InfoCAD which is an interactive, computer aided 2D Design/Draughting system for defining engineering drawings, storage, and output to other systems.

InfoCAD enables the user to interactively communicate with the CAM-X 3D Modeller program and to generate data, either directly from InfoCAD or via the 3D Modeller. The 3D geometry of mechanical components and assemblies can be defined using InfoSOLID which is a computer aided 3D design program. The stages in creating a solid model and extraction of relative parameters from it are as follows:

- The geometry of the face boundaries and profiles of the parts which constitute a body, are defined in 2D, by digitising the points that form them, or typing the coordinates.
- Then the 3D models are constructed from 2D face boundaries and profile data.
- These are then combined, using what is known as boolean operations (unite, subtract, intersect, ...) to model the main body in 3D.
- Model data is then available for manipulation and different views of the body can be generated for visualisation.
- Finally design related information such as surface areas, columns, and moments of inertia, position of center of gravity can be extracted.

When dealing with 3D models, accuracy becomes an issue. Depending on how the model is to be used and the nature of its use, emphasis is put on the trade off between speed of the processing and accuracy. For example, the calculation of the properties of a body which contains blended surfaces, is carried out by taking a user specified number of slices through the body and summing the results obtained for each slice. Now if the number of slices specified is high, then the model will be more accurate at the expense of increased computation time. In general the factors affecting the accuracy of the 3D model include:

- The accuracy to which the model represents the real world.

- The accuracy associated with the interpretation of the real world.
- The precision of the computer.

The 2D model representation was carried out, by digitising the blue print of different sections. The only significant approximation made, was for modelling the motors. However, as the weight and volume of the motors were known, the value of density was fixed, so as to give the correct result for the inertia. As far as the computer precision is concerned, although the the system uses double precision, the points can not be defined separately, closer than the distance that the model resolution dictates. However this resolution was quite satisfactory for the Robot model. The properties that can be calculated are as follows:

- The area of a specified face or faces of the body (mm^2)
- The total surface area of the body (mm^2)
- The volume and centre of gravity of the body (mm^3)
- The mass properties of a the body or sections of the main body for defined axis of inertia (The results take into account the specified density of different sections)

Views of the 3-D model of the waist, shoulder and elbow of MA3000 robot can be seen in figures 3.2, 3.3, 3.4. And the results are tabulated in tables 3.2, 3.3, 3.4. In the tables, note that the position of center of masses, are measured with respect to each link's own coordinate frame. Also only three diagonal elements of the inertia matrix

$$\begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

are non zero, because, for example in

$$I_{xy} = \int_{-\frac{l}{2}}^{\frac{l}{2}} xydm$$

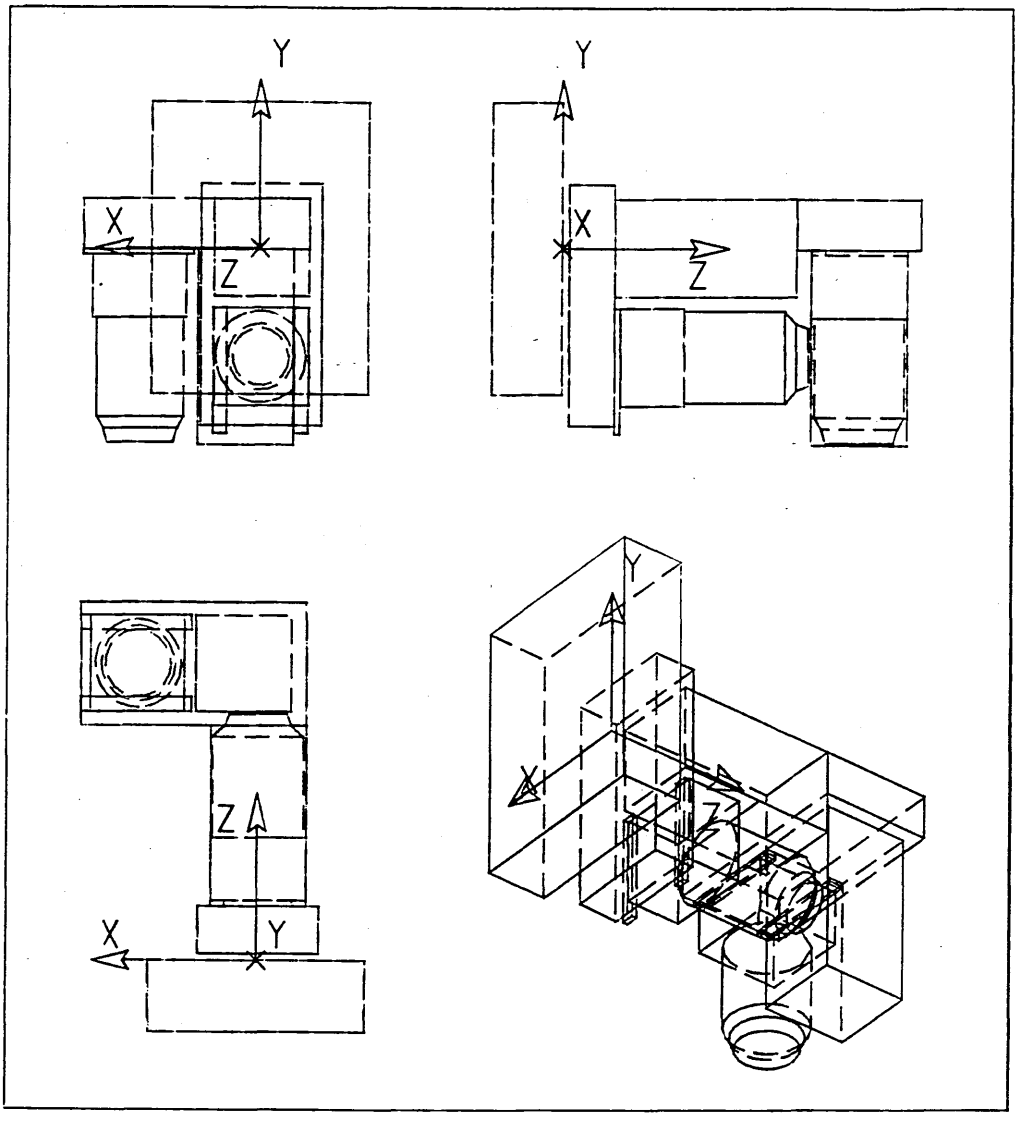


Figure 3.2: Four views from the 3-D model of waist

x comp. position of cent. of mass for waist (m)	32.2461 $\times 10^{-3}$
y comp. position of cent. of mass for waist (m)	-57.1215 $\times 10^{-3}$
z comp. position of cent. of mass for waist (m)	211.881 $\times 10^{-3}$
Inertia I_{xx} for waist $(kg.m^2)$	4.80973 $\times 10^{-1}$
Inertia I_{yy} for waist $(kg.m^2)$	5.53040 $\times 10^{-1}$
Inertia I_{zz} for waist $(kg.m^2)$	2.24603 $\times 10^{-1}$
Mass of waist (kg)	29543.8 $\times 10^{-3}$

Table 3.2: MA3000 link parameters obtained from CAM-X (waist)

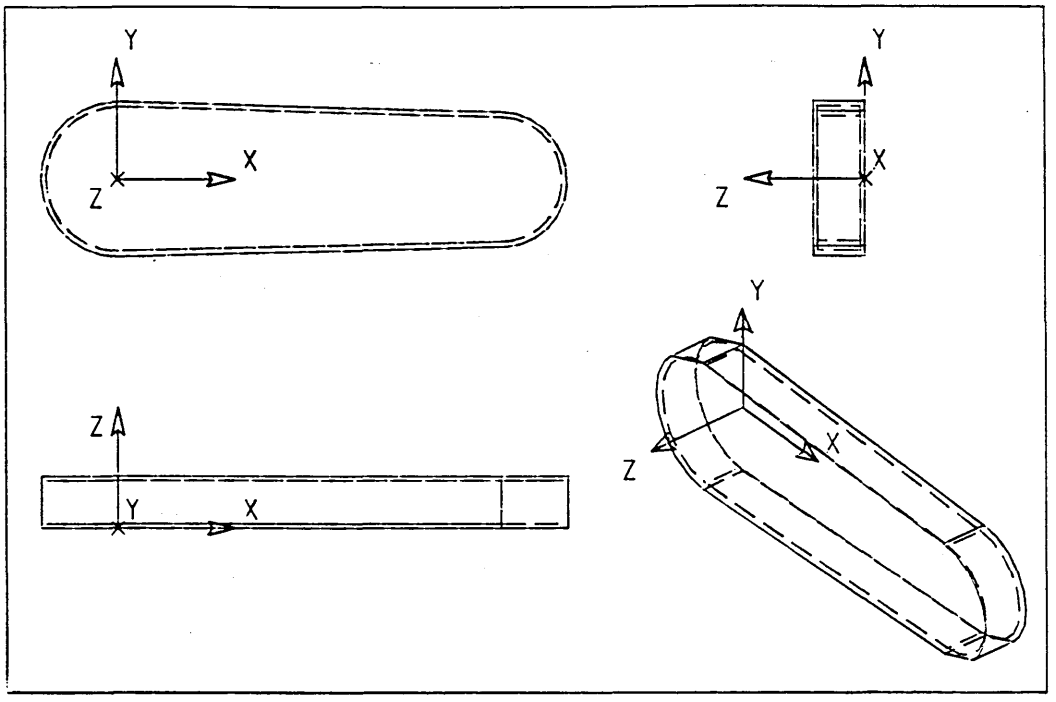


Figure 3.3: Four views from the 3-D model of shoulder

x comp. position of cent. of mass for shoulder (m)	190.587 $\times 10^{-3}$
y comp. position of cent. of mass for shoulder (m)	-1.32364 $\times 10^{-3}$
z comp. position of cent. of mass for shoulder (m)	27.5711 $\times 10^{-3}$
Inertia I_{xx} for shoulder $(kg.m^2)$	7.34089 $\times 10^{-2}$
Inertia I_{yy} for shoulder $(kg.m^2)$	7.94399 $\times 10^{-1}$
Inertia I_{zz} for shoulder $(kg.m^2)$	8.75983 $\times 10^{-1}$
Mass of shoulder (kg)	2847.45 $\times 10^{-3}$

Table 3.3: MA3000 link parameters obtained from CAM-X (shoulder)

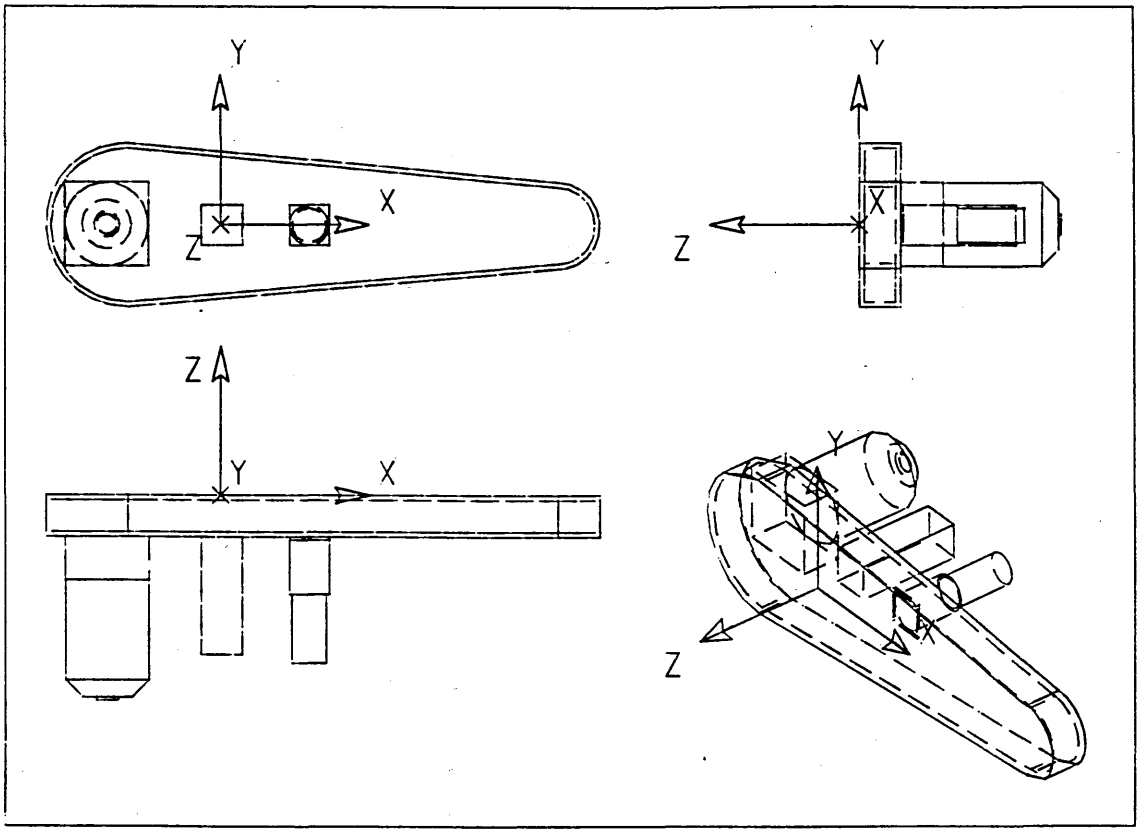


Figure 3.4: Four views from the 3-D model of elbow

x comp. position of cent. of mass for elbow (m)	18.6629 $\times 10^{-3}$
y comp. position of cent. of mass for elbow (m)	-0.439201 $\times 10^{-3}$
z comp. position of cent. of mass for elbow (m)	-86.6715 $\times 10^{-3}$
Inertia I_{xx} for elbow $(kg.m^2)$	1.35227 $\times 10^{-3}$
Inertia I_{yy} for elbow $(kg.m^2)$	1.43033 $\times 10^{-2}$
Inertia I_{zz} for elbow $(kg.m^2)$	2.03530 $\times 10^{-2}$
Mass of elbow (kg)	5384.46 $\times 10^{-3}$

Table 3.4: MA3000 link parameters obtained from CAM-X (elbow)

the limits will go from negative to positive along the length, having equal values at each extreme and hence cancel out one another. This cancellation does not occur in the case of I_{xx}, I_{yy} and I_{zz}. In other words, the symmetry of the links means that the inertial principal axes coincide with the links coordinate system transfered to the centre of mass.

3.2.2 Inverse Kinematics

This deals with the problem of calculating the joint angles, given the position and orientation of the end effector. There are various approaches to the inverse kinematic problem, with individual merits and shortcomings. The main two are, the inverse transform technique of Paul [75], and a geometric approach, which is considered more appropriate for selecting a solution from several alternatives, for a particular arm configuration. In this section, an inverse transform technique is used, based on the use of the 4 × 4 homogeneous transformation matrices for the MA3000 Robot. The D-H transformation matrix for adjacent coordinate frames was discussed in the previous section. Using equation 3.1, in the case of the MA3000, $\cos\alpha_1 = -90$, $\cos\alpha_2 = 0$, and $\cos\alpha_3 = 0$, therefore:

$$A_1 = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & a_1\cos\theta_1 \\ \sin\theta_1 & 0 & \cos\theta_1 & a_1\sin\theta_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & a_2\sin\theta_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & a_3\sin\theta_3 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, a three link transformation matrix from the base to the end of the elbow is:

$$^0T_3 = A_1A_2A_3 \tag{3.3}$$

Now, the transformation ${}^1T_3$ can be obtained in two ways:

$$ {}^1T_3 = {A_1}^{-1} \times {}^0T_3 $$

and

$$ {}^1T_3 = A_2 \times A_3 $$

but

$$ {A_1}^{-1} \times {}^0T_3 = \begin{bmatrix} \frac{\cos\theta_1}{\cos^2\theta_1+\sin^2\theta_1} & \frac{\sin\theta_1}{\cos^2\theta_1+\sin^2\theta_1} & 0 & -a_1 \\ 0 & 0 & -1 & d_1 \\ \frac{-\sin\theta_1}{\cos^2\theta_1+\sin^2\theta_1} & \frac{\cos\theta_1}{\cos^2\theta_1+\sin^2\theta_1} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

which means:

$$ {}^1T_3 = \begin{bmatrix} \cos\theta_1 n_x + \sin\theta_1 n_y & \cdots & \cos\theta_1 p_x + \sin\theta_1 p_y - a_1 \\ n_z & \cdots & -p_z + d_1 \\ \cdot & \cdots & -\sin\theta_1 p_x + \cos\theta_1 p_y \\ \cdot & \cdots & 1 \end{bmatrix} \tag{3.4} $$

The second way :

$$ A_2 \times A_3 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & a_2\sin\theta_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & a_3\sin\theta_3 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

will give :

$$ {}^1T_3 = \begin{bmatrix} \cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3 & \cdots & a_2\cos\theta_2 + a_3\cos\theta_2\cos\theta_3 - a_3\sin\theta_2\sin\theta_3 \\ \cos\theta_2\sin\theta_3 + \sin\theta_2\cos\theta_3 & \cdots & a_3\cos\theta_2\sin\theta_3 + a_2\sin\theta_2 + a_3\sin\theta_2\cos\theta_3 \\ \cdot & \cdots & d_3 + d_2 \\ \cdot & \cdots & 1 \end{bmatrix} \tag{3.5} $$

Now equating member (3,4) of equations 3.4 and 3.5 will give:

$$ -\sin\theta_1 p_x + \cos\theta_1 p_y = d_3 + d_2 \tag{3.6} $$

Let $p_x = r\cos\phi$ and $p_y = r\sin\phi$, for ϕ on the xy plane passing through the base frame.

$$\phi = \tan^{-1}\left(\frac{p_y}{p_x}\right)$$

$$r = +\sqrt{{p_x}^2 + {p_y}^2}$$

substituting for p_x and p_y in equation 3.6 , we get:

$$\sin\phi\cos\theta_1 - \cos\phi\sin\theta_1 = \frac{d_2 + d_3}{r} \tag{3.7}$$

with $0 < \frac{d_2+d_3}{r} \leq 1$, equation 3.7 reduces to

$$\sin(\phi - \theta_1) = \frac{d_2 + d_3}{r}$$

with $0 < (\phi - \theta_1) < \pi$. The Cosine can also be obtained:

$$\cos(\phi - \theta_1) = \pm\sqrt{1 - \left(\frac{d_2 + d_3}{r}\right)^2}$$

then

$$\tan(\phi - \theta_1) = \frac{\frac{d_2+d_3}{r}}{\pm\sqrt{1 - \left(\frac{d_2+d_3}{r}\right)^2}} = \frac{d_2 + d_3}{\pm\sqrt{r^2 - (d_2 + d_3)^2}}$$

hence

$$(\phi - \theta_1) = \tan^{-1}\frac{d_2 + d_3}{\pm\sqrt{r^2 - (d_2 + d_3)^2}}$$

and as a result

$$\theta_1 = \tan^{-1}\left(\frac{p_y}{p_x}\right) - \tan^{-}1\frac{d_2 + d_3}{\pm\sqrt{r^2 - (d_2 + d_3)^2}} \tag{3.8}$$

therefore, since all the parameters in this equation is known, θ_1 can be calculated. Now equating members (1,1), (2,1), (1,4) and (2,4) from equations 3.4 and 3.5

$$\cos\theta_1 n_x + \sin\theta_1 n_y = \cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3 \tag{3.9}$$

$$-n_z = \cos\theta_2\sin\theta_3 + \sin\theta_2\cos\theta_3 \tag{3.10}$$

$$\cos\theta_1 p_x + \sin\theta_1 p_y - a_1 = a_2\cos\theta_2 + a_3\cos\theta_2\cos\theta_3 - a_3\sin\theta_2\sin\theta_3 \tag{3.11}$$

$$-p_z + d_1 = a_3\cos\theta_2\sin\theta_3 + a_2\sin\theta_2 + a_3\sin\theta_2\cos\theta_3 \tag{3.12}$$

now equation 3.9 gives

$$\cos(\theta_2 + \theta_3) = \cos\theta_1 n_x - \sin\theta_1 n_y$$

as values of θ_1, n_x and n_y are available at this stage, the right hand of the equation is known. Let us call it R.

$$\cos(\theta_2 + \theta_3) = R \tag{3.13}$$

Similarly from equation 3.10

$$\sin(\theta_2 + \theta_3) = T \tag{3.14}$$

where T is a known value. From equation 3.11, we have

$$a_2\cos\theta_2 + a_3\cos(\theta_2 + \theta_3) = \cos\theta_1 p_x + \sin\theta_1 p_y - a_1$$

or

$$\cos\theta_2 = \frac{\cos\theta_1 p_x + \sin\theta_1 p_y - a_1 - a_3\cos(\theta_2 + \theta_3)}{a_2}$$

again at this stage all the parameters of the right hand side are known, so call the value M. Therefore

$$\cos\theta_2 = M$$

Similarly from equation 3.12

$$\sin\theta_2 = N$$

where N is a known value. And hence dividing and then inverting the function:

$$\theta_2 = \tan^{-1}\left(\frac{N}{M}\right) \tag{3.15}$$

Dividing equation 3.14 by 3.13, and taking the inverse of the tan, we get

$$(\theta_2 + \theta_3) = \tan^{-1}\left(\frac{T}{R}\right)$$

Finally

$$\theta_3 = \tan^{-1}\left(\frac{T}{R}\right) - \tan^{-1}\left(\frac{N}{M}\right)$$

So all the three angles are found.

It should be noted that the problem of being faced with multiple options of joint angles for a particular end effector position, which results in different configurations is not discussed. A routine can be written that makes the choice. In the literature, a number of works have been reported to carry this out.

The problem of singularity has also been extensively discussed elsewhere in the literature.

3.3 Robot Manipulator Dynamics

Dealing with robot dynamics, two basic problems can be formulated, one of finding the instantaneous joint accelerations, given the joint torques and forces, and second, determining the required joint torques or forces, given the joint variables and their derivatives. The former is referred to as forward or direct dynamics, and is useful for simulation purposes, where the latter is mainly of use in control, when estimates of the joint forces are required for a particular trajectory, which might be prespecified, this is known as inverse dynamics.

There are various approaches available for formulating the robot arm dynamics. Basically the dynamic model for a robot arm can be obtained from known physical laws such as the laws of Newtonian Mechanics, the Lagrangian formulation, and from physical measurements.

Lagrangian equation can be used to represent the dynamic behaviour of a system of rigid bodies. In the Lagrangian approach, the manipulator's Lagrangian is expressed in terms of generalized coordinates and their derivatives, which is then substituted into the Euler-Lagrange equation. This equation is expanded by symbolic differentiation, to give the generalized joint forces in terms of the generalized coordinates, velocities and accelerations. The Lagrangian is defined as :

L = (Kinetic energy of the system) - (potential energy of the system)

and the Lagrangian equation has the form :

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i, \qquad i = 1, 2, \ldots, n$$

where q_i is the generalized coordinated and is equal to angular displacement θ_i for rotary joints. And τ_i is the generalized forcing function.

By application of the Lagrangian equation to a rotary robot with n joints, the equations of motion can be written in the following matrix form :

$$M[\theta(t)]\ddot{\theta}(t) + Q[\theta(t), \dot{\theta}(t)] + G[\theta(t)] = \tau \tag{3.16}$$

where $M(\theta)$ is the N $\times$ N inertial coefficient matrix, which is symmetric and positive definite. Its elements are computed by :

$$M_{ik} = \sum_{j=max(i,k)}^{N} Tr\left\{\frac{\partial T_0{}^j}{\partial \theta_k}\left(\frac{\partial T_0{}^j}{\partial \theta_i}\right)^T\right\} \qquad \text{for } i, k = 1, \ldots, \text{N}$$

$T_0{}^j$ is the D-H transformation matrix from the reference coordinate frame to the j^{th} coordinate frame.

$Q(\theta, \dot{\theta})$ is the N $\times$ 1 Coriolis and centrifugal force vector, elements of which are computed by :

$$Q_i = \sum_{k=1}^{N}\sum_{m=1}^{N} Q_{ikm}\dot{\theta}_k\dot{\theta}_m \qquad \text{for } i = 1, \ldots, \text{N}$$

elements Q_{ikm} are obtained by :

$$Q_{ikm} = \sum_{j=max(i,k,m)}^{N} Tr\left\{\frac{\partial^2 T_0{}^j}{\partial \theta_k \partial \theta_m}\left(\frac{\partial T_0{}^j}{\partial \theta_i}\right)^T\right\} \qquad \text{for } i, k, m = 1, \ldots, \text{N}$$

$G(\theta)$ is the N $\times$ 1 gravitational force vector. Its elements are computed by :

$$G_i = \sum_{j=i}^{N}\left(-m_j g\left\{\frac{\partial T_0{}^j}{\partial \theta_i}\right\}\bar{r}_j\right) \qquad \text{for } i = 1, \ldots, \text{N}$$

where g is the gravity vector expressed in the base coordinate frame, m_j is the mass of j^{th} link, and $\bar{r}_j$ is the mass center vector of the j^{th} link.

Although these equations are structured, they are unfortunately, computationally inefficient. To increase the efficiency, Hollerbach [32] developed a Recursive

method of Lagrangian formulation, which reduced the computational time taken. An approach which has the advantage of speed and accuracy is based on the Newton - Euler vector formulation. The resulting dynamic equations, excluding the dynamics of the control device, gear friction, and backlash, are a set of forward and backward recursive equations. These equations are documented in many text books, eg. [75] and [23].

For the case when the links are rotational these equations are :

Forward equations:

$$\theta_i = \theta_{i-1} + z_{i-1}\dot{q}_i \tag{3.17}$$

$$\dot{\theta}_i = \dot{\theta}_{i-1} + z_{i-1}\ddot{q}_i + \theta_{i-1} \times (z_{i-1}\dot{q}_i) \tag{3.18}$$

$$\dot{v}_i = \dot{\theta}_i \times p_i^* + \theta_i \times (\theta_i \times p_i^*) + \dot{v}_{i-1} \tag{3.19}$$

$$\bar{a}_i = \dot{\theta}_i \times \bar{s}_i + \theta_i \times (\theta_i \times \bar{s}_i) + \dot{v}_{i-1} \tag{3.20}$$

Backward equations:

$$F_i = m_i \bar{s}_i \tag{3.21}$$

$$N_i = I_i\dot{\theta}_i + \theta_i \times (I_i\theta_i). \tag{3.22}$$

$$f_i = F_i + f_{i+1}. \tag{3.23}$$

$$n_i = n_{i+1} + p_i^* \times f_{i+1} + (p_i^* + \bar{s}_i) \times F_i + N_i. \tag{3.24}$$

$$\tau = n_i^T z_{i-1} + b_i\dot{q}_i. \tag{3.25}$$

Where

θ = angle of rotation

$\dot{q}_i$= the magnitude of angular velocity of link i w.r.t. the coordinate system $(x_{i-1},y_{i-1},z_{i-1})$

m_i = total mass of link i

$\bar{r}_i$ = position of the center of mass of link i from the origin of the base reference frame

$\bar{s}_i$ = position of the center of mass of link i from the origin of the coordinate system (x_i, y_i,z_i)

p_i^* = the origin of the ith coordinate frame with respect to the (i -1)th coordinate system

$\bar{v}_i = d\bar{r}_i/dt$, linear velocity of the center of mass of link i

$\bar{a}_i = d\bar{v}_i/dt$, linear acceleration of the center of mass of link i

F_i = total external force exerted on link i at the center of mass

N_i = total external moment exerted on link i at the center of mass

I_i = inertia matrix of link i about its center of mass with ref. to the coordinate system (x_0 ,y_0, z_0)

f_i = force exerted on link i by link i-1 at the coordinate frame (x_{i-1}, y_{i-1} ,z_{i-1}) to support link i and the links above it

n_i = moment exerted on link i by link i-1 at the coordinate frame (x_{i-1}, y_{i-1}, z_{i-1})

b_i = the viscous damping coefficient for joint i

Also

$\theta_0 = \dot{\theta}_0 = v_0 = 0$ and $\dot{v}_0 = (g_x, g_y, g_z)^T$

If we choose a 3×3 rotation matrix which transforms any vector with reference to coordinate frame (x_i, y_i, z_i) to the coordinate system (x_{i-1}, y_{i-1}, z_{i-1}), it is possible to write the recursive equations of motion of a link about its own coordinate frame.

One of the disadvantages associated with N - E equations is the lack of structure which is needed for deriving advanced control laws. [53] on the basis of the Generalized d'Alembert principle, was able to utilise the vector and rotation matrix representation to describe each link's kinematic information, obtain the kinetic and potential energies of the robot arm to form the Lagrangian function, and apply the Ε-L formulation to obtain the equations of motion. This formulation retains the "structure" of the problem with a moderate computing penalty. Another well known method is Kane's method, where for a system with n degrees of freedom, Kanes dynamic equations can be written as:

$$F_r + F_r^* = 0, \qquad r = 1, \ldots, n$$

where F_r is the generalized force and F_r^*, the generalized inertia forces.

To enhance the efficiency some use notations which cut down on computation, for example Featherstone [21] used "Spatial Notation", to represent velocity, acceleration, etc. by a pair of vectors (one linear and one angular) to describe a number of methods for calculating robot dynamics efficiently, and for algebraic and notational convenience. Special vectors are similar to quantities called screws and motors.

In general, it is well established that a closed form of dynamic equations is appealing for both dynamic modelling and control applications. However deriving the closed form of these equations, is a tedious task and a very error prone process. Developing a computer program that generates the robot dynamics in a symbolic form, can alleviate the effort of deriving these equations. In fact several researchers have taken this approach, using various methods. REDUCE, which is a symbolic manipulation system, C & LISP, FORTRAN and MACSYMA are the systems and languages which have been used.

In the next sub section, different approaches for obtaining efficient robot dynamic equations will be discussed.

3.3.1 Customized Robot Dynamics

To enhance computational efficiency for dynamic simulation and real-time control, it is possible to generate a simplified symbolic model on the basis of both the manipulator structure and algebraic simplification, which remove repetitive calculations. The above two principles were exploited, as will be explained later, in obtaining the dynamic equations for the MA3000 robot.

Observations made by Hollerbach [32] about the computational efficiency of manipulator dynamics namely:

- Particular kinematic and dynamic structures of manipulators can be utilised, for improving the computational efficiency of dynamic algorithms. (eg. regularity of manipulator configuration)

- Fundamental physical principles of formulations, can be used to achieve efficiency. (simplification based on structure of the equations)

form the basis of simplifications achievable for manipulator dynamic equations. In addition to these, general simplification procedures that apply to all algebraic systems can be used. A number of research works have been dedicated to developing methods that contribute to the simplification of dynamic equations, both for simulation purposes and control.

Efficient methods for simulating a robotic mechanism is considered in [98], and efficient solution of the dynamic equations for a general N degree of freedom, single open chain robotic mechanism is discussed. As in the general form of equations of motion for a manipulator, the joint torques are linear functions of joint accelerations, the equations of motion are written in the form

$$H(q)\ddot{q} = (\tau - b) \tag{3.26}$$

where q is the vector of joint variables, H is the inertia matrix, τ is the vector of torques of each joint actuator, and b represents the torques (forces) due to gravity, centrifugal and coriolis accelerations, and external forces and moments on link N. Accelerations can then be found by solving this linear equation. By setting q, $\dot{q}$ and the vector of external moments (k) to their current state, but letting $\ddot{q} = 0$, b can be computed.

Four techniques are then described to find elements of the inertia matrix. In the first method, elements of the matrix are evaluated by setting q to its current state, and computing the matrix one column at the time, when joint velocities are zero, there are no gravitational effects and the joint accelerations are all zero apart from one corresponding to the column being computed which is 1, and then solve the

linear equations for acceleration.

The second method they suggested is the same as above, but symmetry of matrix H is used to only calculate the diagonal and the bottom half of the off-diagonal terms. The third method is again the same as the first, but a different procedure is used for computing the inertia matrix.

In the fourth method, an iterative process is used to solve for the joint accelerations. In this method, an initial estimate for the joint accelerations is made and this is followed by successive adjustments until they converge to the correct solution.

They found that the third method was the most efficient one, due to the fact that it takes into account the symmetrical form of moment of the inertia matrix, and also utilises a recursive procedure for computing the mass, the center or mass and the moment of inertia matrix of the composite system of links.

An efficient algorithm for generating the dynamic equations of open chain manipulators is presented in [104]. Their method is based on a modified Lagrange-Christoffel formulation and they include generalized pseudo-inertia matrices of manipulators. The use of Christoffel expression of the Coriolis and centrifugal coefficients:

$$Q_{ijk} = \frac{1}{2}\left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i}\right)$$

leads to substantial simplification for symbolic derviation of these coefficients. This is due to the fact that the obtained inertial coefficients are used.

Calculation of generalized forces is suggested to be divided into two parts by [37], dynamic coefficients in the background and generalized forces in real-time. They created a LISP program capable of symbolic manipulation, to generate the dynamic coefficients automatically, with D-H parameters, masses, center of gravity, and moments of inertia of the links as input, and a C program that calculates the coefficients as output. One of the limitations of their work is that the capability is limited to only the inertial and gravitational coefficients and does not include the centrifugal and coriolis coefficients. This is only suitable for low speed robot

manipulators.

An efficient form of symbolic generation of manipulator equations is presented in [12], where the amount of real time computation required to compute the complete set of configuration dependant dynamic parameters is reduced. They base their simplification upon two stages: firstly factorization of the equations of motion using a set of rules to guide the factorization and simplification, and secondly, segregation of the computations into configuration dependant and configuration independent portions, where possible. The configuration independent portion is computed once and stored as constants.

A detailed description of systematic organisation of, symbolic dynamic robot models, which are generated by the computer program Algebraic Robot Modeler (ARM) is given in [69]. The performance of the systematic organisation procedure is compared with other organisations documented, and its superiority is shown. The systematic organisation procedure, is applicable to both closed-form and recursive dynamic robot models. It consists of removing unnecessary calculations, identifying constant expressions, ordering the calculations, eliminating redundant transcendental function calls and removing repetitive calculations within and across the equations.

The important aspects in the selection of a method for simplification of dynamic equations can be : the complexity of the method, complexity of the equations, cost of computation, possibility of interpretation and possibility of reduction, possibility of integration of the code into an existing code and so on. Recursive numerical methods are known to be faster, with the penalty that structural information about the model can not be obtained.

Since the equations in the case of the MA3000, need only be generated once (as its configuration is not likely to change) and the relatively long computational time for generation of a symbolic code is not really problematic, it was decided to use a software package (REDUCE) for symbolic manipulation of equations. One further point to note is that, the simulations were to be carried out using

MATLAB [1] and it is possible to write a reasonably short code to create outputs in a MATLAB readable format. Also the control programs that make use of the dynamic equations, are written in OCCAM, and creation of OCCAM readable files from REDUCE is not very difficult. The equations used, were the recursive N-E about each links coordinate frame. The steps in the REDUCE code, to carry out the symbolic simplifications and arrangement of the results in a closed-form are :

1. Get the number of degrees of freedom (N)
2. Define the kinematic parameters including D-H parameters, and inertial parameters
3. Define all the vectors and matrices including the homogenous transformation matrix
4. Define the initial value processes
5. Generate the forward dynamic equations for 1 to N
6. Generate the backward dynamic equations for N to 1
7. Define all trigonometric simplifications
8. Output the Inertia matrix M and the vector of coriolis and cetrifugal torques Q, in a required format

The the joint torques from actuators are calculated from:

$$\tau = M\ddot{\theta} + Q \tag{3.27}$$

where $\ddot{\theta}$ is the vector of joint accelerations.

It should be noted that M is symmetric, positive definite and bounded above and

[1] i.e. Matrix Laboratory, is an interactive program for scientific calculations.

below. It is also nonsingular and its inverse is positive definite and bounded. The Kinetic energy of the manipulator has the form

$$\frac{1}{2}\dot{\Theta}^T M \dot{\Theta}$$

where $\dot{\Theta}$ is the vector of angular velocities. and its derivative is equal to the power input by the actuators and gravitational torque. The correctness of these equations were tested against equations generated by the Lagrange-Euler method, using REDUCE. The resulting equations are of the form:

$$\tau = M(\theta)\ddot{\Theta} + Q(\dot{\theta}, \theta) + G(\theta) \tag{3.28}$$

where $G(\theta)$ is the vector of gravity torques. Both approaches, resulted in the same set of equations.

3.4 Actuation Model

In general the actuation systems for industrial robots, are either hydraulic, pneumatic, or electrical. The MA3000 robot, has permanent magnet dc motors for the five main degrees of freedom, and a pneumatic drive for the gripper. Since we only concentrate on the first three links, namely waist, shoulder and elbow, only the relevant motors will be considered.

The motors are equipped with gear systems, to achieve the high torques required, from relatively high rotational velocities and low torque produced. Pulleys and belts are then used to deliver even higher torques to the joint bearings. Figure 3.5 shows a representation of a dc motor with gears and pulley- belt arrangement, together with the load.

The Subscripts a represent the associated parameters of the actuator (motor), m, the manipulator (fixtures and so on) at the motor side, and l, the load parameters. Subscript g is for gravity. The inertias are represented by J, B represents the damping coefficient, τ, torque and θ, angular displacement. θ_s is the angular

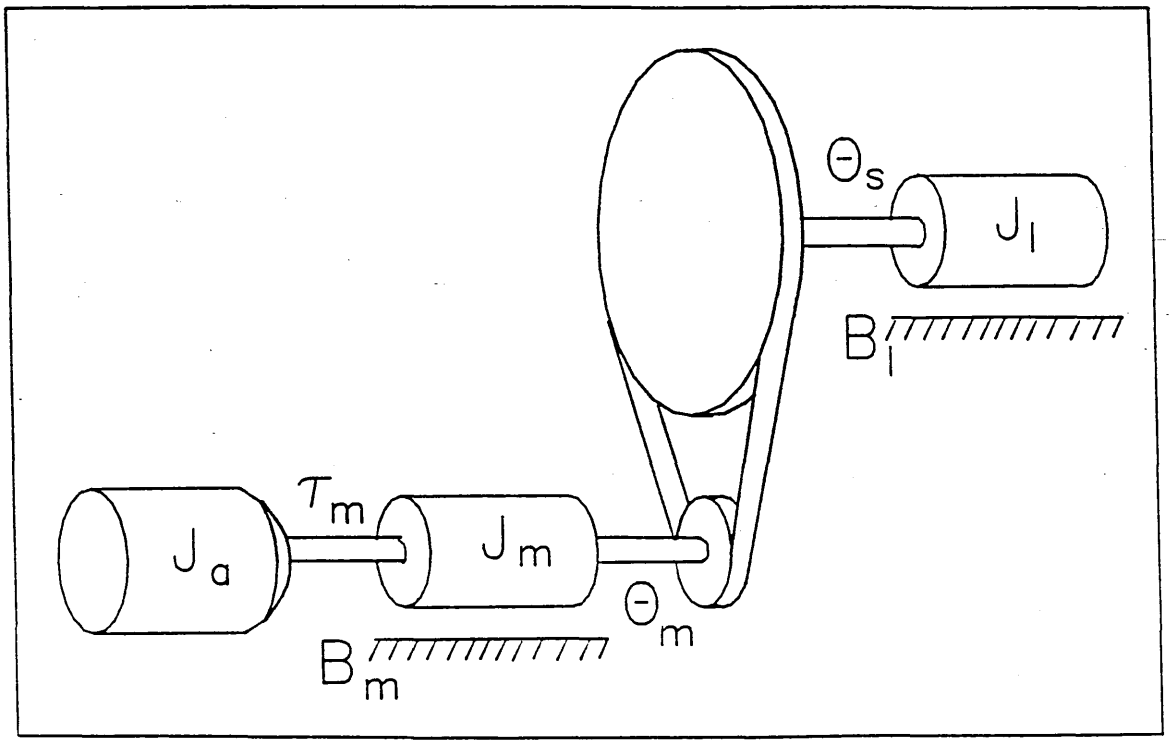


Figure 3.5: Schematic of a dc motor with gears -pulley & belt - load assembly

displacement at the load side.

Now the gear ratio n which in our case will also include the pulley-belt system ratio, is defined as

$$n = \frac{\theta_s}{\theta_m}$$

or

$$\theta_s = n\theta_m \tag{3.29}$$

The inertial load torque can be found using D'Alembert's principle to obtain:

$$T_l - B_l\dot{\theta}_s = J_l\ddot{\theta}_s \tag{3.30}$$

Similarly at the motor shaft, we get

$$T_m - nT_l - B_m\dot{\theta}_m = (J_a + J_m)\ddot{\theta}_m \tag{3.31}$$

From equations 3.29, 3.30, and 3.31, we obtain

$$T_m - n(B_l\dot{\theta}_s) - B_m\dot{\theta}_m = (J_a + J_m)\ddot{\theta}_m$$
$$+ J_l\ddot{\theta}_s$$

or

$$T_m = \underbrace{(J_a + J_m + n^2 J_l)}_{J_{eff}} \ddot{\theta}_m + \underbrace{(B_m + n^2 B_l)}_{B_{eff}} \dot{\theta}_m \tag{3.32}$$

J_{eff} and B_{eff} are the effective inertia and effective damping coefficient at the motor shaft. The Laplace equivalence of 3.32 is

$$T_m(s) = (J_{eff} s^2 + B_{eff} s)\Theta_m(s) \tag{3.33}$$

An electro-mechanical model of a permanent magnet dc motor can be seen in figure 3.6.

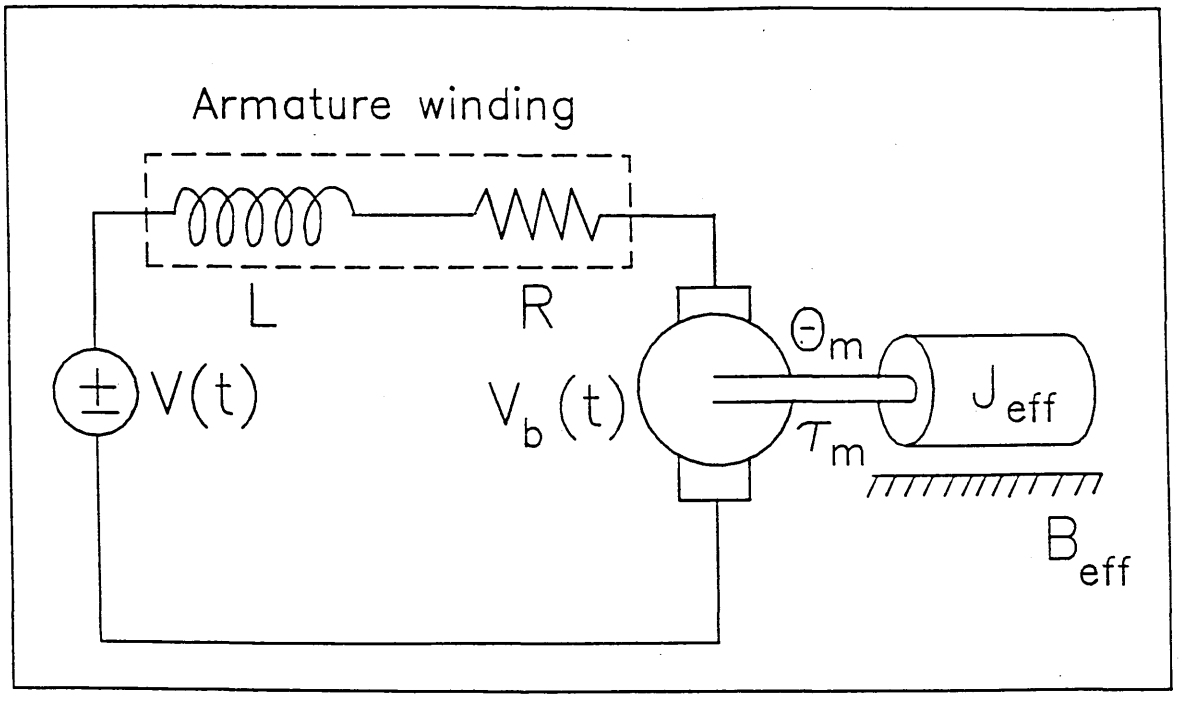


Figure 3.6: Model of an Armature driven dc motor

Resistance and Inductance are represented by R and L respectively. $V(t)$ is the drive voltage and $V_b(t)$, the back emf. $V_b(t)$ is proportional to the angular velocity $\dot{\theta}_m$, which means:

$$V_b(t) = K_b \dot{\theta}_m(t) \tag{3.34}$$

where K_b is a constant. The following frequency domain relation can be obtained by applying Kirchhoff's voltage law to the circuit and noting that the inductance is negligible.

$$V(s) - K_b s\Theta_m(s) = (Ls + R)I(s) \cong RI(s) \tag{3.35}$$

The torque generated will be proportional to the armature current, when the motor is operated in its linear range and hence:

$$T_m(s) = K_I I(s) \tag{3.36}$$

K_I is usually referred to as the torque constant of the motor. Combining equations 3.35 and 3.36, we get

$$T_m(s) = K_I \frac{V(s) - K_b s\Theta_m(s)}{R} \tag{3.37}$$

Replacing T_m in 3.37 with the right hand side of equation 3.33, we get:

$$(J_{eff}s^2 + B_{eff}s)\Theta_m(s) = K_I \frac{V(s) - K_b s\Theta_m(s)}{R}$$

Rearranging, the following transfer function from the applied voltage to the angular displacement of the motor shaft is obtained:

$$\frac{\Theta_m(s)}{V(s)} = \frac{K_I}{s[RJ_{eff}s + (RB_{eff} + K_I K_b)]} \tag{3.38}$$

The block diagram of this transfer function can be seen in Figure 3.7.

The effect of other joints are: inertia couplings, centrifugal and coriolis terms which also contribute to the overall forces (torques). These are shown in the figure, as well as the quantities that represent the reaction from the physical burden to the robot, i.e. the external load torque T_l, and gravitational torque T_g.
Between the angular dispacements at load side and motor shaft, there is a dead zone representation for free play (backlash) between the two gears, where no output is produced while the input changes. Compensation for backlash is not included here, and the above is only included to highlight the nonlinearities that exist. There are various approaches for dealing with backlash. The usual way is

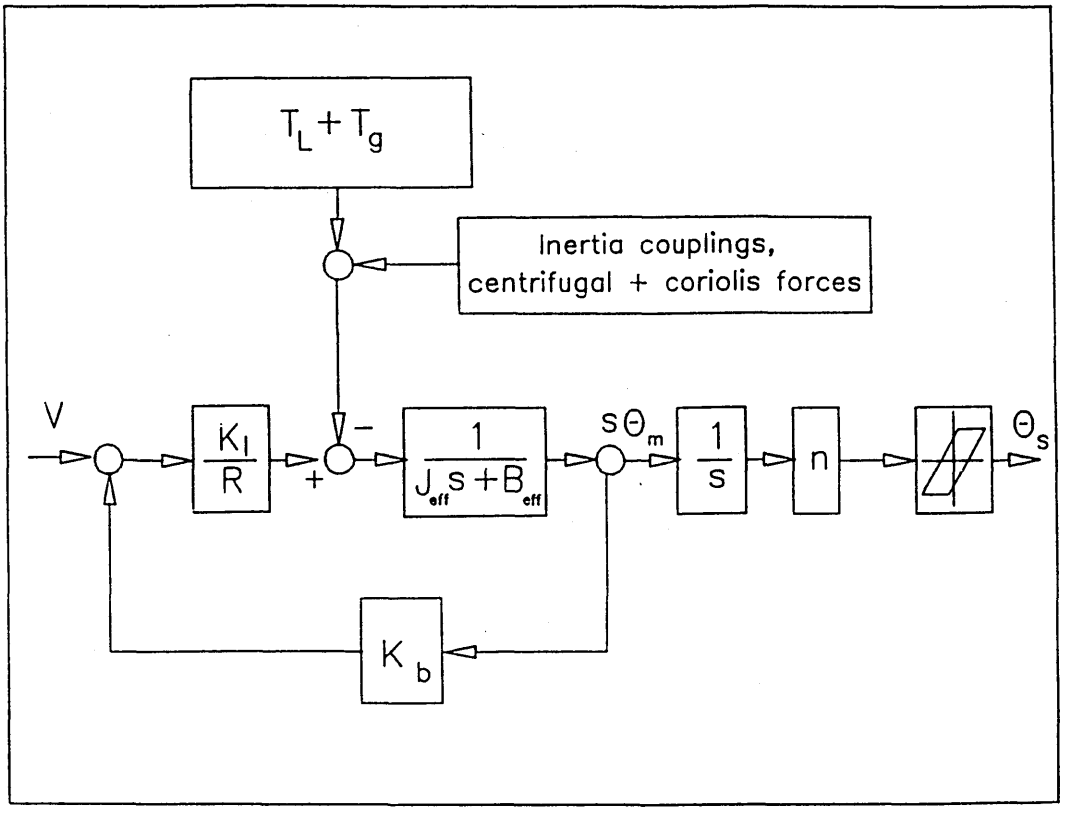


Figure 3.7: Block diagram for one joint of the robot arm

to consider them as unknown high frequency disturbances, when their frequency is higher than the control bandwidth. One approach for dealing with low frequency backlash is suggested by de Silva [17] whereby gear stages are included in the dynamic formulation of the manipulator and using Newton-Euler recursive formulation, drive torques are calculated for each gear stage. Then if drive torque changes sign at a gear stage, it means backlash exists and that stage is disengaged and zero transmitted torque is assumed. By applying the N-E recursion to the final section which is disengaged up to the gripper, the drive torques for the specified gripper trajectory should then be computed. Finally the motion of the remaining sections are computed, using drive torques which were calculated at the start, then disengagement of the sections can be checked.

Calculation of K_I and K_b

At a steady speed, the electrical power of the motor converted to mechanical form, i.e. $V_b I$ or $K_b \dot{\theta} I$, should equal the power at the shaft in mechanical form i.e. $T_m \dot{\theta}$

or $K_I I\dot{\theta}$, which means $K_b = K_I$. An approximation can be made at maximum speed, where speed has reached a constant value and there is no acceleration, by ignoring the frictional effects. This means that, as torque $T_m = J_{eff}\ddot{\theta}$ and $\ddot{\theta}$ is zero, then there will be no torque. And as $T_m = K_I I$, there will be no current. However the drive voltage

$$V = IR + K_b\dot{\theta}$$

and having zero current means :

$$V = K_b\dot{\theta}$$

or

$$K_b = \frac{V}{\dot{\theta}}$$

Hence from a graph of angular velocity against time, with the knowledge of the input voltage, the back emf constant K_b can be calculated.

This can also be seen from equation 3.7, when the friction term B_{eff} is neglected, and $K_I = K_b$, the equation reduces to:

$$\frac{\dot{\Theta}_m(s)}{V(s)} = \frac{K_b}{RJ_{eff}s + K_b^2}$$

which at steady state will reduce to:

$$\frac{\dot{\Theta}_m}{V} = \frac{1}{K_b}$$

To find the value of K_b for the first three joints of MA3000, namely, waist, shoulder and elbow, a step voltage input was applied to each joint and the angular displacements were recorded, from which the angular velocities were obtained. Plots of these can be seen in Figures 3.8, 3.9 and 3.10. As can be seen from the plots, the steady state value for $\dot{\theta}$ is (10.5) for waist, (9.25) for shoulder and (35.5) for elbow. Hence:

For waist

$$K_b = \frac{64}{10.5} = 6.1$$

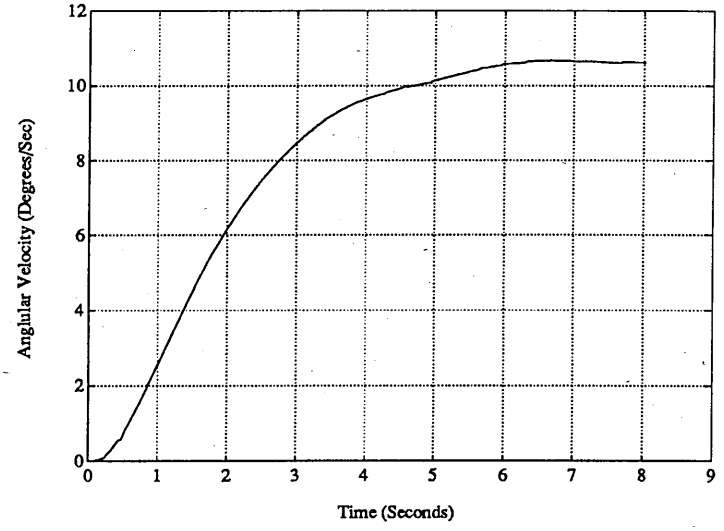


Figure 3.8: WAIST joint data for finding motor constant (64V step input)

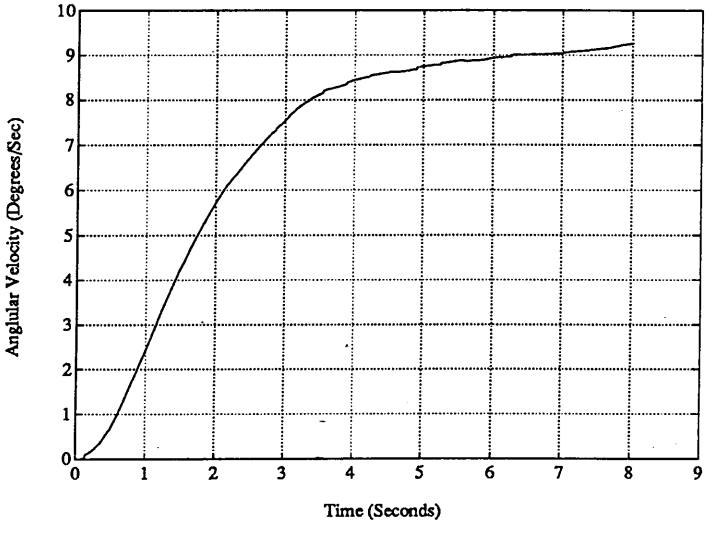


Figure 3.9: SHOULDER joint data for finding motor constant (64V step input)

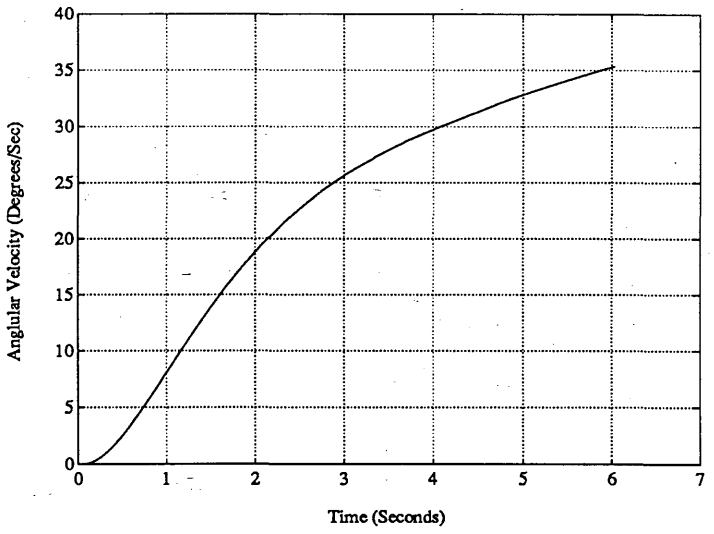


Figure 3.10: ELBOW joint data for finding motor constant (64V step input)

For shoulder

$$K_b = \frac{64}{9.25} = 6.92$$

For elbow

$$K_b = \frac{64}{38.0} = 1.684$$

Strictly speaking, the value of K_I, the motor torque constant, for many motors depends on the relative oscillation of the rotor and stator at high frequencies. This is referred to as ripple torque.

3.4.1 Friction

Frictional effects can at times be considerable and neglecting them may not be realistic. There are two basic types of friction: static, which is also referred to as stiction, is the required force for initiating motion (rolling or sliding) between two contacting surfaces, and dynamic which applies to the bodies in motion, can be in the form of coulomb friction which is caused by irregularities of contacting bodies engaging or viscous friction due to viscosity of a lubricant, or both. Coulomb friction depends on the force with which the two surfaces are pressed together and Viscous friction is considered to be a force proportional to velocity,

or higher powers of velocity at high speeds. But a suitable description is that, viscous friction is a nonlinear function of angular velocity.

There is a great deal of disagreement as to the structure of the friction model. The characteristic functions of models vary from only a constant coulomb friction torque, when the angular velocity is zero, to symmetric models where both static (direction dependent) and dynamic (usually ramps) are represented. Various techniques also have been suggested to compensate for frictional effects. For example Wu and Paul [101] introduced a high gain feed back, which is not suitable for when there is a need for linear compensation of small errors. de Silva [17] used an experimental friction model developed by Shibley, in which the frictional torque for a joint is obtained by

$$T_f = f(\dot{\theta}) \times \tau$$

where f is the friction coefficient as a function of angular velocity, and τ is the reaction torque of the joint. He approximates the relationship between f and $\dot{\theta}$ by two straight line segments. Joint reaction torques and angular velocities are then calculated by Recursive Newton-Euler equations. Velocities are used to find the relevant coefficient of friction from the approximated relationship, and this value is multiplied by τ to get T_f. He also points out that, as reaction torques could change due to the presence of friction, one cycle of computation might not be enough for convergence of the values.

An interesting approach for nonlinear compensation of friction is presented by Canudas et al [13] in which, based on experiments on a servo, a model which is asymmetric and includes both Coulomb friction and viscous friction is suggested and an adaptive compensation scheme is developed, so as to cope with the dependency of friction on operation conditions. This model includes variations and asymmetried of the friction torque which are not dealt with in other models. The

model used is:

$$T_f(\dot{\theta}) = \begin{cases} \alpha_1\dot{\theta} + \beta_1, & \dot{\theta} > 0 \\ \alpha_2\dot{\theta} + \beta_2, & \dot{\theta} < 0 \end{cases}$$

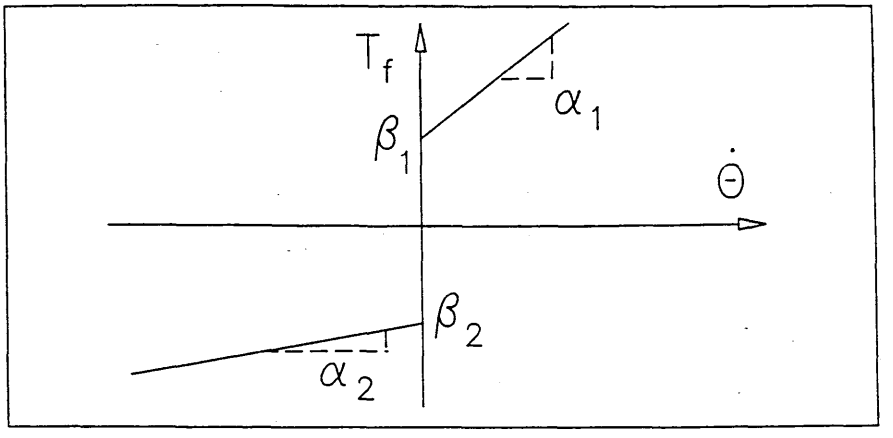


Figure 3.11: Friction model

Then $\alpha 1$, $\beta 1$, $\alpha 2$, $\beta 2$ are estimated. One way of doing this as they suggest, is applying standard linear parameter estimation to the following equation:

$$J\frac{d\tilde{\dot{\theta}}}{dt} = K\tilde{I}(t) - \alpha_i\tilde{\dot{\theta}} - \beta_i$$

where J and K are assumed known, and the current I needs to be measured. Symbol tilde, represents the filtered values. Although they use a tachometer, the values of $\dot{\theta}$, can be found from angular measurements.

This approach seems quite appropriate, and suitable for friction compensation. It should be noted that the method of compensation depends very much on the model chosen.

3.5 Model Validation

So far Kinematics, dynamics and actuation models, particularly relevant to the MA3000 robot, have been discussed. The next step is to make sure that the models developed are accurate enough for the purpose of model based controllers which will be considered in a different chapter. The model validation approach taken is an iterative process in which the behaviour of the model developed, is

compared with the behaviour of the actual robot, and in case of discrepancies, various possibilities, as to the source of errors are considered. Then attempts are made to eliminate these errors, or at least reduce them to an acceptable level. This loop is then repeated a number of times, and if the model does not match the real system, either a different model is employed or the assumptions made about the real system are altered.

A program was written in MATLAB that includes the model and also reads data captured from the real robot, and then by means of exciting the model with the same inputs as the captured data from the robot is based upon, the outputs are graphically compared. The implementation steps are as follows:

1. Read the data file of the first three joint angles of the robot, corresponding to particular voltage inputs.

2. Use a third order filter to filter the data.

3. Define and input values of the motor related parameters, such as the torque constant found in the previous section, resistance of the motors, gear ratios including the pulleys, and so on.

4. Include the kinematic parameters, found using CAM-X and D-H parameters.

5. Input initial values for angle θ, angular velocity $\dot{\theta}$,and angular acceleration $\ddot{\theta}$ and calculate the inertia matrix M and the vector of centrifugal, coriolis and gravity torques Q. These are calculated based on the customised dynamics developed in a previous section and with the kinematic parameters of the previous step.

6. calculate the torque vector τ from

$$\tau_i = \frac{1}{GearRatio} \times \left(K_I \times \frac{V - K_b \dot{\theta}_i}{R_i} \right)$$

where R_i is the resistance of motor i,and V is a voltage input, the same as filtered real values for joints.

7. Then calculate angular accelerations from:

$$\ddot{\theta} = M^{-1}(\tau - Q)$$

8. Integrate $\ddot{\theta}$ to get $\dot{\theta}$ and integrate again to get θ, and save the value of the angles.

9. Return to step 5 and put the values of the previous step in place of initial values and follow on repeating the process for a duration of time, including the same sampling for the integration, as the sampling rate of data acquisition of the real angles.

10. Once the specified duration is over, plot the real angles against the angles obtained from the model.

Figures 3.12, 3.13, show the comparison of the real and the generated data for waist. The first figure is for a pulse-width-modulated voltage input and the

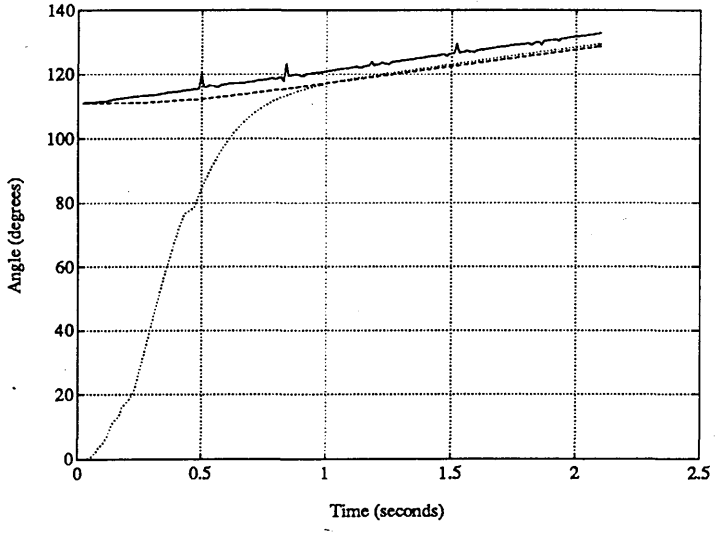


Figure 3.12: Model behaviour and the real robot, WAIST (PWM voltage)

second one is for a constant voltage and current input. The solid line is the real data, dashed line is the data from the model based on the filtered voltage input, and the dotted line is the filtered real data. It can be seen that the model output

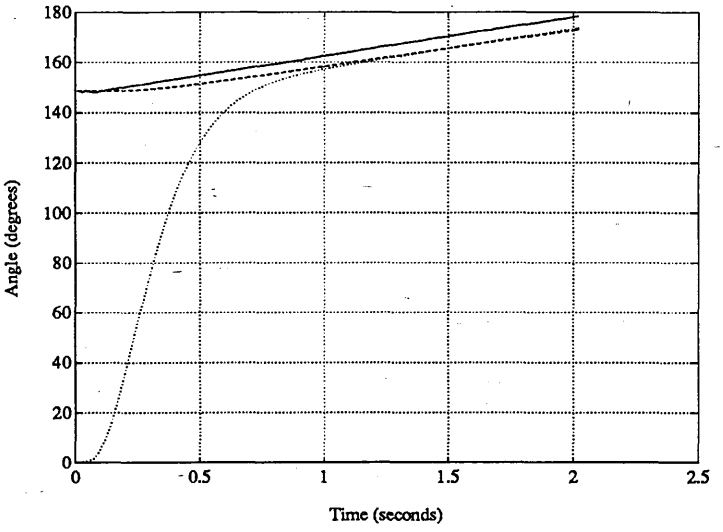


Figure 3.13: Model behaviour and the real robot, WAIST (const. current)

closely matches that of the real data. The small distance between the filtered and real data is due to the data line representing an integrator. Figures 3.14, 3.15, show a similar comparison for the Elbow. Again a close match of the model

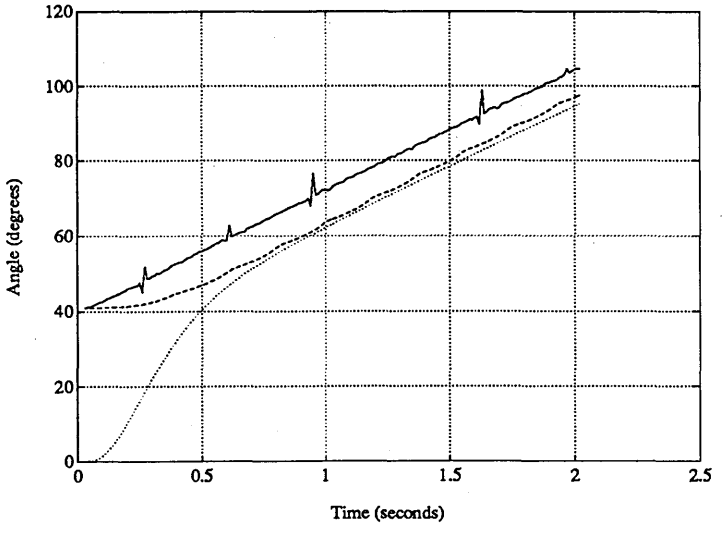


Figure 3.14: Model behaviour and the real robot,ELBOW (PWM voltage)

output with the real data can be seen.

In the case of the shoulder, for a short duration the match is reasonably close, and then it starts to show an oscillatory behaviour. One possible explanation for

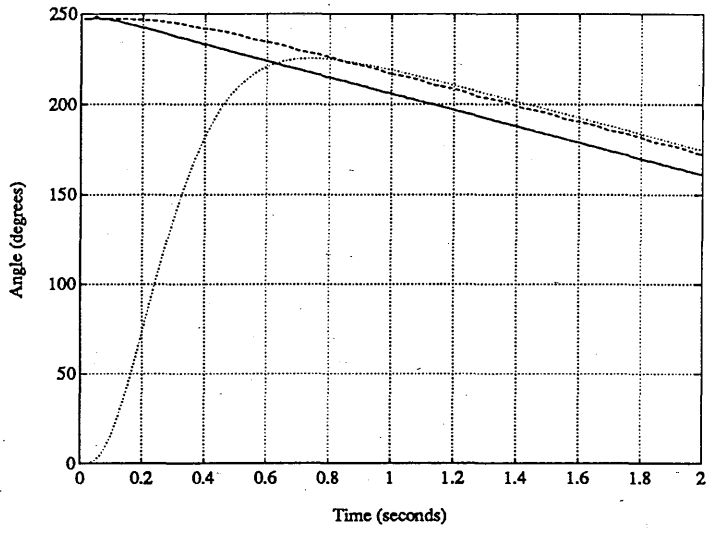


Figure 3.15: Model behaviour and the real robot, ELBOW (const. current)

this is that the kinematic parameters, might not be accurate. As it was explained earlier, the process of model validation is an iterative one. In fact at the beginning, the models for the other links were not so close either, and CAD modelling had to be repeated, a number of times, and each time more details had to be included. The attachment of densities to various parts had to be carefully arranged, so that assumptions of uniform density in some cases could be accurate enough. As the shoulder seems to be the least complex of the three links considered, and the other two give a reasonably correct answer, it is very unlikely that the cause of the mismatch is errors in kinematic parameters. Errors due to incorrectness of dynamic equations is also unlikely, as two approaches, Lagrangian and Newtonian approach, resulted in the same equations of motion. The motor used for shoulder is the same as the one for waist, and the possibility of the motor model or the parameters being wrong is very slim. Ignoring frictional effects and backlash also apply to the waist. The only explanation can be that, as mass of the waist is much greater and inertias are different, whatever effect causes the inaccuracy of the model lies within the fact that they do not dominate the equations for the waist, but do in the case of the shoulder. Even numerical errors can be a possibility.

Chapter 4

Model-Based Identification of Robots

SUMMARY

Some work in parameter estimation of robot manipulator dynamics is reviewed and a new method developed for estimating the mass of any load the manipulator might be holding is presented. The effectiveness of the method is backed by both simulation results and experimental means.

4.1 Introduction

Accurate robot models can be used in the context of adaptive control of robot manipulators, for improving their dynamic accuracy. The parameters of these models need to be fairly accurately obtained for the control schemes based upon them to yield effective results. Mass distribution parameters of a rigid body, namely mass, position of the centre of mass and moments of inertias, are the constituent dynamic parameters and can be obtained by various methods. This was discussed in the previous chapter. One method for finding the values of these parameters, is to employ estimation techniques, following a suitable parameterisation of the dynamic equations of manipulators.

The techniques vary in how the dynamic equations are formulated, and the estimation methods used, as well as what variables need to be measured. The main approach used here is to obtain a linear in the parameters form of the estimation equations, although the dynamic equations of the manipulator are non-linear, and then apply linear estimation methods to estimate the unknown parameters.
Generally, the inertial parameters of the manipulator links need only be obtained once and do not vary. Hence an off-line method for obtaining these parameters is sufficient. However, in various applications, the mass of the load which the robot needs to manipulate might change, or the robot might have to deal with components that have different mass properties. This will affect the inertial parameters of the final link of the manipulator. In this case, an on-line method of estimating the load inertial parameters is needed, to allow compensation of the affect.
Following a literature review of the techniques used for estimation of the load inertial parameters, a new method will be presented for estimation of the load mass, based on developing a linear in the parameter model of a manipulator using symbolic manipulation approach for computational efficiency, and linear estimation techniques. Both simulation and experimental results for the MA3000 robot will then be presented to show the effectiveness of this method. An adaptive controller could be designed to enhance the accuracy of the robot motion based on the developed model with estimation of the load mass. This will be addressed in the next chapter. The load estimator and the controller could operate concurrently, which means that computational speeds can be improved, if parallel processing is utilised. This will be discussed in the chapter on parallel implementation.

4.2 Literature review

Dynamic parameter estimation of Robot manipulators have been the subject of investigation by a number of researchers. Nearly all of these methods rely on simulation to show the convergence of the estimated parameters to their true values.

In majority of these approaches, the computational requirements are the main reason for not being able to implement them on-line.

Similar to their link inertial parameter estimation, which was discussed in the previous chapter, Atkeson et al [4] presented a method of estimating the inertial parameters of a rigid body load using, instead of joint torque, wrist force/torque sensing. In their approach the parameter equation is derived, by relating 3 coordinate systems, (base or inertial coordinate, force reference coordinate system of a wrist force/torque sensor, and principle axis of the rigid body load attached at the centre of mass) and expressing the inertial parameters of the load in terms of the motion of the load and the forces and torques exerted on it, using Newton-Euler equations.

The force and torque measurements by the wrist sensor are obtained and are expressed in terms of the product of known geometric parameters and unknown inertial parameters. Then some notations and quantities are used to formulate these as a system of linear equations with unknown inertial parameters. In addition, measurements or estimates of the position, velocity, acceleration, orientation, angular velocity, and angular acceleration of the force sensing coordinate system are needed to estimate the inertial parameters using Least Squares estimation algorithm. The location of the load's centre of mass, its orientation, and its principal moments of inertia can be recovered from the sensor referenced estimated parameters. These can be used for object recognition and verification. Some other work which utilise joint torque or wrist force/torque sensing to estimate inertial parameters of the load, (mostly simulation) are also discussed in the above reference.

By effective exploitation of the structure of manipulator dynamics, and employing the approach of Atkeson et al, that the dependence of the system dynamics on the unknown parameters can be made linear in terms of a suitably selected set of robot and load parameters, Slotine and Li [86] estimated the unknown manipulator and payload parameters on-line in a full dynamics feedforward compensation

routine for an adaptive robot control algorithm.

Equivalent parameterisation was introduced by Goodwin et al [67] to obtain dynamic equations that are linear in the unknown parameters for a general rigid body. In their method, noting that there are ten free parameters in a general 3D body, suitably chosen fixed position vectors are used and the rigid body is parameterised by ten point masses. These parameters are related to the mass of the body, centre of mass vector times mass and inertia matrix, by a linear transformation. This transformation can be made non-singular by a suitable choice of position vectors. Measurements of the motor torques, and joint position and velocities are required for obtaining the unknown parameters. The significant advantage of the method is that acceleration of the manipulator need not be measured.

Employing this approach, Walker [97] investigated the on-line estimation of the load-mass and its moments based on measurements of the manipulator joint position, velocities and the force exerted by the end-effector on the load. The simulation results presented, show that their algorithm performs reasonably well, but the computational load is fairly high.

A manipulator terminal-link parameter estimation method is presented by [42] in which an Instrument Variable Method (IVM) is used to estimate the unknown parameters.

They use IVM to overcome inaccuracies which result from random noise involved in joint acceleration and joint torque measurements associated with algorithms such as [65] that is based on the least squares method, and uses measurements of the joint position, velocity, acceleration and joint torque.

In their approach, the dynamic equation of a serial-rigid link manipulator is written in terms of the unknown composite dynamic parameter vector which consists of the mass, center of mass, inertia tensor, viscous damping, friction coefficient and the Coulomb friction torque for each link. Since this equation is non-linear, they use Denavit-Hartenberg Convention of coordinate systems (body attached frame) to formulate the linear equations in unknown dynamic parameters from

the Newton-Euler formulation. Based on the assumptions that : the geometric and the actuator parameters are known exactly, the joint position and velocity measurement noises are negligible, the joint acceleration measurement noise is white noise which has a zero mean, and the joint torques are calculated from the actuator equation which are linear with respect to joint acceleration, the composite dynamic parameters are estimated by IVM.

The criterion to be minimized has a weighting matrix. In the case of least squares method, all the elements of the weighting matrix are one. Due to the correlation of two members of the estimation equation, least squares does not yield the true values of the unknown parameters. Simulation results show this. However experimentally least squares and IVM converge to the same value. They conclude that this is due to the acceleration measurement noises being small. Also their experimental results show that if the magnitude of variation of desired trajectory is small, the estimated values show better agreement with the theory. The instrument variable which is correlative to acceleration of joints is obtained from the instrument model, by assuming the dynamic equation of each link motion is linear and time-invariant. In addition to the terminal link parameter variation, some parts of the nominal composite parameter errors must also be estimated in some situations. Real-time estimation calculations need a dedicated VLSI chip.

To do away with the fact that most commercial robots do not provide for internal measurements of torque, velocity or acceleration; and even when they do, the measurements are in most cases inaccurate and noisy, Raucent et al [76] presented an external measurement approach to identify the mass distribution parameters of robotic manipulators.

The method is based on an auxiliary base reaction model which is linear with respect to the mass distribution parameters and completely independent of joint forces and torques, including friction effects. Six generalized coordinates representing three angular and three linear displacements of the base of the robot are added and hence the degrees of freedom become (n + 6). The robot is placed on a

force sensing platform, and six components of reaction torques and forces between the bedplate and the robot are measured. A high precision vision device is also used to accurately determine the motion of the robot i.e. position, velocity, and acceleration.

The method allowed most of the mass distribution parameters to be estimated, as long as persistently exciting trajectories existed.

For simulation of weightless conditions on space emulators, similar methods for estimating some combination of mass properties of manipulators by measuring the reaction moments of their base, have been used. In these methods the mass properties identified are not sufficiently complete for dynamic control techniques, but allow compensation for the gravitational load on the links of the manipulators.

There has not been any published material to the best knowledge of the author, as to any work based on this method to estimate the load mass.

A method consisting of three types of tests performed without decomposing the manipulator into parts, was introduced by Mayeda et al [65] in which first of all static tests are performed, where the manipulator is made to stand still in various configurations, then constant angular velocity for one joint at a time and accelerated motions are carried out. The data which is acquired from the motion is then used to estimate the coefficients of the dynamic equations. This method is only applicable to manipulators with any two adjacent joint axes, parallel or orthogonal. They make the assumption that Coulomb friction force is constant during the test motion, which is critical to the accuracy of the identification. This only applies to a class of manipulators.

One difficulty with the method itself is to find suitable accelerated motion tests such that the whole essential coefficients of the non-linear dynamic model of the manipulator can be determined from the estimated coefficients of the tests.

There is no reference to load mass estimation in this work.

The state variable model of a robot manipulator is linearised around a chosen trajectory, assuming small perturbations, and the discretised version of the steady state innovation of the linearised manipulator is considered by Mahalanabis and Galou [63] for recursive parameter estimation. An observer form of the model is used for the development of an on-line algorithm for the estimation of the model parameters and for subsequent state estimation. In this work the results are illustrated through simulation studies of a three link manipulator, and there is no mention of load mass estimation.

The estimation of unknown payload mass has been addressed in the context of non-linear adaptive control too. Hemami et al[29] developed an adaptive control algorithm with a non-linear reference model for an N-link planar robot with an unknown load. The trajectory error in the vertical direction which represents the potential energy difference between the actual motion and the desired motion, is used to estimate the mass of the load. Computation times need to be reduced, before this algorithm can be implemented in a real-time environment.

4.3 Load Mass Estimation

Parameter estimation can be used in the case of incomplete a priori specification of the dynamics of the manipulator. We specifically consider the case where the load mass is unknown. The idea is to provide the load mass estimates which could be used by a controller to maintain a desired response. This is achieved by using an on-line estimator to provide periodic updates of the estimates of the mass of the unknown load that the gripper holds.

The load can be considered as having the effect of an unknown, possibly time-varying, gravitational force applied to the arm end, as well as inertia effects. This can be extended to the case where the force can be applied in other directions, apart from the vertical. So that we are able to track a slowly varying force applied to the end effector. This method of estimation contrasts with the methods based

on the learning theory of for example [41] which typically requires several cycles of operation.

Here a new method is presented in which a software package for symbolic manipulation of equations (REDUCE) is used to generate the n×n generalized inertia matrix and the vector, containing the centrifugal, coriolis, gravitational torques in symbolic terms, from the forward and backward equations of the recursive N-E dynamic equation for serial-rigid manipulators or from Lagrangian approach, similar to the way discussed in the previous chapter.

An extra link of zero length, but of finite mass is included in the definition of the links which represents the load mass carried by the gripper. This will be the unknown parameter which will be estimated. The equations are linear in the unknown parameter and least-squares is used for estimation.

Simulation and experimental results show that the estimated parameter converges to its true value of the load mass in a short period.

4.3.1 On-line adaptive estimation of load mass

In the context of Self-Tuning Adaptive Control, it is recognised that there are two stages of parameter estimation [93]. In the first stage, from the dynamic model of the system, parameter estimation equations are derived and then in the second stage, an estimation method is used to estimate the parameters of the model.

Method of linear Filters, where the linear dynamic operator is a low-pass filter is one approach for deriving the parameter estimation equations. Using the state-variable filter approach for estimating parameters of continuous-time transfer functions is described in [27] etc.

Estimation Procedure

Based on the state-variable filter approach for estimating parameters of continuous-time transfer functions, the dynamic equation of the Robot Manipulator is formulated in a way that, although nonlinear, the parameter estimation equations are linear-in the-parameters. Then a standard least-squares estimation can be used to estimate the load mass.

Using the customised robot dynamic algorithm based on the recursive N-E equations of motion, the dynamic equations for a four link manipulator, with the final link represented as a link of zero length and finite mass say m_4 is obtained. The explicit expressions for the inertia matrix M and the torque vector Q are shown in Appendix D.

As the centrifugal, coriolis and gravity torque associated with the final link, as well as the corresponding inertia matrix member and joint acceleration are zero, the equations are linear in the parameter m_4:

$$\underline{\tau} = \underline{M}_1(\theta)m_4\ddot{\underline{\theta}} + \underline{M}_2(\theta)\ddot{\underline{\theta}} + \underline{Q}_1(\theta,\dot{\theta})m_4 + \underline{Q}_2(\theta,\dot{\theta}) \tag{4.1}$$

where

$\underline{\tau}$ is a 3×3 vector of torques

$\underline{M}_1(\theta, m_4)$ is a 3×3 inertia matrix which is a function of joint angle θ and load mass m_4 . It is linear in m_4 .

$\underline{M}_2(\theta)$ is a 3×3 inertia matrix, not a function of m_4

$\underline{Q}_1(\theta, \dot{\theta}, m_4)$ is a 3×1 vector of centrifugal, coriolis and gravity torques. It is linear in m_4

$\underline{Q}_2(\theta, \dot{\theta})$ is a 3×1 centrifugal, coriolis and gravity torques, not dependent on m_4

$\ddot{\underline{\theta}}$ is a 3×1 joint acceleration vector.

If the load mass is represented as added mass on the third link, then the customised robot dynamic equations are no longer linear in the parameter. This is due to the fact that, as load mass varies, so does the position of the center of

mass of the third link. And this means that the estimation method will not apply.

Hence using the linear in the parameter equations and taking the terms that depend on m_4 to one side, equation 4.1 becomes:

$$\underline{\tau} - (\underline{M_2}\ddot{\underline{\theta}} + \underline{Q_2}) = (\underline{M_1}\ddot{\underline{\theta}} + \underline{Q_1})m_4 \tag{4.2}$$

Now let

$$\underline{\tau} - (\underline{M_2}\ddot{\underline{\theta}} + \underline{Q_2}) = \begin{bmatrix} \psi_a(t) \\ \psi_b(t) \\ \psi_c(t) \end{bmatrix}$$

and

$$\underline{M_1}\ddot{\underline{\theta}} + Q_1 = \begin{bmatrix} x_a(t) \\ x_b(t) \\ x_c(t) \end{bmatrix}$$

Then

$$\begin{bmatrix} \psi_a(t) \\ \psi_b(t) \\ \psi_c(t) \end{bmatrix} = \begin{bmatrix} x_a(t) \\ x_b(t) \\ x_c(t) \end{bmatrix} m_4 \tag{4.3}$$

There are a number of ways to estimate the value of m_4 from equation 4.3. One way would be to take one component of the equation for example

$$\psi_a(t) = x_a(t)m_4$$

and based on this obtain the data and output vectors with members corresponding to each sample interval. The following equation is formed

$$\begin{bmatrix} \psi_a(t_1) \\ \psi_b(t_2) \\ \vdots \\ \psi_c(t_n) \end{bmatrix} = \begin{bmatrix} x_a(t_1) \\ x_b(t_2) \\ \vdots \\ x_c(t_n) \end{bmatrix} m_4 + \begin{bmatrix} e(t_1) \\ e(t_2) \\ \vdots \\ e(t_n) \end{bmatrix} \tag{4.4}$$

where $e(t_1) \cdots e(t_n)$ represent the errors.

or

$$\underline{\Psi} = \underline{X}m_4 + \underline{E} \tag{4.5}$$

note that m_4 is a scalar.

Then using the least squares method, the criterion function to be minimised is

$$R = \underline{E}^T \underline{E} = (\underline{\Psi}^T - \underline{X}^T m_4)(\underline{\Psi} - \underline{X} m_4) \tag{4.6}$$

multiplying

$$R = \underline{\Psi}^T \underline{\Psi} - \underline{\Psi}^T \underline{X} m_4 - \underline{X}^T m_4 \Psi + \underline{X}^T m_4 \underline{X} m_4$$

as $\underline{\Psi}$ and $\underline{X}$ are both vectors, then $\underline{\Psi}^T \underline{X} = \underline{X}^T \underline{\Psi}$, therefore

$$R = \underline{\Psi}^T - 2\underline{\Psi}^T \underline{X} m_4 + \underline{X}^T \underline{X} {m_4}^2$$

The value $\hat{m}_4$ which minimises R makes the gradient of R with respect to X zero:

$$\frac{\partial \underline{R}}{\partial \underline{X}} = 0$$

using a standard result for derivatives of vector expressions, i.e.

$$\frac{\partial \underline{B}^T \underline{A}}{\partial \underline{A}} = \underline{B}$$

we have

$$-2\underline{\Psi}\hat{m}_4 + 2\underline{X}\hat{m}_4^2 = 0$$

or

$$\underline{\Psi} = \underline{X}\hat{m}_4$$

pre multiplying by $\underline{X}^T$

$$\underline{X}^T \underline{\Psi} = \underline{X}^T \underline{X} \hat{m}_4 \tag{4.7}$$

therefore

$$\hat{m}_4 = (\underline{X}^T \underline{X})^{-1} (\underline{X}^T \underline{\Psi}) \tag{4.8}$$

and hence $\hat{m}_4$ which is the estimated value of m_4 can be found using least squares estimation.

Another way of estimating m_4 from equations 4.3 is to take all, as opposed to

one component of the equations and obtain the data and output vectors for each sample interval to form

$$\begin{bmatrix} \psi_a(t_1) \\ \psi_b(t_1) \\ \psi_c(t_1) \\ \vdots \\ \psi_a(t_n) \\ \psi_b(t_n) \\ \psi_b(t_n) \end{bmatrix} = \begin{bmatrix} x_a(t_1) \\ x_b(t_1) \\ x_c(t_1) \\ \vdots \\ x_a(t_n) \\ x_b(t_n) \\ x_c(t_n) \end{bmatrix} m_4 + \begin{bmatrix} e_a(t_1) \\ e_b(t_1) \\ e_c(t_1) \\ \vdots \\ e_a(t_n) \\ e_b(t_n) \\ e_c(t_n) \end{bmatrix} \tag{4.9}$$

and proceed similarly to the previous approach. The results in both cases are similar and this can be verified by simulation.

In the case of a real robot, position sensors can be used to obtain joint positions. After filtering these, joint velocities and accelerations are calculated by differentiation. The corresponding input torques applied to joints are also filtered using a third order filter

$$C(s) = s^3 + c_1 s^2 + c_2 s + c_3$$

Then the values of filtered torque, filtered angle, angular velocity and acceleration are used in the customised dynamic formulation to obtain the output and data vectors. Least squares is then used to estimate the load mass ie. $\bar{m}_4$. Recursive Least Squares can also be used for estimation, making use of the solutions of previous equations and hence reducing the computational time. In situations where position data is obtained and processed in parallel in order to increase computational speed, the Recursive LS can still be used, as the data and output vectors have to be formed sequentially. The need for sequential formation of these vectors are due to the fact that, interaction of the joints are taken into account and as a result, information about other joints is needed in order to calculate inertias and other quantities due to one joint. In the case of recursive LS, one

component of equation 4.3 i.e.

$$\psi_a(t) = x_a(t) \times m_4$$

is used at each sampling interval. Hence the recursive equations will be :

$$\hat{m}_4(N+1) = \hat{m}_4(N) + k(N) \times [\psi_a(N+1) - x_a(N+1) \times \hat{m}_4(N)] \quad (4.10)$$

$$k(N) = P(N) \times x_a(N+1) \times [1 + x_a(N+1) \times P(N) \times x_a(N+1)]^{-1} \quad (4.11)$$

$$P(N+1) = [1 - k(N) \times x_a(N+1)] \times P(N) \quad (4.12)$$

And for time varying load mass, a loss function with exponential weighting is used i.e.,

$$R = \sum_{t=1}^{N} \lambda^{N-t} \times [\psi_a(t) - x_a(t) \times m_4]^2 \quad (4.13)$$

where λ is the "forgetting factor". The least squares estimate is then given by replacing equations 4.11 and 4.12 by the following two equations respectively.

$$k(t) = P(t) \times x_a(t+1) \times [\lambda + x_a(t+1) \times P(t) \times x_a(t+1)]^{-1} \quad (4.14)$$

$$P(t+1) = [1 - k(t) \times x_a(t+1)] \times P(t)\lambda \quad (4.15)$$

where λ is less than one. Usually about 0.98 .

4.4 Simulation

The simulation is based on the first method of estimating the load mass described in the previous section. It consists of five parts:

First, the definition of desired trajectory for each joint and calculation of the corresponding torques, based on the dynamic equation of the robot.

Then in the second part, the values of torque are used in a robot simulator, in order to obtain the position, velocity and acceleration of each joint. Although in the first part, the torque values were based on one value of load mass, here the load mass is varied. In the third part, the torques, angles, velocities and accelerations are filtered using a third order filter. To implement the state variable

approach, the filter transfer function is represented in state space and then the discrete forms of A, B, C, D are found. Then states (filtered position, velocity

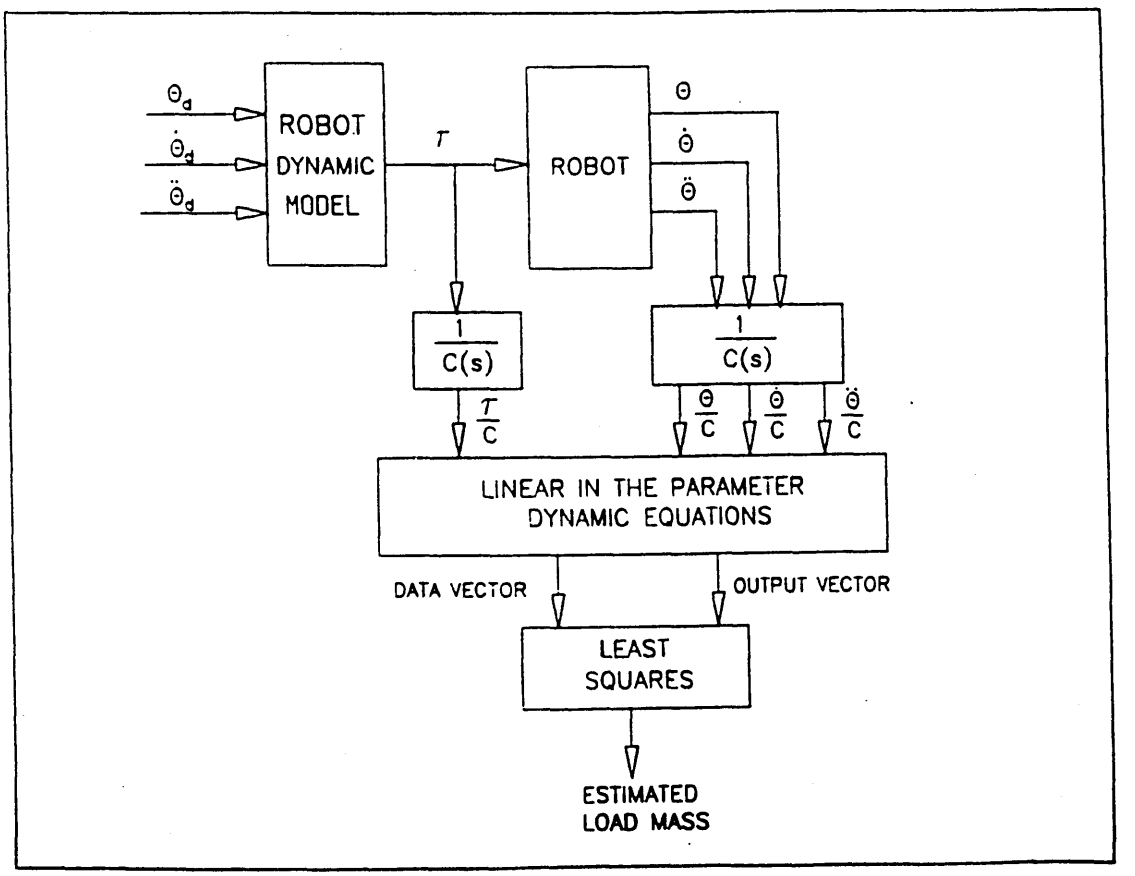


Figure 4.1: Load Mass Estimation

and acceleration) are updated [1].

Then customised dynamic equations which are linear in the parameter, as explained earlier are used to form the data and output vectors.

Finally, least-squares estimation is used to estimate the load mass.

A block diagram representing this procedure is shown in Figure 4.1.

4.5 Implementation

A REDUCE program that takes as input the number of links and the definitions of kinematic and inertial parameters of a manipulator, was written to symbolically

[1]The details of the state variable approach can be found in reference [24] etc.

manipulate the N-E equations of motion for robot manipulators and perform all algebraic simplifications possible.

Four links were defined, the final being of finite mass and zero length, representing the load mass held by the gripper of the manipulator (strictly speaking, the load mass also includes the mass of the wrist).

The outputs of the program were the elements of the inertia matrix as well as the vector of centrifugal, coriolis, and gravitational torques.

A factorisation was then performed to separate the terms that depend on the load mass, hence the following were obtained:

- $\underline{M}_1(\theta)$, $\underline{M}_2(\theta)$
- $\underline{Q}_1(\theta, \dot{\theta})$, $\underline{Q}_2(\theta, \dot{\theta})$,

The terms with subscript 1 are dependent on the load mass. These correspond to equation 4.1.

The explicit expressions for these terms were then modified to a MATLAB readable format.

A function was then written in MATLAB that uses these expressions together with the values of kinematic and inertia parameters of the MA3000 robot, with external input of the joint angles and angular velocities, to output the actual values of the above terms.

This function was used in a MATLAB program to obtain the joint torques corresponding to a defined trajectory (angular displacement, velocity and acceleration of each joint) according to equation 4.1. A known load mass (e.g. 5 Kg) was used in the calculations.

Figures 4.2 and 4.3 show the defined trajectory, as well as the resultant torques for a 5 Kg load mass.

Then, using the first set of values of joint torques obtained from the previous stage, and some initial values for angular position and velocity of the joints, in the MATLAB function described above, the first values of angular accelerations

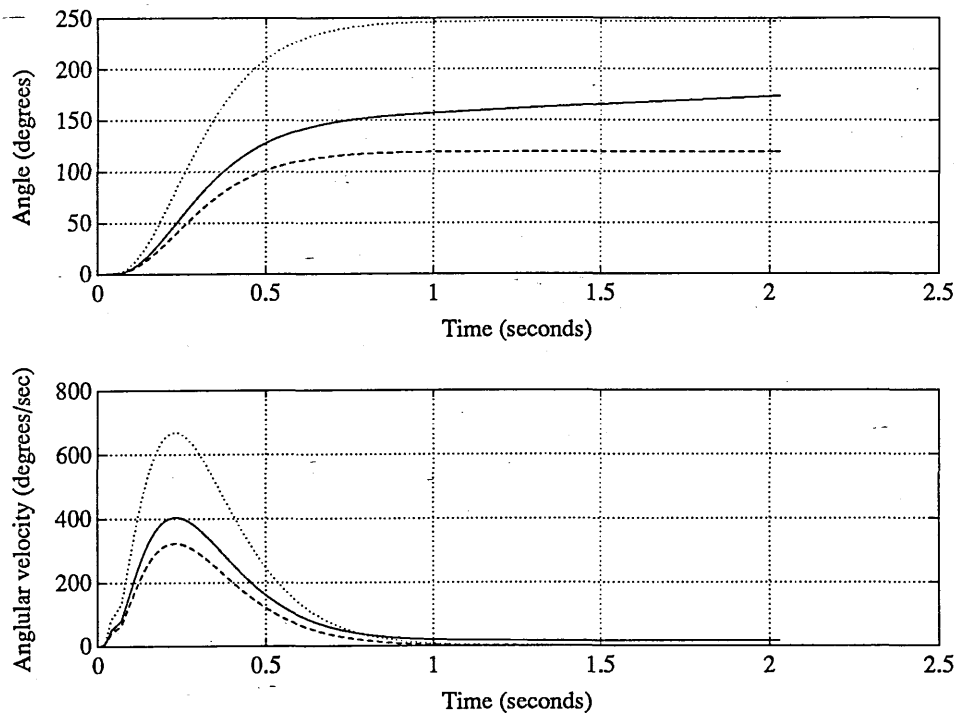


Figure 4.2: Angular position and velocity, against time

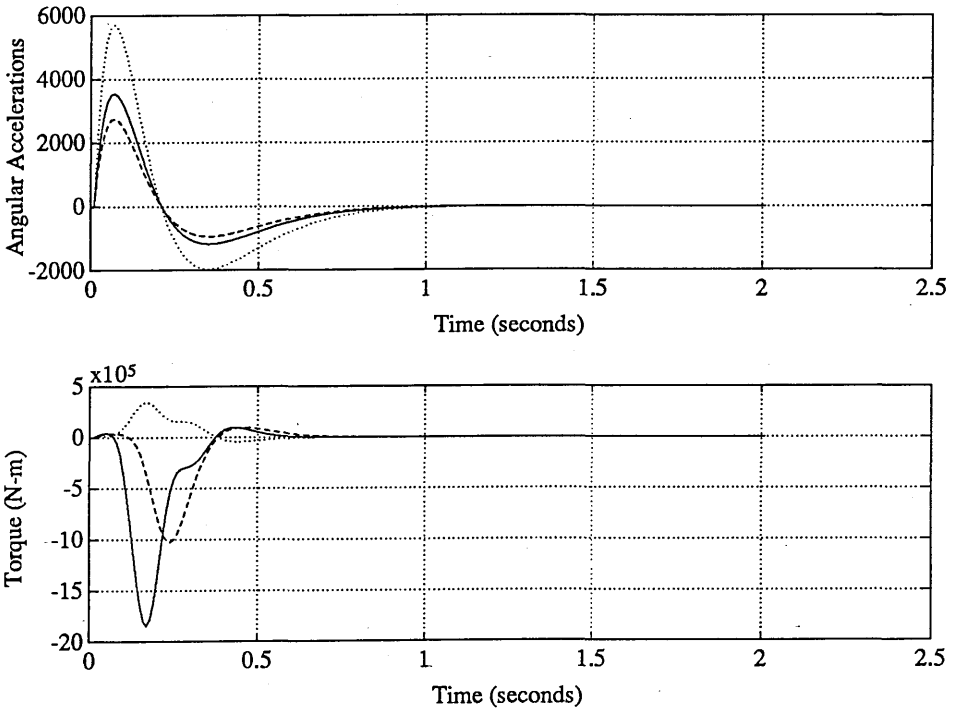


Figure 4.3: Angular accelerations and Resultant Torques

are calculated according to

$$\underline{\ddot{\theta}} = (\underline{M}_2 + \underline{M}_1 m_4)^{-1}[\underline{\tau} - (\underline{Q}_2 + \underline{Q}_1 m_4)] \tag{4.16}$$

By integration, values of angular position and velocity are found from angular accelerations, which are then used in the next time step, and the process is repeated for all the values of joint torques.

This implements a robot simulator to which, joint torques are input and form it angular acceleration, velocity and position is obtained. However, the load mass m_4 is varied from $0Kg$ to $5Kg$ in $0.5Kg$ increments as the process is repeated, this is despite the fact that the values of the joint torques correspond to a 5 kg load mass. This introduces perturbations from the original joint torques.

The values of the torques, angular position, velocity, and acceleration are then filtered using the Lsim function in matlab toolbox for simulation of continuous-time systems.(a third order filter is used).

Then the data vector and the output vector is found and the equations of motion are written in the standard form of least squares to estimate the unknown parameter, namely the load mass.

Figure 4.4 shows the values of Load Mass and their estimation. As can be seen the estimated values that are shown as dashed lines are very close to the actual values. One point to note is that the original value of the load mass on the basis of which the torques were generated, are important not to be too large or too small compared to the estimated values, otherwise the estimation will not be accurate. Simulation results of when recursive LS is used for estimation, also show that the values of the estimated load mass converge to their true value after a few samples. Having been able to show that, it is possible to estimate the varying load mass that a gripper carries, control algorithms that utilise this can be implemented for various applications. The algorithm can also be extended to the case of varying reaction forces on the gripper, when the equations are formulated to include three components of force/torque, as opposed to just gravity torque.

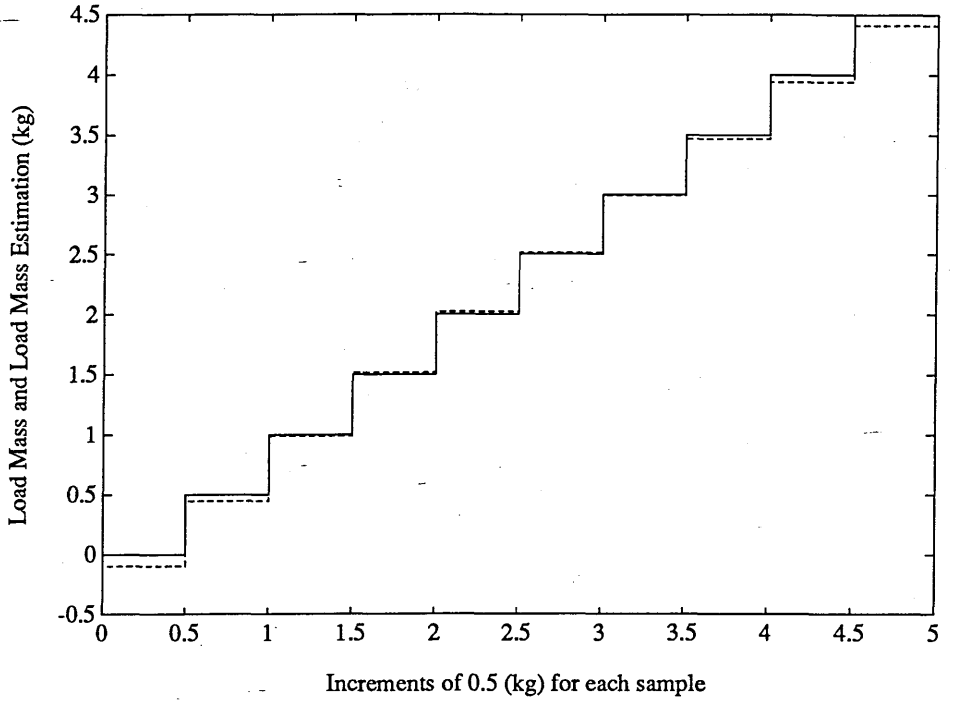


Figure 4.4: Comparison of Load Mass and estimation values

The next step is to verify these results experimentally, using the MA3000 robot manipulator.

4.6 Experimental Results

Step voltage inputs were applied to the waist joint motor of the robot, and angular position θ of each joint was obtained from plastic film potentiometers with linearity of $\pm$ 0.25 % measured to 12 bit resolution, and saved in data files. The operation was repeated for various load masses, attached to the gripper of the robot.
A third order filter was chosen

$$\frac{1}{a_0 s^3 + a_1 s^2 + a_2 s + a_3}$$

and this representation was changed to state-space and discretized to get

$$X_{i+1} = A_d X_i + B_d u \tag{4.17}$$

After including the initial values, at each sampling interval, the input $u = \theta$ is used to get

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where x_1 is filtered acceleration, x_2, filtered velocity and x_3 filtered angular position. This is repeated for each sample of 100 HZ. In the implementation, $a_0 = 0.001$, $a_1 = 0.03$, $a_2 = 0.3$, and $a_3 = 1$ were chosen empirically, to give good results.

Figure 4.5 shows a block diagram representation of the above procedure. Once these filtered values were obtained, motor torques were calculated, using

$$T_m = \frac{V - k'\dot{\theta}_m}{R} k'$$

where k' corresponds to the motor torque constant which is equal to the back emf constant, at steady speeds (the values of these were found to be 0.01, 0.17, and 0.009 for waist, shoulder and elbow respectively), $\dot{\theta}_m$ is the angular velocity of the motor, R, the motor resistance, and V the voltages applied.

Then the contribution of the load torques are included (i.e. $\frac{1}{n}T_L$, where n represents the combination of gear and pulley-belt system ratios) in the equation of motion:

$$T_m - \frac{1}{n}T_L = J_{eff}\ddot{\theta}_m$$

However to calculate T_L, the value of load mass say m_4 need to be known, and as this is the value to be estimated, the torque is divided into two parts, one which depends on the load mass, T_1, and another that does not, T_2. Then

$$J_{eff}\ddot{\theta}_m + \frac{1}{n}T_1 - T_m = T_2 m_4 \tag{4.18}$$

letting

$$\underline{X} = T_2$$

$$\underline{\Psi} = J_{eff}\ddot{\theta} + \frac{1}{n}T_1$$

where $\underline{\Psi}$ is the output vector, and $\underline{X}$ the data vector. Least squares estimation is used to find m_4 .

A number of different voltages were applied, both pulse width modulated, and constant step inputs, to individual joint motors for various load masses and angular outputs were used in the above algorithm to estimate the load masses.

The results show that, the best estimates are obtained when the waist data is used for estimation, this is probably due to the fact that the rotation of this joint is about the vertical axis, which means that the affect of variation of gravitational torques is less, compared with other joints.

Also it was found that applying higher voltages which effectively meant higher accelerations, resulted in better estimates. Also PWM signals, caused the angular

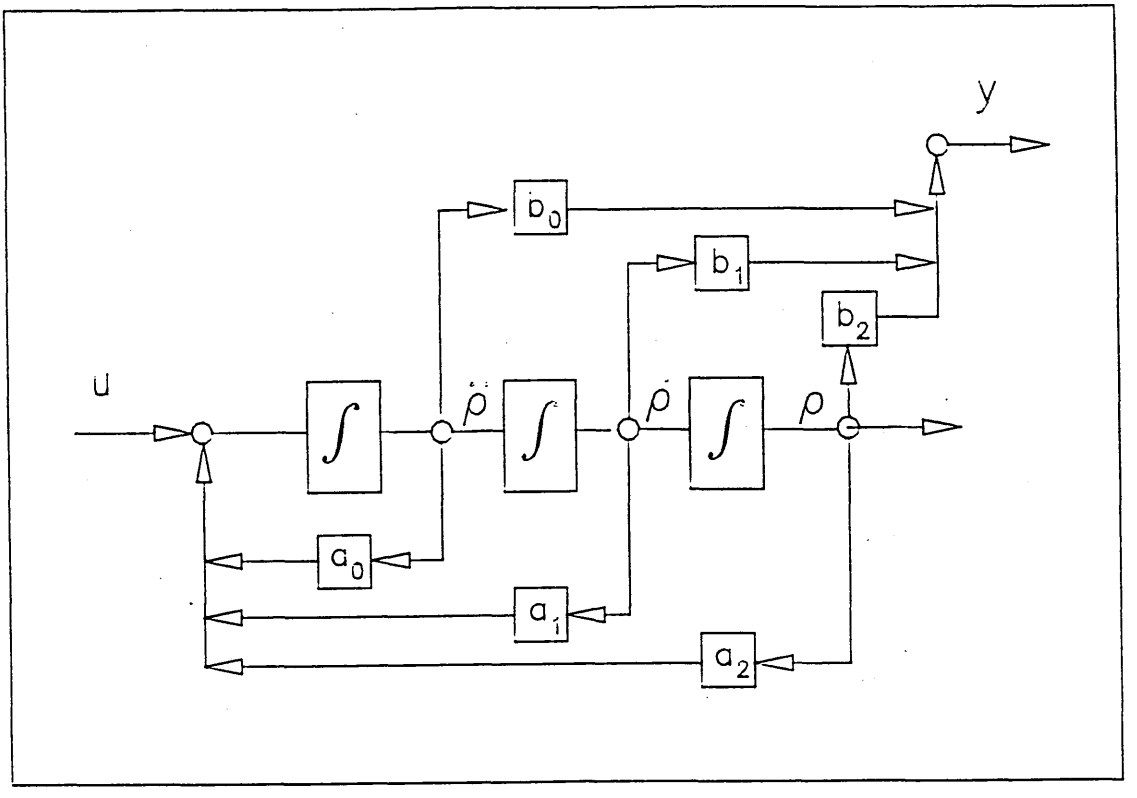


Figure 4.5: filtered values of accn., vel., and angle from angle input

position data obtained to be noisier, compared to constant voltage which, using

Voltage applied	Real Load Mass (kg)	Estimated Load Mass(kg)
88.5v constant	0	0.0319
88.5v constant	0.908	0.9457
64v PWM	0	0.4401
64v PWM	0.908	1.247

Table 4.1: Comparison of Real and Estimated Load Mass

a different amplification hardware that will be explained in a future chapter, provides constant current input.

The noisier data also contributed to the estimates, being less accurate. Table 4.1 shows as an example that higher voltage of 88.5 volts and constant current, gave better estimated results for both 0 kg and 0.908 kg, compared to 64v PWM voltage input.

However even the less accurate estimates were quite reasonable considering the sources of errors which will be discussed in the next section.

4.7 Discussion

It is certainly true that identification of the link parameters of robots tends to be a on-off task in general, as their value do not change greatly, to have to calibrate them frequently. And there are other methods other than estimation, which at times are more convenient and serve the purpose, to produce accurate enough results, such as the CAD approach discussed in the previous chapter.

Nevertheless the load mass that the robot carries can vary considerably during operations, and it is useful to be able to estimate the value of the load mass and hence the affect of its variation on the terminal link of the manipulator.

Some work was discussed, where by the parameters of the mass distribution of the load are estimated. Amongst these, the work of Atkeson et al [4] seem to be one in which, there are reasonable implementation results to be able to see the effectiveness of their approach.

In the new method introduced above, only the value of the load mass are estimated, as opposed to the cases where moments of inertia and centre of mass parameters are also estimated. It should be noted that it is possible to formulate the equations in such a way that they are linear in one of the unknown parameters. The use of symbolic manipulation to derive these equations, is important.

Attention should also be drawn to the fact that from the findings of Atkeson et al, the estimation of moments of inertia parameters, despite the requirement of extensive computational efforts, does not lead to accurate estimates, and even in some cases, for example its application to the PUMA robot, needs special test movements, as the torque due to gravity for example is about 40 times greater than the torque due to the maximum angular acceleration. Acceleration capacity requirements for the algorithm to yield reasonable results, were higher than the acceleration capability of the PUMA. Loads specifically designed to produce large moments of inertia, had to be used, for testing the estimation algorithm. And slight noise in the data resulted in poor estimates.

They finally concluded that for control purposes, even poor estimation of the moment of inertia parameters, will allow good estimates of the total force and torque necessary to achieve a trajectory.

Although their algorithm worked better for the case of their Direct Drive Arm to estimate the moment of inertia parameters, it raises the question that in large majority of industrial robots the impracticality and inconvenience of using direct drive arms, together with having to set up force sensors and providing large computational means of implementing the algorithm, justifies the ability to inaccurately estimate the moment of inertia parameters, which even if their estimates are not accurate, will not affect the performance of their controller anyway.

The semiconductor strain gauges used for force sensing are prone to drift and frequent calibration of them is necessary to prevent additional sources of error in the estimation.

The algorithm introduced, that estimates the load mass, only requires angular

position sensing, it is not computationally intensive, and it gives reasonable estimates, that can be improved when

- A smooth high order polynomial trajectory is used to minimize the mechanical vibrations and not excite unmodelled dynamics
- The kinematic and dynamic parameters obtained are as close to their true values as possible
- Noise of the data obtained are reduced
- A smooth voltage input is used
- Data for the waist is used in the algorithm, so that the estimated values are closer to the true value of the load mass

In addition, there is a possibility that other parameters of the load can also be estimated, one at the time, using symbolic manipulation of the equations.

Chapter 5

Transputer-Robot Interface

SUMMARY

A description of the hardware units which were build in order to operate an MA3000 robot using a network of transputers, independent of its standard controller is given. The corresponding interface software developed is also presented. The transputer network consists of a Parallax system which includes an ADC and a DAC, and a Meiko computing surface runing OPS, MeikOS and CS Tools.

5.1 Introduction

When a high speed operation is required in a robotic application, increased joint velocities imposed on the manipulators result in an increase in the effects of coupling between the joints.

More joint interaction certainly brings about a need for employment of advanced controllers that are able to compensate for the effects that can cause loss of performance and accuracy.

Model-based controllers that incorporate a complete dynamic model of the manipulator including joint interactions, are seen as ideal schemes to achieve high performance. However a combination of high computational requirements of these

methods and lack of accurate manipulator models, has made their real-time implementation prohibitive.

The motivation behind building an interface between the MA3000 robot and the transputer network which is described in this chapter is to be able to exploit the capabilities of the transputer as a fast processing unit and as part of a network for implementing advanced control algorithms in real-time.

Although simulation results can give a lot of insight as to the suitability and effectiveness of a particular algorithm, there are various factors that only practical implementation can truly incorporate.

In this chapter, following a description of the standard interface unit of the MA3000 to a host computer and outlining its limitations, a brief overview of the transputer and, the occam language with its unique features for programming concurrent systems, designed for implementation on the transputer, is given to justify their choice. Then, the three components which form the controller of the robot, , namely a Parallax System, a Meiko Computing Surface with provisions to communicate with the parallax, and an electronic interface unit between the ADC and DAC of the parallax, and the joint motors of the robot, are dealt with.

5.2 Standard interface

The controller of the MA3000 TecQuipment Robot is connected to an IBM PS/2 host computer, via an interface unit, which provides communication, using the 8255 Peripheral Interface Adaptor (PIA) and associated circuitry, resident in the backplane assembly.

It is possible to develop basic PID (Proportional, Integral, Derivative) algorithms to control individual joints. The operating software of the controller, and the default values for the PID control parameters are held in ROM.

The controller, as well as the host computer can access areas of RAM used as a work space and shared memory, which is used to relay control information to the

controller CPU (a 6502 microprocessor is used).

A 256 byte memory slot is required for memory mapping, which is generated as a pseudo memory expansion bus, using programmable input/output ports. This is slower than direct memory mapping and restricts the maximum speed of the robot joints. The host computer addresses the area with the interface card.

The interface unit is also used for host communication with two sets of read write latches, used for process inputs and outputs, also for keyboard and front panel controls

One limiting factor in using this set up to implement novel real-time controllers, which are capable of taking the interaction of the joints into account and allow tuning of the controller parameters, is the computational speed. Considerably higher processor speeds are required to deal with the computational burden that this class of controllers impose.

A choice which can help achieve the speeds required, is the Inmos Transputer, which is a high performance single chip computer, with its capabilities as a fast single processing unit and whose architecture facilitates the construction of parallel processing systems. Different parallel architectures could be devised to suit various control algorithms, hence making the real-time implementation possible.

5.3 The Transputer and Occam

The Inmos Transputer, is a general purpose, very high speed processor, designed as a processing element which can be connected to other transputers, to provide a concurrent processing environment with extensive capabilities.

In addition to the high processing speeds attainable, either as a single unit or part of a network, it has a number of unique features. Figure 5.1 shows a block diagram of the T800 Transputer. The processor is 32 bit wide and it occupies 25 % of the total silicon area [35]. The following are some special features of the Transputer:

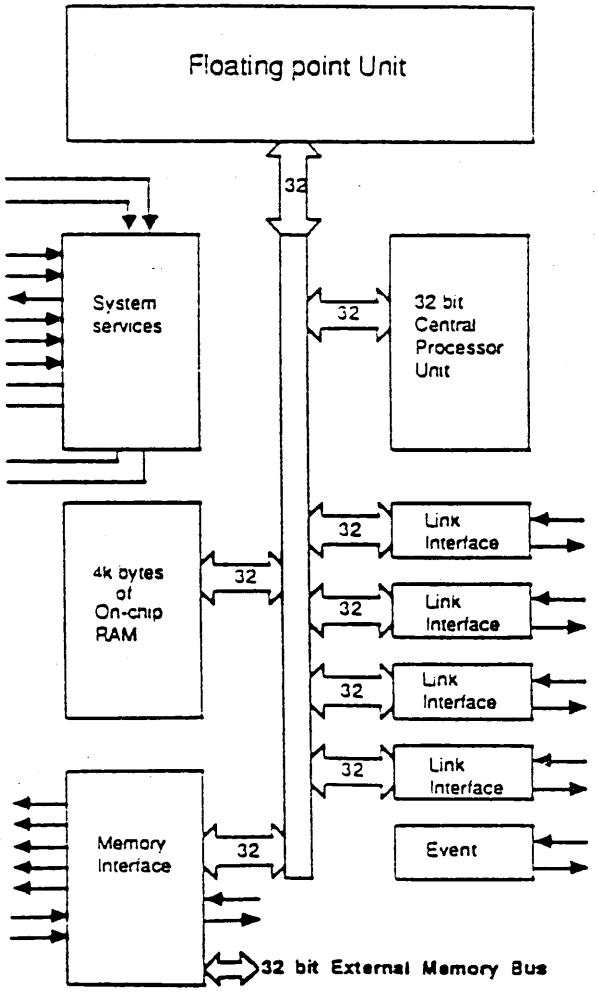


Figure 5.1: Block Diagram of IMS T800 Transputer

- Eight 20 Mbits/S on chip serial communication links (DMA engines), four IN and four OUT
- Communication being autonomous from the processor, which only issues authorisation, that takes 1 μs per message, regardless of the length of the message.
- 4 kbytes of static on chip RAM to allow high speed I/O and computation, matched to the speed of the processor and can be accessed and cycled in 50 nS

- Minimal Instruction Set, few registers, and can operate at a peak 20 MIP rate, although the overall throughput is quoted as 10 MIP, as some instructions take longer than a single cycle
- Extra external memory of up to 4 Gbytes can be fitted
- The hardware scheduler of the Transputer allows, even a single transputer, handling multiple jobs at the same time

To utilise these features of the Transputer, Occam has been designed, as a software support that is capable of expressing systems level requirements of concurrency and communication.

Any number of parallel processes can be specified in an occam program, and these can run on one or more transputers. Multiple processes are supported by means of task switching mechanism provided by the instruction set of the transputer.

Communication between processes is via point to point links known as channels. Occam channels between parallel processes in the same machine are implemented by locations in memory.

Communication provided by the occam model of concurrency is synchronised and unbuffered [36]. This avoids the overheads of organising message queues, data buffer and so on , while giving advantages in terms of simplicity of programming and avoidance of many of the simple causes of deadlocks.

The occam process, is a procedural unit, defined by the **PROC** statement. Channel declaration **CHAN**, defines the interface through which the process communicates with the rest of the program.

Processes are built from 3 primitive processes (assignment, input, output SKIP, and STOP). SKIP is a process that does nothing and terminates, used for completing the syntax of constructs. STOP is a process that does nothing, but does not terminate. These primitive processes are combined to form constructs:

SEQ sequential, **PAR** parallel, **IF** conditional,

WHILE iteration, **ALT**, alternative.

A construct itself is a process and may be used as a component of another construct, suitable for hierarchical decomposition of complex tasks. This is unlike conventional sequential programs, expressed with variables, and assignments combined to form sequential, conditional and iterative constructs.

Real-Time programming in occam is possible, by using Timers. A channel like type **TIMER** which is only capable of input, is used to declare named objects which can input the current time represented as a value of type **INT**. The operator **AFTER** can be used followed by an expression representing time, to introduce delays.

PRI can be used in real-time programming, to signify priority among processes. Prioritized PAR, has the effect of assigning a priority to the parallel processes, the level of which is determined by the textual order of the processes. In prioritized ALT, when two inputs become ready simultaneously, the process with the higher priority will be executed.

A powerful feature of occam is that it allows the use of a device called the replicator to be used with one of the constructs, for construction of arrays of processes, in addition to data and channel arrays.

A process can communicate with external devices which are connected to the processor's memory system [64], using memory mapped interfaces, provided by occam. A port specification which is similar to a channel specification, is used. A port input, inputs a value from the port, assigns it to the variable and then terminates. A port output evaluates the expression and outputs the result to the port. Occam also provides bitwise operators, to allow low level operations on the individual bits in a value.

5.4 The Parallax System

The Parallax system, consists of four JDOO2s (each providing two T800 Transputer based systems) and two single T800 transputer based GBUS96 systems

(providing G64 and G96 bus interfaces to application boards). A 12bit resolution ADC and a 12bit DAC board were connected to the GBUS96s.

5.4.1 Hardware

The ADC and DAC boards are interfaced to the network of transputers via G64 busses provided by GBUS96 IO Motherboards.

GBUS96 (Transputer based I/O Motherboard)

On this board as well as a T800 transputer chip, there is half a megabyte of configuration EPROM and half a megabyte of RAM .Therefore it could be used as a stand alone system or be part of a network [31]. The edge connector allows access to four transputer links, as well as the up, down ports which enable external control of transputer Reset and Analyse signals and the system port , which in a parallax system containing a special system management card can be used to control and monitor the transputer systems environment.

The board provides an interface to application boards such as ADC or DAC via a G64/G96 bus. It supports asynchronous and synchronous modes for data transfer, interrupt and bus request handling in bus master mode, as well as direct memory access requester mode.

The G64/G96 interface is mapped into the upper half of the transputer memory addressing space. Five ports are available for synchronous(2) and asynchronous(2) addressing of cards as well as managing the I/O bus control signals(1). It should be noted that the port addressing space for synchronous and asynchronous ports overlap and application cards must be placed at different addresses to avoid conflict.

Address	Read	Write
BASE	Data most sig. byte	Init. 12-bit conv.
BASE + 1	Data least sig. byte	Init. 8-bit conv.
BASE + 2	STS word	Clears EOC latch
BASE + 3	(unused)	Multx. chan. latch

Table 5.1: Register Accessed

bit	in OCCAM	function
7	STS AND with # 80	converter STS line (high during conv.)
6	STS AND with # 40	latched end of conv. (active high)
0 - 4	STS AND with # 1F	multx. chan. address latch cont.

Table 5.2: 8 bit Status Word

SYN-DAC8 Digital to analogue converter (By Syntel microsystems)

This is a G64 bus compatible analogue output module with eight 12bit resolution channels [89]. A set of dip switches are used for selecting the board base address within the system Valid Peripheral Address (VPA).The base address set is relative to the system VPA base address.

DAC8 occupies 16 bytes of VPA space (two bytes per channel).The base address offset is set to zero. Updating the DACs are done by writing the correctly formated data to the appropriate address. The data format was chosen to be left justified and the jumpers were connected so that the output range was between -10V and +10V.

SYNC-ADC4 A/D converter module

This is a 32(single ended) or 16(differential) channel, 12bit resolution (left justified) G64 bus compatible board which occupies 4 bytes of system VPA space [88]. Voltage range of -10V to +10V was chosen by connecting the appropriate links on the board. The base address is selected using a number of dip switches. This address was set to 800 to be different from the address of DAC8. Accessing the registers for the board are summerised in table 5.1 The status word is made up

of 8 bits and the function of single bits are described in table 5.2.

5.4.2 Driver software

Transputer Development Systems (TDS), was used to develop the Occam driver software. This is a complete interface for Occam, combining a folding editor with configuration and compilation utilities. SYN-ADC4 and SYN-DAC8 are both addressed synchronously. Thus the synchronous port of the I/O motherboard is used to address them. Control. port is defined as PORT OF INT in the occam program and is placed at #38000000 which is the OCCAM MAP word offset. The diagram of the memory map, which specifies the machine map byte address and the occam map word offset can be seen in Table 5.3.

To access IO memory space in synchronous mode Synnc.IO.Space are declared as memory INT arrays (INT = 4 bytes), covering all the length of G64 bus addressing space (16 data bit bus with 2×64 kbyte addressing space). Sync.IO.Space is placed at #34000000 (look at the memory map).

Three jumpers LK8, LK9 and LK10 were configured on the motherboard to select the operation of the board as G64 bus-master mode as opposed to Direct Memory Access(I/O device).

Addressing the ADC and DAC and arbitrating between bus request and/or interrupts are the responsibility of bus management port. This port consists of a single register to which a control word can be written and the status read back. This is done during initialisation, where configuration data is written to the control register to define the operating mode of the card. Once this is done the G64 bus must be enabled by deasserting the G64 Reset.

To drive the converter boards bus master board, Procedure Init.Interface (BOOL Bus.Master.Mode, PORT OF INT Ctrl.Port) is used with bus.master.mode set to TRUE . This initialises the interface such that the I/O motherboard is operating in bus-master mode and the bus activity is enabled.

The declarations of control and status values required to drive the I/O motherboard bus interface via the control port and declarations of maximum page length is included in a library. Example of these declarations are:

```
PORT OF INT Control.Port :
PLACE Control.Port AT # 38000000 :
[ G96.max.page.length ] INT Sync.IO.Space :
PLACE Sync.IO.Space AT # 34000000 :
```

The procedures for the interface are also in a library These include Initialising the Interface, Bus Master Event handeler, Direct Memory Access on and off.
The initialisation of the interface PROC looks like this

```
PROC  Init.Interface ( BOOL Bus.Master.Mode,
                       PORT OF INT Ctrl.Port )
  VAL INT Set.BM                IS #0A000000
  VAL INT Clr.BM                IS #0A010000
  VAL INT Enable.Interface      IS #0A060000
  VAL INT G64.max.page.length IS #10000
  SEQ
    IF
      Bus.Master
        Ctrl.Port ! Set.BM
      TRUE
        Ctrl.Port ! CLr.BM
    Ctrl.Port ! Enable.Interface
:
```

The library logical names are included in the Toolkit fold for when TDS is used. For ADC, a 12bit conversion is initiated and when bit 7 of the status word is low, 8 relevant bits (data most sig. byte) are read to base0, by masking off the other

Machine Map. Byte Address	Hi Lo	Occam Map. Word Offsets
# 7FFF FFFC		# 3FFF FFFF
	EPROM Code & Config data	
# 7000 0000		# 3C00 0000
	Control Word	
# 6000 0000		# 3800 0000
	Sync. I/O	
# 5000 0000		# 3400 0000
	Async. I/O	
# 4000 0000		# 3000 0000
	Sync Mem P1	
# 3000 0000		# 2C00 0000
	Sync Mem P0	
# 2000 0000		# 2800 0000
	Async Mem P1	
# 1000 0000		# 2400 0000
	Async Mem P0	
# 0000 0000		# 2000 0000

Table 5.3: MEMORY MAP

bits of the BASE. Then the remaining 4 bits (data least sig. byte) is read to basel from BASE + 1. note that data is left justified. Then these are combined by shifting the most sig. byte of the data to the left and the least sig. byte of the data to the right by 4 bits and using bitwise OR.

In the case of DAC, the digital values are written to BASE address and BASE +1 in order to produce the converted voltage on channel 0. Note that there are 2 bytes per channel and data is left justified.

A library was created which contains the procedures for driving individual links in a specified direction, as well as opening and closing of the gripper for the MA3000 Robot. This library also contains a procedure that sets all the output channels of the DAC to zero.

As an example, the waist or link 1 is driven using channel 2 of the DAC i.e.

Sync.IO.Space[804]

Sync.IO.Space[805]

and its direction is selected using channel 5.

The values for the direction correspond to the level of voltage required for the particular direction choice, as described earlier.

5.5 Meiko Computing Surface

The idea behind using this system, was to extend the computing power of the Parallax, by addition of more processors. Another motivation was, utilisation of the provision of the facilities which allow cross-development compilers, offered by Meiko Computing Surface. This meant that control programs and routines in pascal, did not have to be rewritten in OCCAM.

Using the electronically configurable link connections, was an added bonus.

The Meiko Computing Surface provides an architecture for building scalable concurrent computing systems [66]. The boards within the system, consist of different function blocks, Computing Elements, consist of a processor and some memory, Supervisor Bus Interface, provides a global communications route between the local Host and the processors within the system, monitoring and diagnostics, Link Network Interface, allows manual or electronic routing of user determined link connectivity, and Event Control Logic, describes the sources, masking and enabling of events to the processors.

The surface can comprise of a number of modules, configuration of which and connection between them, depending on the requirements of the application, is decided by the user. The modules should be connected together by a control interface and user links.

The processors in the system used are hosted by an MK050. In all, eight quad compute boards, MK060s which consist of 4 $\times$ T800 processors each, together with communications, DRAM, and an interface on to the supervisor bus, are included. In addition, there is a T800 based Frame Grabber (MK027) and a Graphics board (MK015), which were not used.

Meiko Multiple Virtual Computing Surface (MMVCS) operating system was used to organise domains of processors in a variety of sizes and shapes, as required and develop programs. It contains two other operating systems, MEiKOS, which is Meiko's UNIX-based operating system, and OPS, which is the Occam Programming System, similar to TDS, on Parallax.

Meiko Pascal Compiler and associated software tools can be used both within OPS and stand-alone compiler from MEiKOS. This allows parallel versions of pascal to run concurrently with Occam programs and communicate with them.

In this way, Occam, which is the native language of the Transputer, with control constructs such as (SEQ, PAR, ALT, IF, WHILE) closely modelling the control capabilities of the transputer hardware, while its communications primitives give effective access to the Transputer's inter-process and inter-processor communication links, can be used, and at the same time communicating pascal routines could also run concurrently with them.

The possibility of being able to employ Communicating Sequential Tools (CS Tools), which is a program development toolset for multiprocessor computer systems on Meiko is also appealing.

CS Tools supports the programming of multiprocessor applications using familiar development environments and standard languages. It consists of cross-development tools, compilers and configuration systems, as well as runtime facilities such as high level communication services and symbolic debuggers.

Parallel applications are expressed as multiple sequential code threads which co-operate by message passing. One of the utilities CS Build, used for loading the sequential threads on to application processors, operating at load time, enables a parallel application to be re-targeted to different parallel machine configurations with no modification to the application itself.

Communication with Parallax

To be able to communicate with the Prallax, from MCS, two hardware interfaces are needed, one link and the other control. The link interface can be implemented in a similar way to the inter-module link interface, which is described by a Meiko Technical Info. sheet.

Due to the inherent low tolerance of Inmos link protocol for data errors, a high quality connection technique, with good noise immunity, allowing for 20 Mbits/s link speed is needed and a moderate distance between the Meiko and the Parallax. The transputer links, use full CMOS output drivers and TTL threshold receivers, which are single ended, and when unbuffered, are unlikely to offer adequate performance.

The best noise rejection and noise margin is provided by a differential buffering technique. To provide this and also allow for high bandwidth, an ECL driver receiver pair using differential signaling is used, which according to data sheets and experiments by Meiko, fall within the specifications.

By using TTL-ECL, ECL-TTL translators, typical bandwidths of 85 MHz can be achieved, with typical combined skews of around 2.5 nS

Resistors are used to form a series termination network into a cable made up of twisted pairs, insulated and individually screened. This way the maximum output current of the translators are limited, under short circuit conditions. A distorted waveform at the transmit end of the cable, but a clean signal at the received end is generated.

Resistors are also used to provide pull down for the output of the driver, provide a small bias current to the output of the differential receiver, when the cable is disconnected, and to limit the potential current that flows between driver and receiver pairs during power up.

Control Interface to provide remote reset, analyse, software and hardware error connections and a link for supervisor type communication, are also needed. The

control signals are buffered on input and output and daisy chained between the Meiko and Parallax. They use RS423 signaling levels and active low signals. The rise and fall time of the drivers are set to approximately 100 μS to minimise crosstalk.

A driver software which allows addressing of the free Ports on the CS is included with the MMVCS (To Slave Port is used for control, and for links, Inter-Cabinet Link interface is used)

5.6 Electronic Interface Units

Two interface units were designed, one current driven for computed torque control, as in the type of motors used, torque is proportional to current $T \propto i$, and one voltage driven for position and velocity control, as velocity is proportional to voltage $\dot{\theta} \propto V$. Also the current profile under the voltage controlled scheme, was observed not to be smooth enough for one of the controller implementation, namely adaptive computed torque with load mass estimated on-line, hence the current controlled approach was employed.

5.6.1 Voltage Controlled Output, Driver Circuit

The motor drive amplifiers, as well as the feedback circuitry in the controller unit of the MA3000, were used in their original form and the rest of the hardware needed was built. A schematic representation of the interface board built is shown in Figure 5.2.

Analogue voltage signals from the DAC provide both direction and speed control of individual joint motors. The motors for base and shoulder are Bodine 32D3BEPM-W permanent magnet D-C motors, for elbow Bodine NPM-13 permanent magnet D-C motor and a Portescap 34L11-224E5 is used for pitch motion. For all the motors except the last, amplified Pulse Width Modulated (PWM) signals are used to drive them. As can be seen from Figure 5.2, a comparator is used to produce

a PWM signal from the analogue voltage from DAC together with a repetitive ramp waveform (sawtooth) signal obtained a tap point on the controller board of the MA3000.

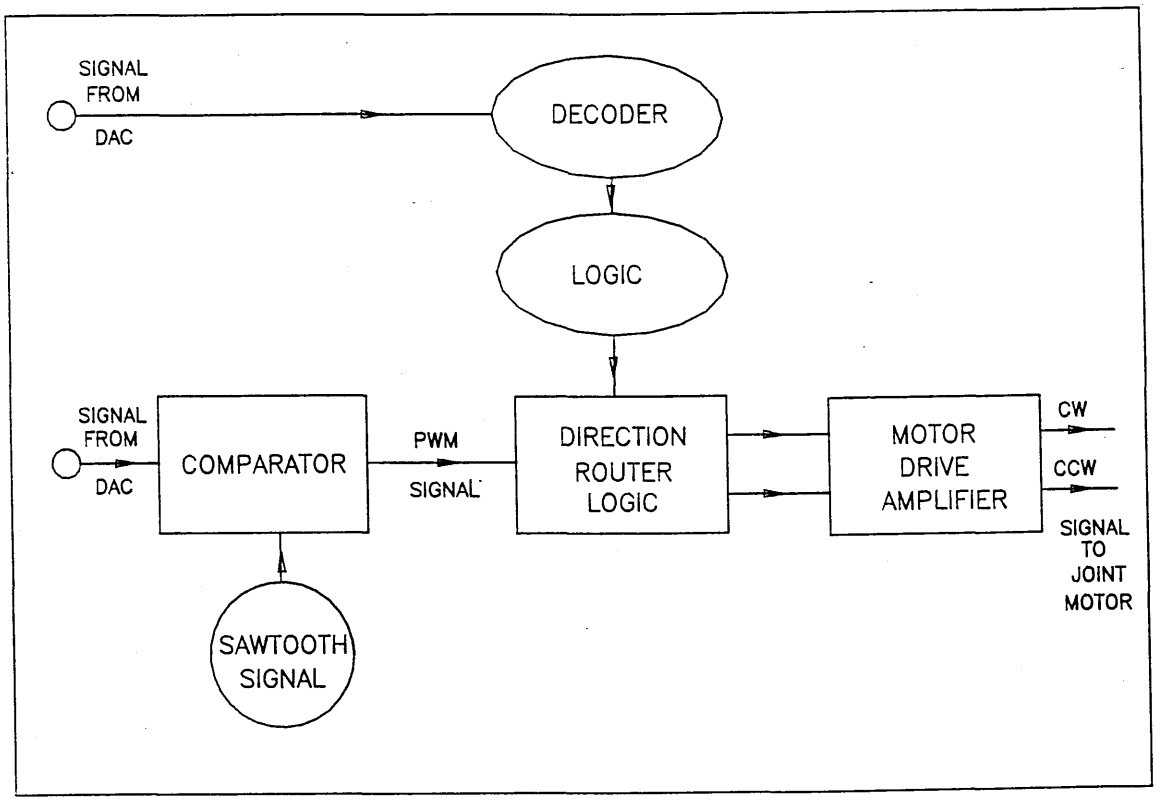


Figure 5.2: Schematic of the Interface Board

DAC voltages on different channels are decoded and passed through some initial logic gates before going to direction router logic with the PWM signal in order to establish the direction of movement. The resulting signals are sent to the motor joints via motor drive amplifiers for the first three motors.

The interface board provides arrangements for the interface of the fifth joint motor (ie. roll movement) as well as the gripper , but roll movement motor is not used at present. The circuit diagram of the interface board is shown in Figure 5.3. The assignment of DAC channels and the pin numbers corresponding to individual joints are shown in the Figure.

Since there are only 8 DAC channels, a decision had to be made as to either build

a circuit that allows the direction selection by providing positive or negative drive voltage corresponding to cw or ccw, or if direction signal had to be provided from a different DAC channel, one channel was capable of providing direction signals for two joints. These two options are represented in Figures 5.4 and 5.5. As

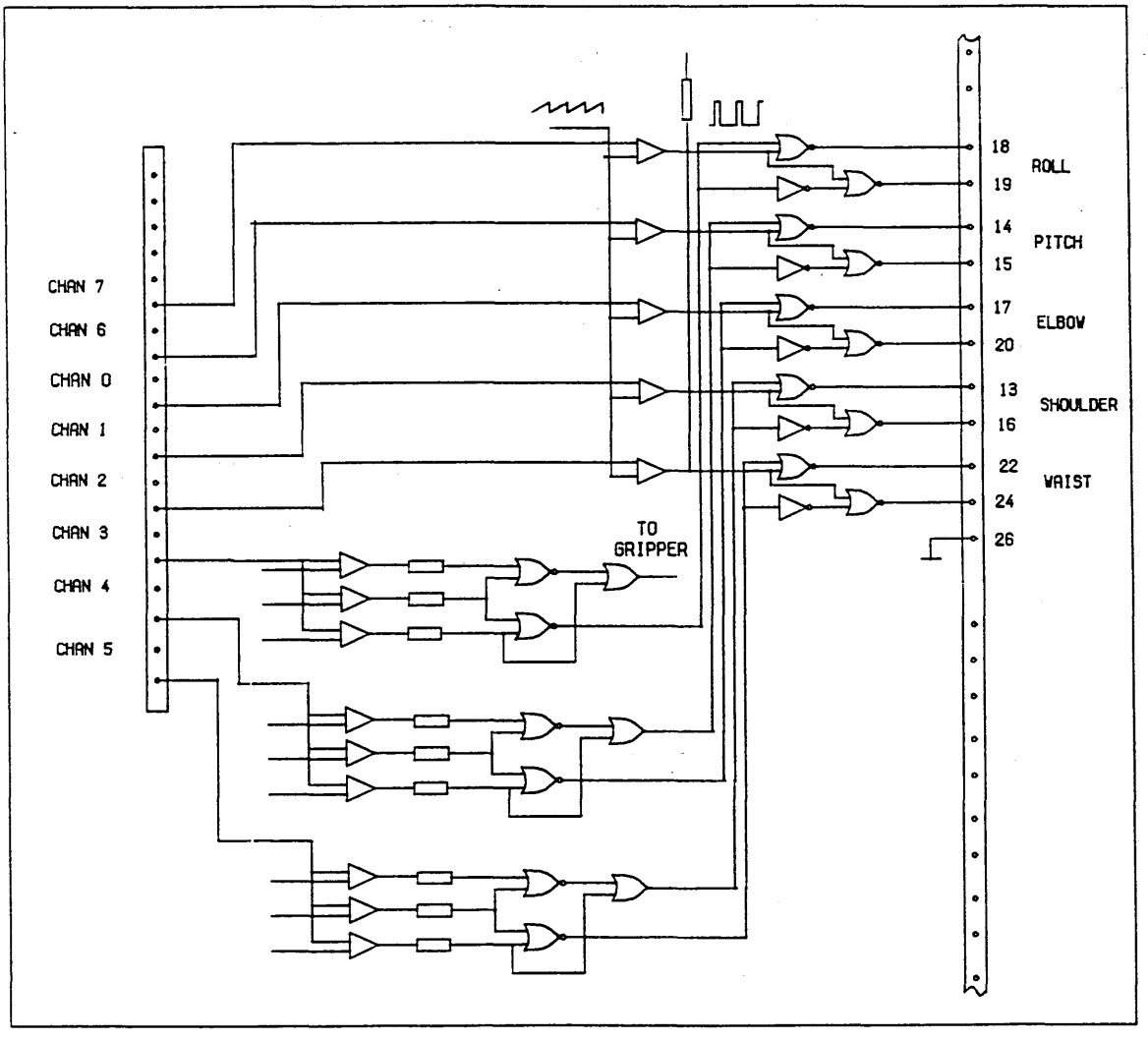
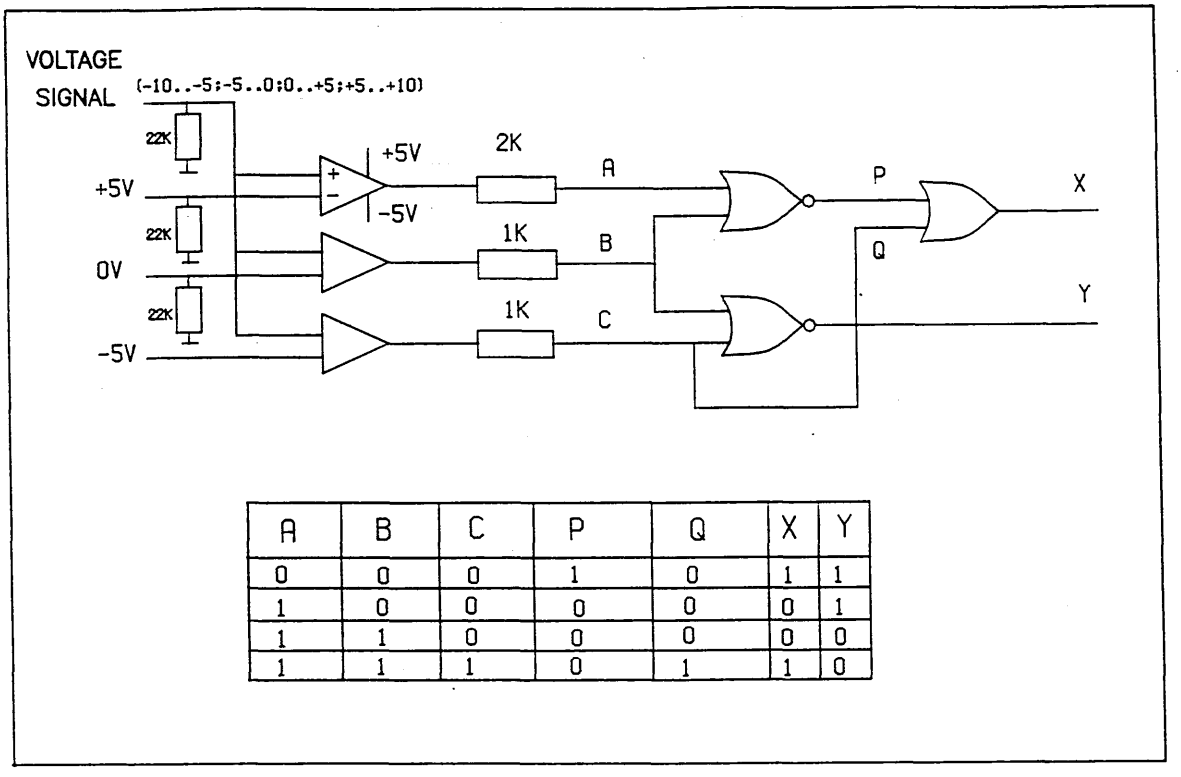


Figure 5.3: Interface Board Circuit Diagram

can be seen the first option involves having a DAC channel that provides four distinct voltage levels namely -10...-5, -5...0, 0...+5, +5...+10 in order to supply +5V or -5V for two of the joints, according to the truth table shown. In the second option, both direction and speed signals for a motor is provided from one DAC channel, by means of positioning two diodes with operational amplifiers, hence



A	B	C	P	Q	X	Y
0	0	0	1	0	1	1
1	0	0	0	0	0	1
1	1	0	0	0	0	0
1	1	1	0	1	1	0

Figure 5.4: First option for drive and direction signals using DAC chans

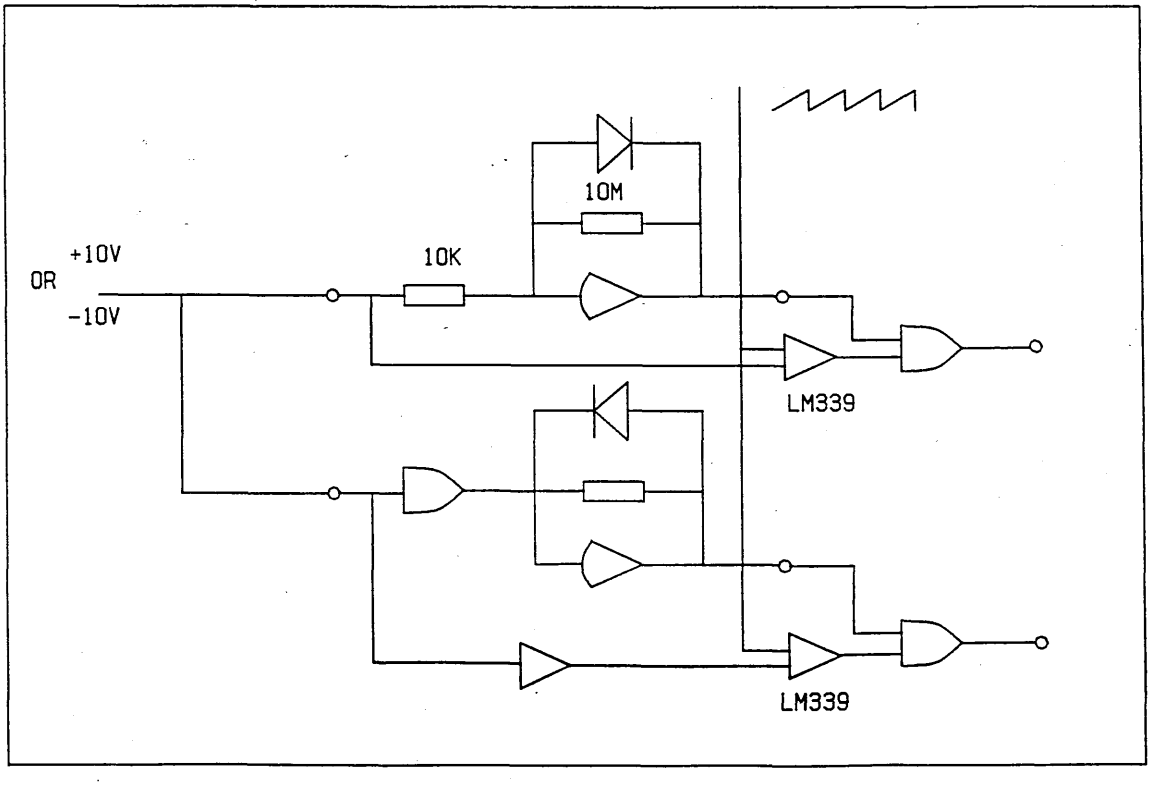


Figure 5.5: Second option for drive and direction signals using DAC chans

restricting the flow of current in either sense depending on whether the voltage signal is negative or positive.

Although the second option would have decreased the programming effort, but due to its complexity and extra hardware needed ,for example a large number of resistors around the opamps to prevent bias problems which could arise when voltages of around 0V are provided, option one was chosen. A power supply to give +5V and -5V was included on the interface board to give the required voltages of this option.

The Overall Circuit

It was decided to leave the circuit which provide the emergency stop signal and the signal which indicates whether any of the joints are out of their limits or not (info. by means of LEDs on the front panel of the controller).Also it was decided to leave the circuit for ROLL movement, so that it could be operated manually from the front panel. The overall connections could be seen in Figure 5.6. Voltage signal from DAC is provided through a 20 way socket-plug arrangement to the interface board. The eight channels available provide both direction and drive signals. In the case of direction these pass through LM348 opamps which will output either +5V or -5V for each motor depending on the level of the input voltage. These were grounded via 22K resistors to prevent floating. For the logic gates 7428LS single ended nor gates and 7432LS or gates were used.

A sawtooth signal is supplied from TP6 of the controller board. This signal together with the DAC drive voltage signals are input to LM339 single-ended comparators which output PWM signals and these go through direction router logic 7404 NOT gates with direction signals.

These are then connected to a 26 way socket-plug (not all of the 26 pins are connected(only the ones shown in Figure 5.3 are)) and are further connected via a ribbon cable to a junction, where another ribbon cable from the 26 way socket-plug of the controller board joins. From this junction all the wires of the first

ribbon cable and the wires from the second cable that correspond to the missing ones of the first are combined and sent to the 26 way socket of the power amplifier board. There is also a 20 way socket on the controller board which provides the

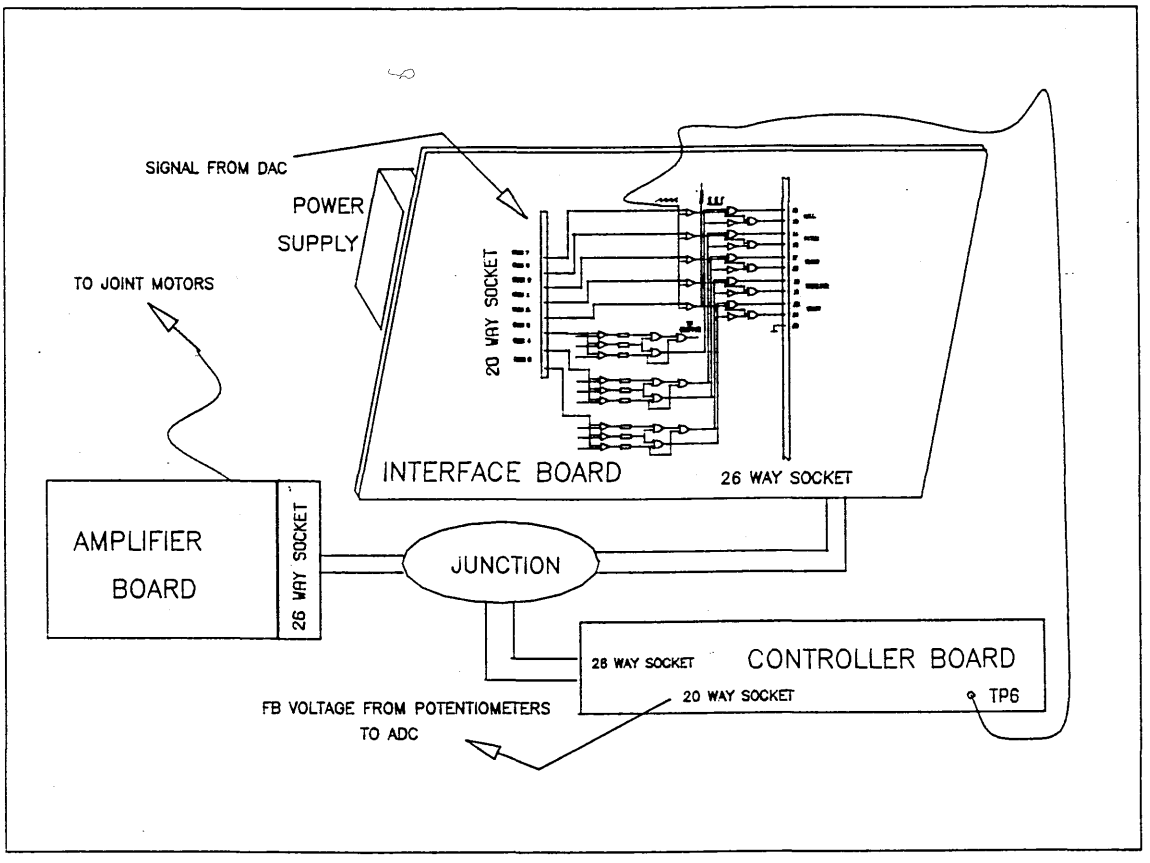


Figure 5.6: Overall connections of the Interface

Feed Back Voltage from potentiometers mounted on individual joints. This is connected to the ADC. The designation of ADC and DAC channels are shown in section 4.

5.6.2 Current Controlled Output, Driver Circuit

In this circuit, PWM is not used, as current control is not possible without, significant additional hardware, and due to rapid switching of the current through an inductor (the armature), electrical electrical noise is created.

In addition when direct control of torque is required, as is the case for model-based control algorithms of robots, then linear operation with an amplifier is preferable

to PWM.

To be able to control the amount of current that is supplied to individual motors, by varying the input voltage, n-channel enhancement type power MOSFETs are used.

In this type of MOSFET, when gate is made positive with respect to source, the field of the positive gate repels holes in the p-type substrate away from the insulating layer, leaving behind a narrow channel of n-type silicon. This narrow

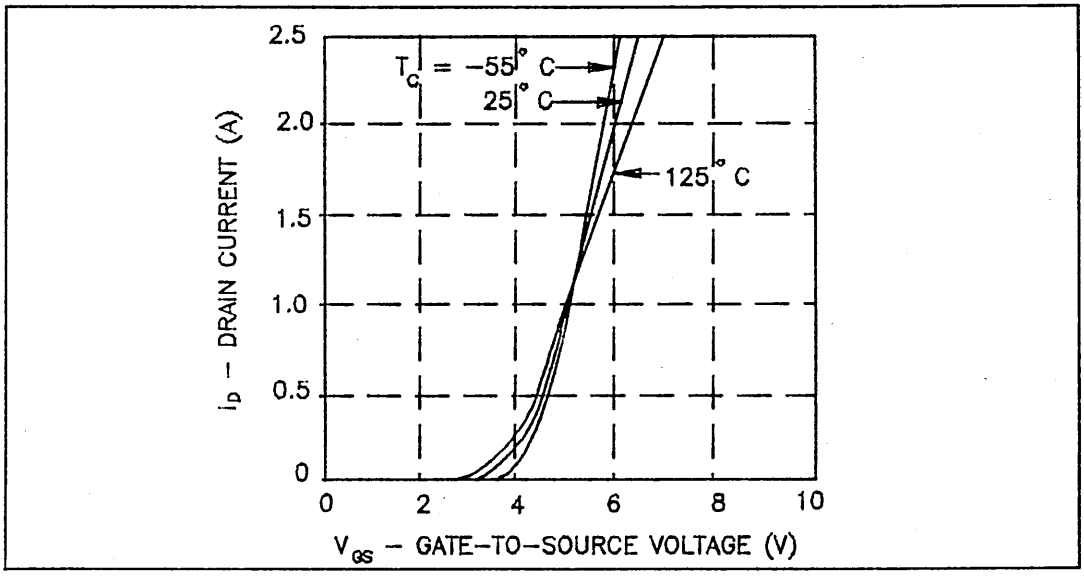


Figure 5.7: Typical Transfer Characteristic od IRF610

channel provides a conducting path from source to drain. In this way, given a certain positive voltage on the gate to make the device conduct, the drain current is under the control of the gate voltage.

There are advantages in using MOSFETs, as opposed to bipolar transistors. They operate at much higher speeds, and have a positive temperature coefficient, that means their resistance increases with temperature, so that the MOSFET is inherently stable in response to temperature changes and protected against thermal run-away.

They can also be operated in parallel with other MOSFETs without "current hogging", which means if any device overheats, its resistance increases, and the

current is rerouted to the cooler chip [33]. A typical transfer characteristic of the

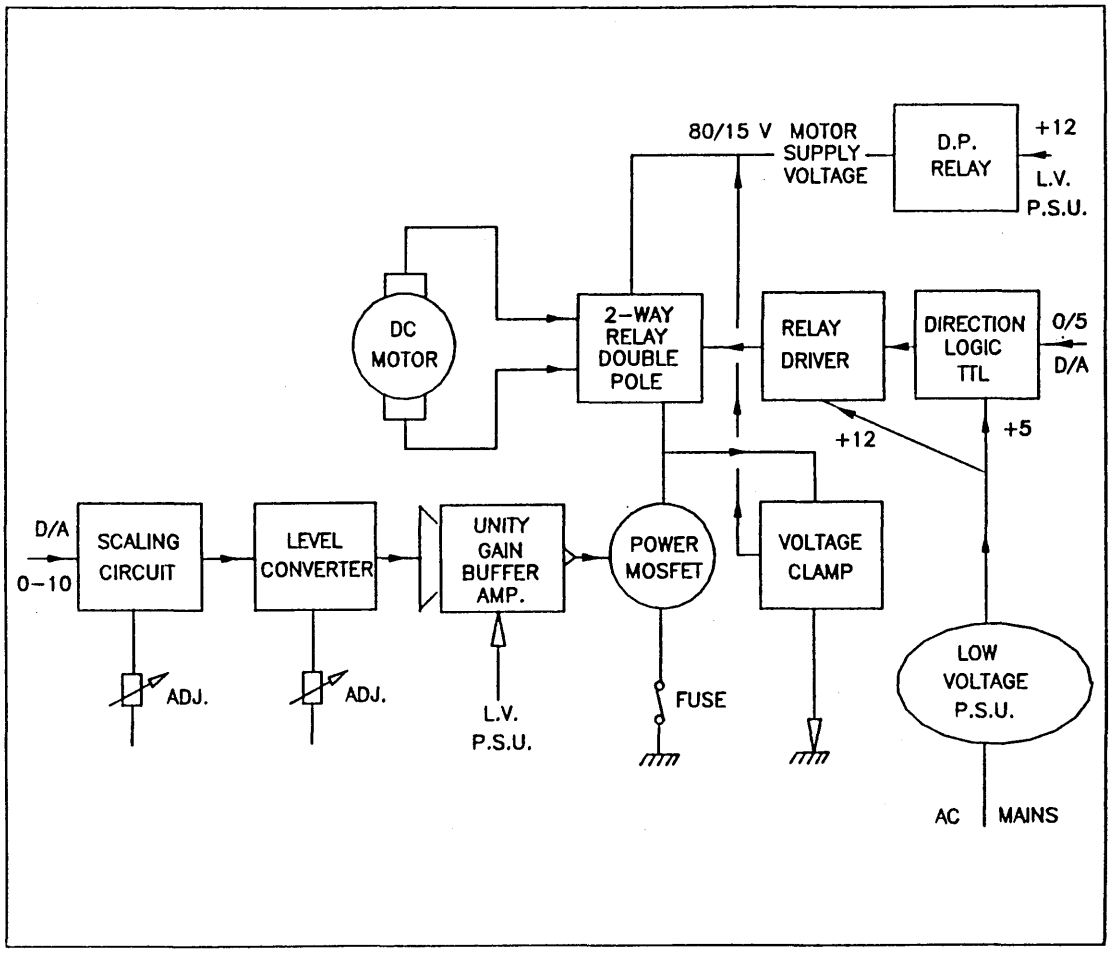


Figure 5.8: Description of current drive circuit

IRF610, which was used after considering various types of MOSFETs is shown in figure 5.7. As can be seen the linear region of the graph, spans for about two volts, which in practice due to high currents resulting at the top end of the scale, saturation occurs before reaching the maximum. Hence input variation of one volt can be used to linearly vary the current.

The circuit diagram description for implementing the current drive is shown in Figure 5.8. It shows that the control voltage which is output from DAC (between 0-10 V) is fed through a scaling circuit and a level converter, so that the voltage range is scaled down and added to the threshold voltage value specified by the

transfer characteristics of the MOSFET. This is then passed to the power MOSFET, after a buffer amplifier LM324.

DAC channel outputs (0-5) are also passed through direction logic (7405) and a relay driver to operate a relay for direction change of the motor when required. A relay is also used to prevent the output plug pins of the circuit which can carry

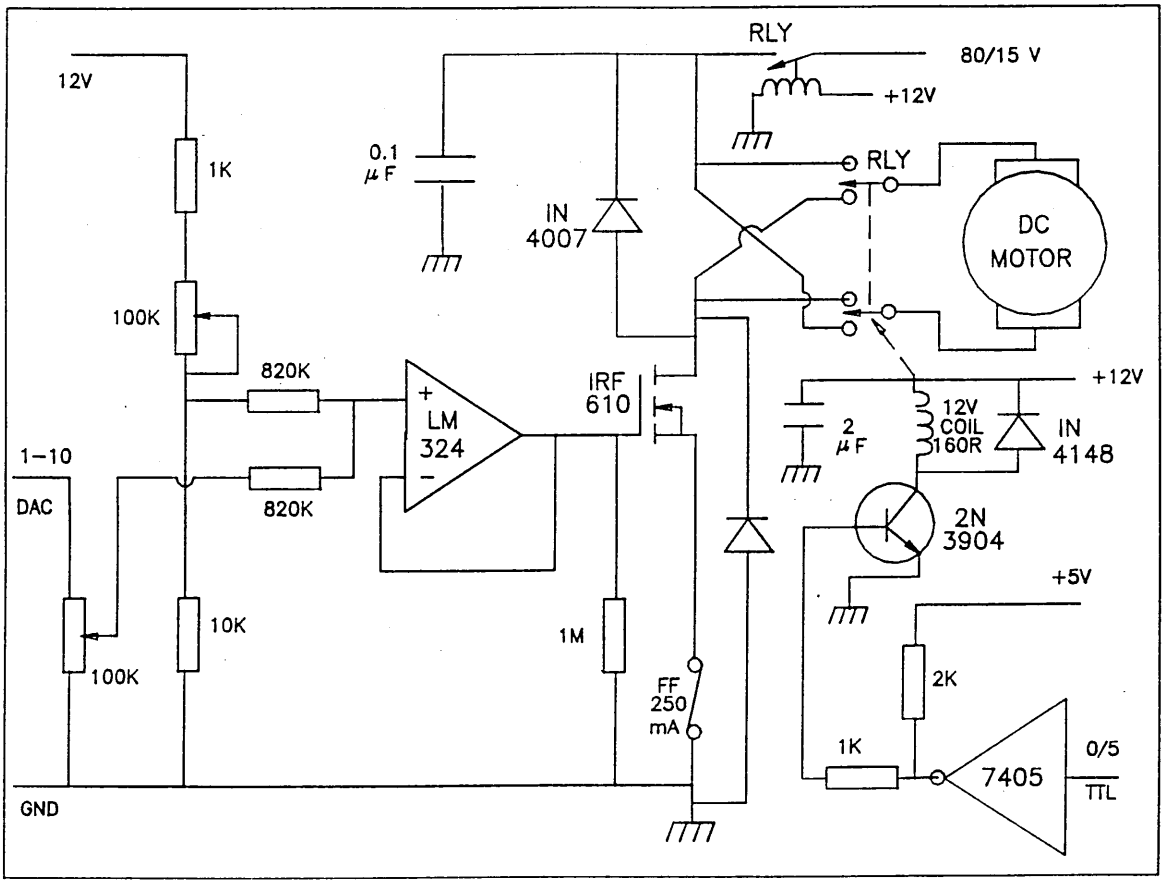


Figure 5.9: Current drive circuit diagram

up to 88.5 volts, to have any voltage present, unless the plug is coupled with the robot socket, as a safety measure.

The circuit diagram of the above is shown in Figure5.9 A current limiter is also added to the circuit, in order to prevent excessive currents to the motor. This is not shown in the diagram.

The overall representation of the system connections , including, the Meiko, Parallax Interface Unit, and the MA3000 is shown in Figure 5.10.

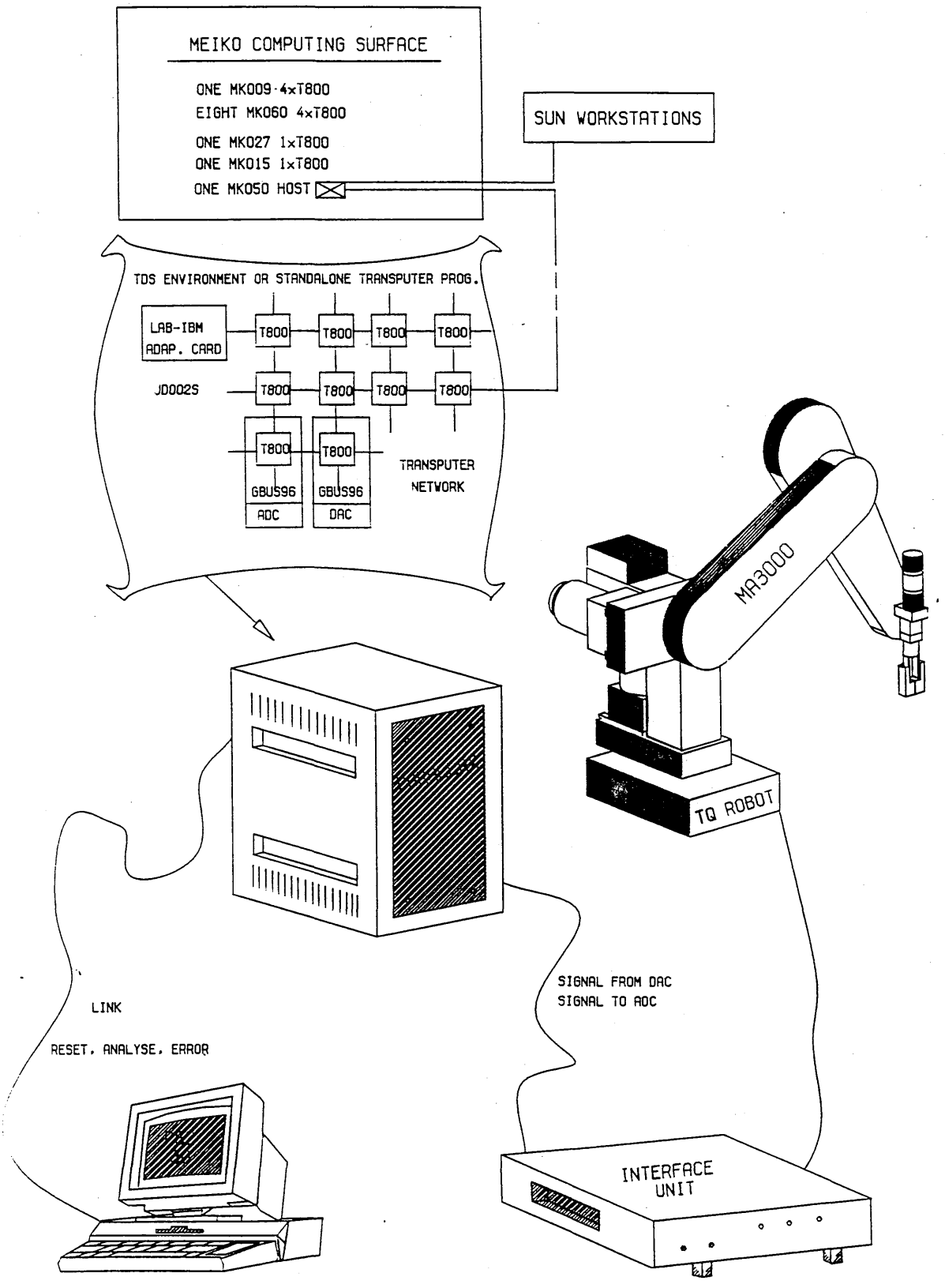


Figure 5.10: The overall connection of system components

Chapter 6

Control of Robot Manipulators

SUMMARY

After a brief introduction which points out the difficulties with PID controllers that a great majority of robot manufacturers employ for manipulator control, a review and discussion of major control schemes that have been proposed to improve the performance and capabilities of manipulators for mainly trajectory following tasks is presented. Two new adaptive schemes that utilise a priori knowledge of robot dynamics are introduced in sections 6.7 and 6.8. Implementation of these methods to control an MA3000 robot will be explained and discussed in the next chapter. One of the control shemes is model based control with load mass estimation, and the other is model based continuous time variable structure self tuning.

6.1 Introduction

The robot manipulation problem is basically defined as dynamic control under kinematic constraints. To control the manipulator to follow a desired path, one needs to servo the manipulator's joint actuators.

Prior to this, path planning ought to be carried out to ensure a particular type of

movement, obstacle avoidance etc.

Therefore robot manipulation is done in two stages. First the trajectory planning, where the desired path is usually specified by a class of polynomial functions which generate a sequence of time-based control set points. And secondly the motion control, where on the basis of the knowledge of kinematics and dynamics, a control strategy is employed to achieve a desired response and performance.

Kinematics and dynamics of manipulators were discussed in a previous chapter. As stated dynamic equations of robots can be represented by a set of highly coupled nonlinear second order differential equations.

Industrial robot controllers, mainly operate on joint basis and ignore the dynamic interaction of the joints, and as a result accurate trajectory following and high speed operations are not catered for. Simplicity and low price of implementation associated with the PID (Proportional, Integral, and Derivative) control method has led the majority of industrial robot manufacturers to employ this approach. Each joint of the robot is taken to be a second order system with inertia and frictional damping, and usually gravitational loading as well. By careful choice of the controller gains to suit system inertia and damping, a satisfactory transient response with little overshoot can be achieved but only if the inertias were constant. Configuration dependant inertias can vary largely during an operation cycle. In addition, although steady state errors due to gravitational loading can be reduced to acceptable values by gain adjustment, handling different load masses, results in inertia changes.

This means that one PID setting can result in transient responses which are over, under, or critically damped, depending on the position of the robot and the load mass that it carries. The usual design procedure is to predict the worst case of inertia that each joint actuator is likely to encounter and then choose the PID parameters that give a critically damped transient. As a result the system will be overdamped most of the time during an operation.

Depending on the kind of application that the robot is meant to be used for, this

type of control strategy is quite frequently appropriate and the results are acceptable. For example for low speed point to point operations it gives good accuracy. For trajectory tracking, the accumulation of errors resulted from individual joints can be quite significant at times, and at high speeds the centrifugal and coriolis forces which create mechanical coupling between the joints can be large enough, to require some kind of dynamic compensation for good performance.

There have been a number of attempts to calculate the robot dynamics on-line and use the information within a control scheme, which will be discussed. The general view point in this chapter will be that in addition to this, there is a need for an adaptive control scheme to allow for parametric and structural uncertainties that are invariably present, as well as neglected dynamics in the modelling process and existence of variety of disturbances.

6.1.1 Robot Control Strategies

For a six degree of freedom robot arm, there are two type of movements, one that involves the first three joints and presents the first phase of the control objective to move the arm from initial position and orientation to the vicinity of the desired final position and orientation, and the second is the fine adjustments to get to the desired location as close as possible, which requires control of the end effector degrees of freedom.

In the former motion also referred to as *gross motion* the dynamic coupling between the joints and the configuration dependent inertias are significant, and majority of the control strategies in the literature address the control strategies for this type of motion.

In addition to path trajectory tracking, there are various other control schemes that are presented within the context of manipulator control, for example impedance and force control, coordinated control of multi robot systems, or incorporation of control schemes within learning algorithms.

The path trajectory tracking can be joint motion control, cartesian space control where the motion of various joint motors are combined and resolved into separately controllable hand motions along a specified coordinate system by means of simultaneously running the joint motors at different time-varying rates.

Adaptive controllers are in general employed to deal with inaccurate modelling and uncertainties.

They are designed to achieve objectives such as insensitivity to parameter uncertainties and unknown payload variations and to obtain decoupled joint response.

There are various ways of classifying adaptive controllers. It could be on the basis of the control objective which results in a particular controller structure, or whether the update of the parameters is done according to the error between the estimated parameters and the true parameters (prediction error), or the difference between the actual and the desired output (tracking error).

Another approach can be based on whether there is any approximation involved or not. The non-approximation methods generally consider the nonlinear dynamics of the manipulators fully, which are time varying and reflect their coupled nature. The main bulk of recent work in this area utilise the idea of being able to select a set of parameters that the manipulator dynamics depend linearly upon and as a result the restrictive assumptions such as slow variation of inertia matrix or decoupled motion of the joints can be lifted. This certainly also gets round having to linearly approximate the dynamics around a desired trajectory too.

Finally the type of measurements required can form the basis for classifying adaptive controllers.

In the following sections first of all nonlinear feedback control algorithms for manipulators, and some methods that have been developed to take advantage of the possibility of linear parameterisation of manipulator dynamics are discussed.

As classification of the form stated above might confuse the issue, some main schemes are discussed on the basis of the authors that presented them. They

basically cover most of the variations. In all cases global convergence of the algorithms have been shown.

Then linear perturbation adaptive control, model reference and self-tuning adaptive controls, robust control and in particular variable structure control applied to robot manipulators and an adaptive controller based on the idea of estimating the load mass that the robot carries are discussed. Finally a new method namely a model-based variable structure self tuning adaptive control is presented which decouples the dynamics and accounts for nonlinearities before applying a variable structure self tuning controller to each joint of the manipulator. This is usually implemented by first designing a non-linear feedback to transform the robot model into an equivalent linear system, and then use a linear controller for achieving desired closed loop response of the joints.

Although the above does not include all the approaches in adaptive control of robot manipulators, it represents a reasonable cross section which other methods are slight variation of. In the last section, miscellaneous methods which are of importance will be mentioned.

Where necessary issues of convergence and robustness will be included.

6.2 Some non-linear feedback control algorithms for Robots

In this class of controllers, the dynamic robot model is utilised to control the robot joints.

There are three main points to note, firstly the exact knowledge of the robot dynamic model is needed, second, the calculations of the equations need to be performed on line in real-time when implementing the scheme, so speed of computation ought to be considered, and finally the scheme is not robust in the presence

of modelling and parameter errors. In this section the general methodology behind this model-based approach will be given and then the work which has been done will be looked at.

The dynamics of the robot manipulators was discussed in a previous chapter. The structured closed form dynamic robot model for rotary joints can be represented as

$$M(\theta)\ddot{\theta} + Q(\theta, \dot{\theta}) = \tau \tag{6.1}$$

where θ represents the joint angles, M the inertia matrix, and Q the vector of centrifugal and coriolis forces.

This provides physical insight into the nonlinear system. It should be noted that the inertia matrix can be shown to be positive definite over the entire work space as well as bounded from above since it contains only polynomials involving transcendental functions of θ.

The equation can be brought to the state-space form by letting

$$x \triangleq (\theta, \dot{\theta})^T \quad ; u \triangleq \tau$$

to get

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -M^{-1}(x_1)[Q(x_1, x_2) - u] \end{aligned} \tag{6.2}$$

Actuator (motor) dynamics should be included in the dynamic equations of the manipulator to give a realistic model of the robot behaviour, the way which was stated earlier, or similar to the method of Tourassis [91], in which the dynamic model of the actuator of joint i is represented by a second-order differential equation

$$J_{Ai}\ddot{\theta}_i + F_{Ai}\dot{\theta}_i + \frac{\tau_i(t)}{N_{Ai}^2} = K_{Ai}U_i(t), \quad \text{for } i = 1, 2, \cdots, N \tag{6.3}$$

where J_{Ai} is the motor inertia, F_{Ai} the motor damping coefficient, N_{Ai} the gear ratio, K_{Ai} the gain, $\tau_i(t)$ the the actuating joint torque, and $U_i(t)$ is the input voltage.

Also

$$F_{Ai} = B + \frac{K_t K_B}{R} \quad \text{and} \quad K_{Ai} = \frac{K_T}{N_{Ai} R}$$

B being the motor damping constant, K_T, the torque constant, R the armature resistance, and K_B, the back e.m.f constant.

Then the constant actuator inertias scaled by the squares of the gear ratio are superimposed on the diagonal elements of the inertia matrix, and the reflected motor torques are directly added to the Q vector.

Sometimes the dynamic equations are represented in the form

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau \tag{6.4}$$

where compared to equation 6.1 G,the gravity dependent terms are represented separately.

The adaptive control implementation of Craig et al. [16] etc. suggest replacing M, C, and G by their estimates. In other words, based on the representation of equation 6.1 the actuating torque signals required to control the robot model are calculated from

$$\tau(t) = \hat{M}(\theta)u(t) + \hat{Q}(\theta, \dot{\theta}) \tag{6.5}$$

where circumflex represents the estimated values, and $u(t)$ is the control signal, which is designed to be the superposition of a nominal feed-forward signal $r(t)$ and a linear state-variable feedback control signal.

Equating the torques in 6.1 and 6.5:

$$M(\theta)\ddot{\theta} + Q(\theta, \dot{\theta}) = \hat{M}(\theta)u(t) + \hat{Q}(\theta, \dot{\theta}) \tag{6.6}$$

and adding $\hat{M}\ddot{\theta}$ to both sides and then rearranging, the following equation is obtained:

$$\ddot{\theta} = u(t) + [\hat{M}(\theta)]^{-1}\{[\hat{M}(\theta) - M(\theta)]\ddot{\theta} + [\hat{Q}(\theta, \dot{\theta}) - Q(\theta, \dot{\theta}]\} \tag{6.7}$$

This can be written in a simple form:

$$\ddot{\theta} = u(t) + s(t) \tag{6.8}$$

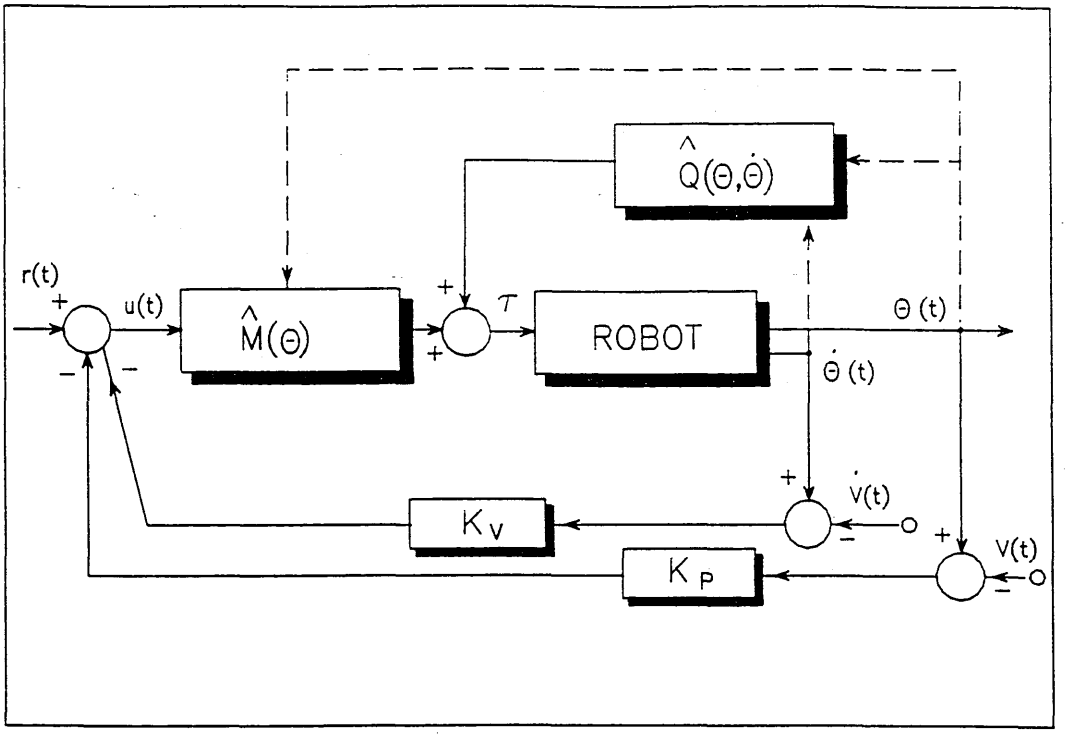


Figure 6.1: Nonlinear feedback

The block diagram representation of this strategy is shown in figure 6.1.

The control signal is equal to

$$u(t) = r(t) - K_v(\dot{\theta} - \dot{v}) - K_p(\theta - v)$$

where $v(t)$ represents the measurement noise vector, and K_v and K_p are the velocity and position feedback gain matrices respectively.

Substituting $u(t)$ into equation 6.8, we get

$$\ddot{\theta} + K_v\dot{(\theta)} + K_p\theta = r(t) + s(t)\overset{+}{\overset{\vee}{K}}_v\dot{v} + K_pv$$

This is the closed loop system including modeling errors and measurement noise. The dynamic equations of the manipulator, in addition to being used in control schemes that will be discussed, can also used to study asymptotic tracking of desired joint positions. For example it can be shown that a PD controller of the form

$$u = K_p\tilde{\theta} - K_D\dot{\theta} \tag{6.9}$$

where $\tilde{\theta} = \theta_d - \theta$ and K_p and K_v are diagonal matrices of positive proportional and derivative gains, achieves zero steady state error [87] in the absence of gravity. As described in [87], considering the Lyapunov function candidate

$$V = \frac{1}{2}\dot{\theta}^T(M(\theta))\dot{\theta} + \frac{1}{2}\tilde{\theta}^T K_p \tilde{\theta} \tag{6.10}$$

where V is positive except at $\theta = \theta_d$, $\dot{\theta} = 0$ where it is zero.

It can be shown that along any motion of the robot V decreases to zero. Since θ_d is constant

$$\dot{V} = \dot{\theta}^T(M(\theta))\ddot{\theta} + \frac{1}{2}\dot{\theta}^T\dot{M}(\theta)\dot{\theta} - \dot{\theta}^T K_p \tilde{\theta} \tag{6.11}$$

Solving for $M\ddot{\theta}$ in the dynamic equation of the form

$$\tau(t) = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + B\dot{\theta} + G(\theta) \tag{6.12}$$

where B represents damping coefficients, with gravity terms equal to zero and substituting the results in the above equation and using the skew symmetric property of the dynamic equations $\dot{M} - 2C$, with the PD control law substituted it can be shown that

$$\dot{V} = -\dot{\theta}^T[K_D + B]\dot{\theta} \leq 0 \tag{6.13}$$

as long as $\dot{\theta}$ is not zero, V is decreasing.

The possibility of the angles not being equal to their desired values when the angular velocity is zero is ruled out by assuming $\dot{V} \equiv 0$, then equation 6.13 implies that $\dot{\theta} \equiv 0$ and hence $\ddot{\theta} \equiv 0$. From equations of motion with PD control

$$M\ddot{\theta} + Q(\theta, \dot{\theta} = -K_P\tilde{\theta} - K_D\dot{\theta}$$

$-K_p\tilde{\theta}$ must equal zero, so $\tilde{\theta} = 0$, $\dot{\theta} = 0$. Using LaSalle's Theorem that states: Given the nonlinear system $\dot{x} = f(x)$ on $\Re^n$, supposing a Lyapunov function candidate V is found such that, along solution trajectories $\dot{V} \leq 0$ then the system is asymptotically stable if the only solution of the system satisfying $\dot{V} \equiv 0$ is the null solution, the system must be asymptotically stable.

This study further shows that in the presence of the gravity term in the dynamic

equations, the PD controller alone cannot guarantee asymptotic tracking and modifications to cancel the effect of the gravity need to be included.
For the purpose of control, the methods which are discussed in the next section utilise some properties of the equation of motion of manipulators.
Three main properties are

- Symmetric, positive definitness of the inertia matrix and for revolute joints the boundedness of it and its inverse.
- Ability to write the dynamic equations in a form that is linear in some parameters of interest such as masses, moments of inertia etc.
- If we represent the dynamic equations in the form 6.4 then $\dot{M}(\theta) - 2C(\theta, \dot{\theta})$ is skew symmetric

Using these, global asymptotic stability can be established. In cases where filtered acceleration is used in methods the third property is not needed.

6.2.1 Method of Goodwin and Middleton

This method is described in [67]. Presenting the dynamic equations of manipulators in the form of equation 6.4, linear parameterisation is used to obtain the model in the form

$$\tau = Y_1(\theta, \dot{\theta}, \ddot{\theta})a \tag{6.14}$$

where Y_1 is a non-linear matrix function of the angles, and their first and second derivatives. and a is a vector of equivalent parameters
Then the acceleration terms are eliminated from the above equation by filtering both sides of the equation through an exponentially stable and strictly proper filter shaped to effectively filter out high and low frequency disturbances.
Convolving both side of the equation by $w(t)$, the impulse response of the filter, and eliminating $\ddot{\theta}$ by integrating by parts the following equation is obtained

$$y(t) = W(\theta, \dot{\theta})a \tag{6.15}$$

where y is the filtered torque and W the filtered non-linear matrix function Y_1. Then a prediction of the filtered torque and a prediction error can be generated based on the estimated parameters $\hat{a}$ from the adaptation law

$$\hat{y}(t) = W(\theta, \dot{\theta})\hat{a}(t)e = \hat{y} - y = W\tilde{a} \tag{6.16}$$

with $\tilde{a} = \hat{a} - a$ being the estimation error.

6.2.2 Method of Slotine and Li

The method is introduced in [83]. Again representing the dynamic equations of the manipulator in the form 6.4, the estimated parameters $\hat{a}(t)$ are substituted in $M(\theta)$, $C(\theta, \dot{\theta})$ and $G(\theta)$ to obtain $\hat{M}$, $\hat{C}$ and $\hat{G}$. The control law is written as

$$\tau = \hat{M}(\theta)\ddot{\theta}_r + \hat{C}(\theta, \dot{\theta}, \dot{\theta}_r) + \hat{G}(\theta) - K_D s \tag{6.17}$$

This equation represents an adaptive feedforward action to adaptively cancel the robot dynamic forces, and a PD action regulating the tracking error to zero.

And the adaptation law is

$$\dot{\hat{a}}(t) = -P_0 Y^T(\theta, \dot{\theta}, \dot{\theta}_r, \ddot{\theta}_r)s \tag{6.18}$$

where K_D that can be time-varying is a uniformly positive definite matrix, the adaptation gain P_0 is a constant symmetric positive definite matrix, and

$$\dot{\theta}_r = \dot{\theta}_d - \Lambda\tilde{\theta}$$

which is introduced to guarantee convergence of the tracking errors, rather than just the velocity errors.

$$s = \dot{\theta} - \dot{\theta}_r = \dot{\tilde{\theta}} + \Lambda\tilde{\theta} \tag{6.19}$$

where $\tilde{\theta} = \theta(t) - \theta_d(t)$, and Λ is a constant positive definite matrix.
This is a measure of tracking accuracy and is used to drive the parameter adaptation.

Matrix Y of equation 6.18 is defined by

$$\tilde{M}(\theta)\ddot{\theta}_r + \tilde{C}(\theta,\dot{\theta})\dot{\theta}_r + \tilde{G}(\theta) = Y(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{a}$$

Y can be calculated from measurements of only θ and $\dot{\theta}$, as

$$\ddot{\theta}_r = \dot{\theta}_d - \Lambda\dot{\tilde{\theta}}$$

Global tracking convergence of the adaptive controller, is shown by first of all substituting the control law into the manipulator dynamics and obtain the closed-loop dynamics

$$M\dot{s} + (K_D + C)s = Y\tilde{a} \tag{6.20}$$

And considering the Lyapunov function candidate

$$V(t) = \frac{1}{2}[s^T M s + \tilde{a}^T P_0^{-1}\tilde{a}] \tag{6.21}$$

as well taking to account the skew-symmetry of $(\dot{M} - 2C)$, the following resulted

$$\dot{V}(t) = -s^T K_D s \leq 0 \tag{6.22}$$

As a result boundedness of s and $\tilde{a}$ and convergence to zero of both position error and velocity error is shown. However exponential convergence of the tracking errors in the presence of persistent excitation has not been shown.

6.2.3 Another method of Slotine and Li

An approach that uses both tracking error and prediction error for their adaptation law was presented in [84]. Using the same symbols as the previous method, in this method the adaptation law has the form

$$\dot{\hat{a}}(t) = -P(t)(Y^T s + W^T R(t)e) \tag{6.23}$$

W being the filtered value of a nonlinear matrix function of θ, $\dot{\theta}$, and $\ddot{\theta}$. $R(t)$ is a uniformly positive definite weighting matrix, and $P(t)$ is uniformly positive definite gain matrix.

The form of the control law and the closed loop dynamics are the same as the previous method.

In [85] they show the global asymptotic and exponential convergence of the tracking errors and parameter errors for the above adaptation law. The convergence analysis uses the Lyapunov function candidate

$$V(t) = \frac{1}{2}[s^T M s + \tilde{a}^T P^{-1} \tilde{a}] \tag{6.24}$$

They also show that the method has faster parameter convergence and better tracking performance compared to the previous method, by simulation.

P can be generated using the gain update technique of various parameter estimators. Slow convergence of the gradient method and unsuitability of the standard least squares for time varying parameters led them to study parameter estimation methods such as bounded gain forgetting and cushioned-floor method [56] with desirable robustness and convergence properties.

In this method like the previous one discussed, there is no need for measurement of joint acceleration or for inverting the estimated inertia matrix. In fact it is an extension of the previous method in the sense that the tracking error in this method is essential to the Lyapunov convergence analysis.

6.2.4 Method of Craig

A description of this method can be found in [16]. In this proposed control law in the ideal case of perfect knowledge of parameter values and no disturbances, the gains of the closed loop dynamics may be chosen to place the poles of the system, and is of the form

$$\tau = \hat{M}(\theta)\ddot{\theta}^* + \hat{Q}(\theta, \dot{\theta}) \tag{6.25}$$

where circumflex represents the estimates and

$$\ddot{\theta}^* = \ddot{\theta}_d + K_v \dot{E} + K_p E \tag{6.26}$$

E representing the difference between the desired and true values of the angles. Including the control law in the dynamic equation of the form

$$\tau = M(\theta)\ddot{\theta} + Q(\theta, \dot{\theta}) \tag{6.27}$$

the following error equation is obtained

$$\ddot{E} + K_v\dot{E} + K_pE = \hat{M}^{-1}(\theta)[\tilde{M}(\theta)\ddot{\theta} + \tilde{Q}(\theta, \dot{\theta})] \tag{6.28}$$

where *tilde* represents errors in the dynamic model used in the controller, i.e. the difference between the estimated values and the actual values.

The error equation is written in the form

$$\ddot{E} + K_v\dot{E} + K_pE = \hat{M}^{-1}(\theta)W(\theta, \dot{\theta}, \ddot{\theta})\Phi \tag{6.29}$$

Φ being the $r \times 1$ vector of parameter errors and W an $n \times r$ matrix of functions.

The filtered servo error

$$E_1 = \dot{E} + \Psi E$$

where $\Psi = diag(\psi_1\, \psi_2 \cdots \psi_n)$ with $\psi_i > 0$

is used, so that parameter estimates can be changes as a function of this. This is the adaptation law.

The criterion for choosing ψ_i is that the transfer function

$$\frac{s + \psi_i}{s^2 + k_{vi}s + k_{pi}}$$

is strictly positive real.

In the state space form the filtered error equation is given by

$$\dot{X} = AX + B\hat{M}^{-1}W\Phi E_1 = CX \tag{6.30}$$

where A,B, and C are block diagonal with the matrices of a minimal state space realisation of the filtered error equation on the diagonal, and X is the collection of state vectors.

Then Lyapunov theory is used to derive the following adaptation law

$$\dot{\hat{P}} = \Gamma W^T\hat{M}^{-1}E_1 \tag{6.31}$$

Also the parameter update law is augmented to restrict the parameter estimates within the bounds defined.

In this method stability of the adaptive system in the sense that all the signals remain bounded is shown. Also convergence of the servo errors to zero is shown. A condition on the desired trajectory such that all parameters will be identified after a sufficient learning interval was derived in this work.

Practical applicability of the above method is questioned for two reasons, firstly the requirement to measure joint acceleration is noise prone and secondly the inversion of the estimated inertia matrix which is assumed to remain uniformly positive definite in the course of adaptation is computationally expensive.

6.2.5 Method of Sadegh and Horowitz

As explained in [77], this method exploits the skew symmetry property of the manipulator dynamic equations. When the dynamic equation is written in the form of linear in the parameters, the matrix of known functions is independent of the joint acceleration, which is an advantage.

The method can be easier analysed if its presentation is based on a theorem considered in [71].

The theorem is as follows with different notations:

Let $t \rightarrow \theta_d(t)$ be a given twice differentiable function and define $e(t) = \theta(t) - \theta_d(t)$. Consider the differential equation

$$M(\theta)\dot{r} + C(\theta, \dot{\theta})r + K_v r = \Psi \tag{6.32}$$

where M and C are as in 6.4, $K_v = K_v^T > 0$, r is given by

$$r = F(s)^{-1}e \tag{6.33}$$

where $F(s)$ is strictly proper, stable, and the mapping $-r \rightarrow \Psi$ is passive, i.e.

$$\int_0^T -r^T(t)\Psi(t)dt \geq -\beta \tag{6.34}$$

for all T and for some $\beta \geq 0$. Then $e \in L_2^n \cap L_\infty^n$ [3], $\dot{e} \in L_2^n$, e is continuous and $e \to \bar{0}$ as $t \to \infty$. In addition, if Ψ is bounded, then $r \to 0$ as $t \to \infty$ and as a result, $\dot{e} \to 0$.

Given the system dynamics 6.4, the control law is chosen as

$$\tau = \hat{M}(\theta)a + \hat{C}(\theta, \dot{\theta})v + \hat{G}(\theta) - K_v r \tag{6.35}$$

where r is defined as $r = \dot{\theta} - v$,

$$v = \dot{\theta}_d - sK(s)e; \quad a = \dot{v} = \ddot{\theta} - K(s)e$$

for $K(s) = K_p + K_d s + K_I/s$, an outer loop PID control law.

Substitution into 6.4, gives

$$M\ddot{\theta} + C\dot{\theta} + G = \hat{M}a + \hat{C}v + \hat{G} - K_v r \tag{6.36}$$

Since $\ddot{\theta} = \dot{r} + a$ and $\dot{\theta} = r + v$,

$$M\dot{r} + Cr + K_v r = \tilde{M}a + \tilde{C}v + \tilde{G} \tag{6.37}$$

The left hand side of this equation is identical to equation 6.32 and if the right hand side is arranged in the form of a matrix of known functions (being a function of angle, angular velocity, v and a, which depend on the velocity and acceleration of the reference trajectory) multiplyed by a parameter vector.

$$Y(\theta, \dot{\theta}, v, a)\tilde{\Phi} = \Psi$$

Parameter update law

$$\dot{\tilde{\Phi}} = -\Gamma^{-1} Y^T r$$

for some symmetric positive definite matrix Γ, is used such that the mapping $-r \to \Psi$ is passive.

[3] Standard Lebesgue spaces L_∞ and L_2 are defined as $L_\infty^n(R_+) = \{f: R_+ \to R^n$ such that f is Labesgue measurable and $\|f\|_\infty < \infty\}$ and $L_2^n(R_+) = \{f: R_+ \to R^n$ such that f is Labesgue measurable and $\|f\|_2 < \infty\}$ respectively.

6.2.6 Comments

The above methods take full consideration of the nonlinear and coupled nature of robot dynamics and can be regarded as globally convergent.

However they do not give any account of the transient performance or any discussion on nonuniformity of asymptotic stability that can lead to instability in the presence of small changes in the dynamics.

These issues were recently discussed in [71], where they propose possible modifications to alleviate the problems. They discuss *persistency of excitation*, *update laws*, and *robustness*, but not the effect of unmodeled dynamics or bounded disturbances on these methods.

6.3 Linear Perturbation Adaptive Control

This method involves the use of perturbation feedback control to control the manipulator in the vicinity of a desired trajectory. The formulation reduces the nonlinear control law to a linear one.

A nominal trajectory is used to obtain nominal torques using N-E equations of motion. Then nominal states from the planned trajectory and the nominal torques are used in the manipulator dynamic equation to get

$$\dot{x}_n(t) = f(x_n(t), u_n(t), t) \tag{6.38}$$

where n denotes nominal.

Then dynamic equation is linearised about the nominal trajectory using Taylor series expansion and 6.38 is subtracted from it to obtain

$$\delta\dot{x}(t) = \nabla_x f|_n \delta x(t) + \nabla_u f|_n \delta u(t) \tag{6.39}$$

where $\nabla_x f|_n \text{and} \nabla_u f|_n$ are the gradients of $f(x, u, t)$, evaluated at x_n and u_n respectively, and δ represents the difference between the actual and nominal values. The control problem is then to determine $\delta u(t)$ that drives $\delta x(t)$ to zero and the

system remains asymptotically stable.

The gradient functions above are complex and are not known exactly, so an adaptive approach is needed.

Lee and Chung [52], discretise the model and by assuming the parameters of the system are slowly time varying, all state variables are measurable, and measurement noise is negligible, use recursive least squares for identification and employ a coupled linear quadratic controller.

Their scheme presents a heavy computational burden that slows down the adaptation rate in real time, as $6n^2$ parameters need to be estimated on line, n being the number of links.

Takeyaki and Arimoto [90] by using an approximated robot model investigated an adaptive control law for δu based on MRAC Lyapunov design technique, and by simulation demonstrated the effectiveness of their method.

Vokobratovic and Kircanski [96] presented a method based on asymptotic regulator properties that relied on linearisation of the dynamic manipulator model along a given nominal trajectory. It also involved an algorithm for synthesis of a robust linear regulator to ensure the decoupled control of the system.

The resulting equation of the linear perturbation method is in general time varying because of nonconstant nominal trajectories, and hence determination of the stability of the system is difficult.

Bounds on the error terms can be imposed and then the stability of the linear system can be investigated.

6.4 Model Reference Adaptive Control

This method was originally introduced by Whitaker in 1958, basic ideas of which are given in [73]. This method was applied to robotics by Dubowsky et al [19]. In this method a model which gives a desired performance response is chosen, and the the difference between this and the system's response is the basis for adjustment

of the controller's parameters in order to provide a suitable control input to the plant. Usually the desired reponse is chosen to be a stable linear time invariant decoupled system.

The adaptation algorithm is designed based on asymptotic stability requirements of the MRAC. Lyapunov, hyperstability criteria etc. have been used to ensure stability. There are various ways of implementation, a continuous-time direct method could be summarised as follows

Defining the plant as

$$y(t) = \frac{B}{A}u(t) \tag{6.40}$$

where u is the control input, y the plant output, and A and B are polynomials in the differential operator. We assume that the system is proper (i.e. $\deg A \geq \deg B$) and A is monic (its first coefficient is unity).

Let the model which gives the desired response be represented by

$$y_m(t) = \frac{B_m}{A_m}u_c(t) \tag{6.41}$$

where u_c is the command signal and A_m and B_m are polynomials in the differential operator.

A general control law

$$Ru = Tu_c - Sy \tag{6.42}$$

where R, S, and T are polynomials, can be used.

Eliminating u between equations 6.40 and 6.42,

$$(AR + BS)y = BTu_c \tag{6.43}$$

Factorising B

$$B = B^+B^- \tag{6.44}$$

where B^- represents the factors that contain poorly damped or unstable zeros that can't be cancelled. The assumption that B^+ is monic, assures the uniqueness of the factorisation.

Now equating the closed loop system's response and the desired response:

$$\frac{BT}{AR+BS} = \frac{B_m}{A_m}$$

But $AR + BS$ must have $A_m B^+$ as a factor to obtain the desired closed loop response i.e.

$$AR + BS = A_m B^+ A_0 \tag{6.45}$$

Therefore

$$\frac{B^+ B^- T}{B^+ A_0 A_m} = \frac{B_m}{A_m}$$

where A_0 is interpreted as observer polynomial. Hence

$$\frac{B^- T}{A_0 A_m} = \frac{B_m}{A_m}$$

which means

$$B_m A_0 = B^- T$$

But B^+ divides B_m and as a result

$$\left.\begin{aligned} B_m &= B^- B'_m \\ T &= B'_m A_0 \end{aligned}\right\} \tag{6.46}$$

In Diophantine equation 6.45 B^+ divides R

$$R = B^+ R_1 \tag{6.47}$$

Now if we divide 6.45 by B^+, we get

$$AR_1 + B^- S = A_0 A_m \tag{6.48}$$

Equations 6.46, 6.47, and 6.48 give the controller polynomials.

In the case of a minimum phase system, a controller that cancels all the system zeros can be used, then $A_0 A_m = AR_1 + b_0 S$ then if we multiply by y and use the model equation

$$A_0 A_m y = b_0 (Ru + Sy) \tag{6.49}$$

The polynomials on the left-hand side are known and can be used to estimate the unknown controller parameters on the right.

For pole placement design for example, A_0 is specified and R and T are found from equations 6.47 and 6.46 and substituted in the controller of equation 6.42. R_1 and S should satisfy equation 6.48. In addition pole excess of the model cannot be smaller than the pole excess of the system and the degree of the observer polynomial should be sufficiently large.

Once a suitable structure is chosen, then an error model is derived. The error model should be linear in the parameters, except if the *Gradient* method is used. Tricks such as filtering and error augmentation is used to to derive the error equation.

Following the derivation of the error model, parameters can be updated using a law such as

$$\frac{d\theta}{dt} = \gamma\varphi\varepsilon \tag{6.50}$$

where, when using the Lyapunov stability

$$\varphi = [-u_c \;\; y]^T$$

where u_c represents the command input and y, the plant output. ε is the error. Representation of a model reference adaptive control is shown in figure 6.2.

In general although the method of MRAC is not computationally demanding, it is difficult to implement it and adaptation to disturbances is less than satisfactory.

The method of [19], is entirely based on the Model Reference Adaptive Controller, and the adaptation algorithm is based on the steepest descent, followed by a separate stability analysis using linearisation.

The adaptation is based on acceleration error which requires measurement of the acceleration. The controller functions on joint basis and the coupling terms between joints are ignored. The computation of the torques requires the inversion

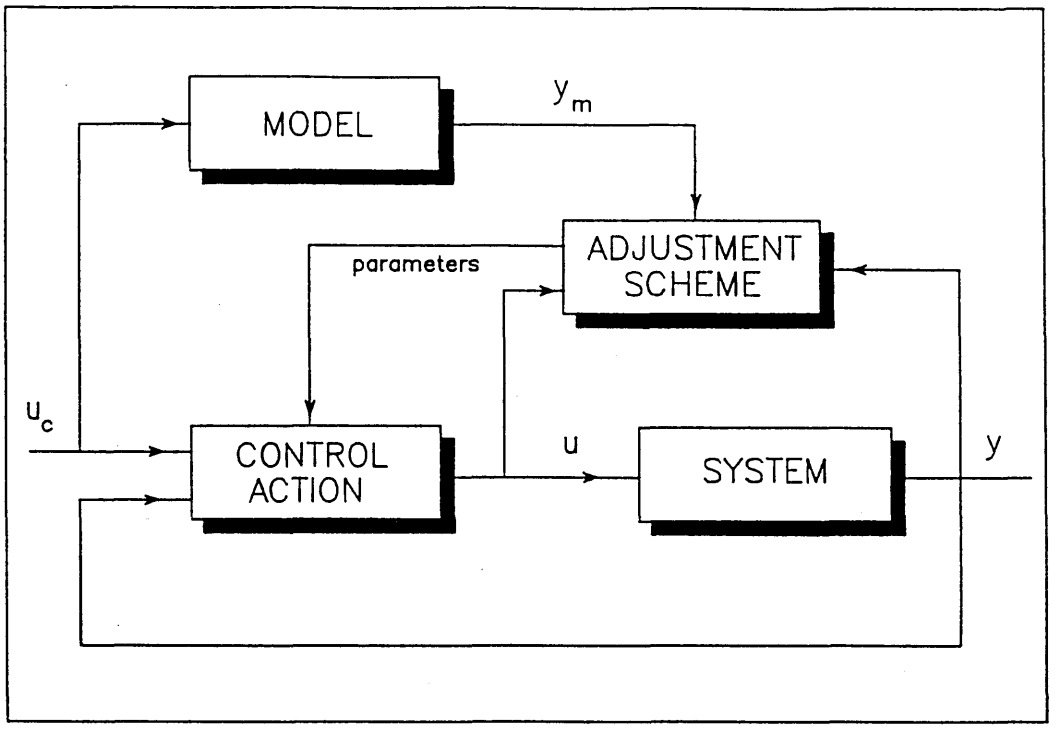


Figure 6.2: Model Reference Adaptive Control

of the generalised inertia matrix which introduces difficulties for real-time implementation.

Later Tomizuka and Horowitz [34], improved on Dubowsky and DesForges approach and based their overall control strategy on MRAC in an inner loop, but included a PID fixed gain controller in the outer loop. Their design method is based on the hyperstability approach, and they explicitly consider the coupling among joints and the nonlinear terms in the manipulator equations of motion. They however, only presented simulation results to show that their control system was insensitive to variations of manipulator configuration and payload.

Anex and Hubbard [5] implemented and evaluated the adaptive control of Tomizuka and Horowitz, giving insight into practical problems associated with implementation, although the structure of the manipulator that they used (Rhino XR-2) does not allow full evaluation of the algorithm as they suggested due to speed limitations.

They addressed disturbances due to Coulomb friction, gravitational loading, and

actuator saturation.

They also developed a *bond graph* model of the manipulator which is simpler than usual models and allows all manipulator subsystems to be included in a uniform way.

They found that the manipulator has significant amount of Coulomb friction which is a nonlinear disturbance, that unless cancelled degrades the performance of the controller and had to include it in the model.

They also found that the motors saturated easily due to an integral gain term in the inner loop of the adaptive controller, which was not really needed for the elimination of the steady state error. This elimination was already being provided by the integral nature of the parameter adaptation of the feedback loop.

They suggested that due to the fact that a system can never be modelled exactly, the adaptive scheme should be modified slightly to make the reference model independent of the plant output.

6.5 Self-Tuning Adaptive Control

In this type of adaptive control strategy, a set of desired controller parameters are found according to a design procedure for example minimum variance, linear quadratic, pole-placement, model-following etc., and an estimation method is used to estimate the unknown parameters. The estimation method can be least squares, stochastic approximation, etc. .

The starting point for self-tuning control was when Kalman in 1958 introduced a deadbeat controller combined with least squares estimation. However at that stage an analysis of the closed loop system was not given. Then it was Wieslander and Wittenmark [100] who based their design on minimum-variance and least-squares and some consideration was given to uncertainties of the estimates. Astrom and Wittenmark [6] presented analysis of the asymptotic properties of a direct self-tuning control whereby controller parameters can be estimated directly.

In the minimum variance design, a prediction model that allows prediction of the output d steps ahead is estimated, and this model is used to determine a control signal that brings the predicted value to a desired value.

A similar type of controller was also developed by Clarke and Gawthrop [14] known as generalised minimum variance, which unlike the minimum variance method does not result in large control signals. It decreases the variation of the control signal by generalising the loss function to contain a penalty of the control signal. It resembles Linear Quadratic self tuning, but with a reduced computational burden due to simplification of the problem on the basis of one-step ahead loss function.

The idea of self-tuning adaptive control can be seen by the block diagram representation of figure 6.3.

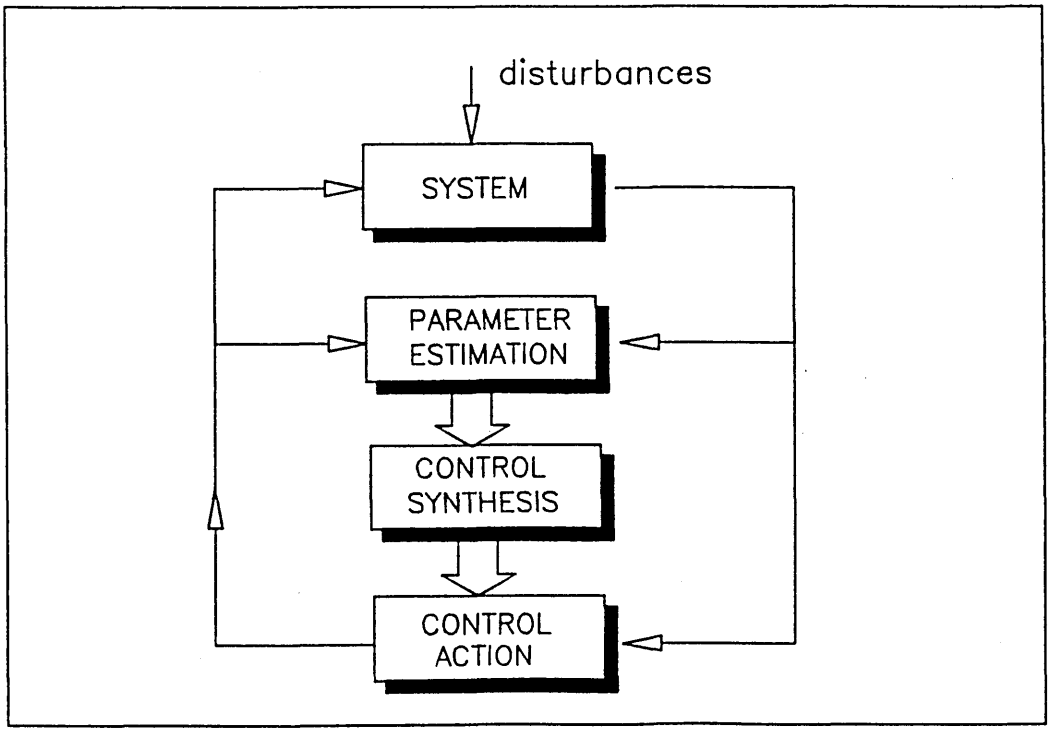


Figure 6.3: Self-Tuning Adaptive Control

As Landau [50] points out the limitations of the first generation of self-tuning controllers based on minimum variance, such as lack of robustness with respect to noise, unmodelled dynamics, and disturbances, lead to the development of pole

placement, linear quadratic control and generalised predictive control.

Self-tuning controllers were originally developed using a discrete-time design approach, and a brief description of the operation of pole placement and model following self tuning in this context is as follows

Defining the SISO plant as

$$A(q)y(t) = B(q)u(t) + C(q)e(t) \tag{6.51}$$

where y is the output of the system, u, the input to the system, $e(t)$ is a sequence of independent equally distributed Gaussian variables, and A, B, and C are polynomials in the forward shift operator q.

For pole-placement and model following, the desired closed loop response can be specified by the following with notation similar to MRAC.

$$A_m(q)y(t) = B_m(q)u_c(t) \tag{6.52}$$

and the controller of the form

$$R(q)u(t) = T(q)u_c(t) - S(q)y(t) \tag{6.53}$$

The Diophantine equation is

$$AR_1 + B^-S = A_0A_m \tag{6.54}$$

variables in paranthesis are not shown for simplicity where

$$B = B^+B^- \tag{6.55}$$

$$B_m = B^-B'_m$$

$$T = A_0B'_m \tag{6.56}$$

$$R = B^+R_1 \tag{6.57}$$

These equations are the basis for various design problems, once causality of the controller is ensured.

6.5.1 Indirect design

The closed loop transfer function B_m/A_m and a desired observer polynomial A_0 is specified and then the coefficients of A, B, and C are replaced by their estimates in equation 6.51 and then equation 6.54 is solved to obtain R_1 and S. T and R are then found from 6.56 and 6.57. The control signal is then calculated from 6.53.

6.5.2 Direct design

The model of equation 6.51 is reparameterised in B^-, R, and S. One way of reparametrisation is by multiplying equation 6.54 by $y(t)$ and using equation 6.51 to get

$$\begin{aligned} R_1Ay(t) + B^-Sy(t) &= A_0A_my(t) \\ R_1Bu(t) + B^-Sy(t) + R_1Ce(t) &= A_0A_my(t) \\ B^-(Ru(t) + Sy(t)) + R_1Ce(t) &= A_0A_my(t) \end{aligned} \tag{6.58}$$

Estimating B^-, R, and S means that the controller polynomials S and R can be found directly. Following this, using equation 6.56 control signal can be obtained from 6.53.

6.5.3 Continuous-Time approach

This approach was developed by Gawthrop, for details see [24], in which controller design is carried out in continuous-time based on a continuous-time representation of the system, and after a continuous to discrete transformation, a discrete-time control is implemented.

A observation made by Landau [50] is probably a good starting point for justifying a continuous-time approach to the design of self-tuning control. He explains that experience with various applications show that the assumption of the plant being represented by a discrete-time model with a stable inverse and a fixed delay is not realistic except for special applications. He then goes on to point out that even successful applications have required a careful selection of the sampling frequency

which is a lengthy process..

The advantages of employing a continuous time approach are listed in [24]. Briefly these are that system characteristics such as relative degree, and zero location can be directly addressed, control engineers find interpretation of continuous time results easier, and there is no need for considerations given to controller sampling interval before and during the design.

The SISO system is represented by the Laplace transform equation

$$\bar{y}(s) = e^{-sT}\frac{B(s)}{A(s)}\bar{u}(s) + \frac{C(s)}{A(s)}\bar{v}(s) \tag{6.59}$$

where $\bar{y}$ is the system output, $\bar{u}$, the system input, and $\bar{v}$, the disturbance input. The polynomial $A(s)$ is the system denominator, $B(s)$, system numerator (control signal), and $C(s)$ is the system numerator (disturbance) and is regarded as a design parameter. The choice of $C(s)$ does not affect the dynamics of the system, as far as the control signal is concerned, and it only affects the interpretation of the disturbance term $\bar{v}$. The choice of $C(s)$ should ensure its stability and its degree should be equal or less than one to the system denominator. A constant or stepped disturbance term can be included in the system model by letting initial values of B and A be zero, while initial value of C is non zero.

Extending the idea of *Smith's Predictor*, which uses an inverse delay (unrealisable) to remove a delay from the loop gain of the system by means of *emulating* the unrealisable component (inverse delay) through realisable transfer functions operating on the system input and output, to also cancel out a high relative degree and zeros with positive real parts.

The unrealisable transfer function that cancels the aforementioned undesirable components, generates quantity $\bar{\phi}$ from the system output as:

$$\bar{\phi} = e^{sT}\frac{P(s)}{Z(s)}\bar{y}(s) \tag{6.60}$$

and as a result the net delay should be reduced to zero through e^{sT}, the net relative degree should be reduced to zero when $deg(P) - deg(Z) = deg(A) - deg(B)$, and

the net number of unstable zeros should be reduced to zero when the denominator of the unrealisable transfer function contains all the unwanted factors of the system numerator.

An *emulator* $\bar{\phi}^*$ can be designed to equal the unrealisable quantity $\bar{\phi}$ in the absence of disturbances:

substituting for y in equation 6.60 from 6.59, and for simplicity not including (s), we get

$$\bar{\phi} = \frac{PB}{ZA}\bar{u} + e^{sT}\frac{PC}{AZ}\bar{v} \tag{6.61}$$

By construction the transfer function that multiplies $\bar{u}$ is realisable, but the one multiplying $\bar{v}$ is not. Now dividing the later transfer function into realisable and unrealisable, that is :

$$e^{sT}\frac{PC}{AZ} = \underbrace{e^{sT}\frac{E}{Z}}_{\text{unrealisable}} + \underbrace{\frac{F}{A}}_{\text{realisable}} \tag{6.62}$$

where $deg(F) < deg(A)$. Substitution of this decomposed transfer function into 6.61, gives

$$\bar{\phi} = \frac{PB}{ZA}\bar{u} + \frac{F}{A}\bar{v} + e^{sT}\frac{E}{Z}\bar{v} \tag{6.63}$$

Now $\bar{\phi}^*$ can be defined as

$$\bar{\phi}^* = \frac{PB}{ZA}\bar{u} + \frac{F}{A}\bar{v} \tag{6.64}$$

$\bar{v}$ can be eliminated from the above equation by using the system equation 6.59

$$\bar{\phi}^* = \frac{EB}{ZC}\bar{u} + \frac{F}{C}\bar{y} \tag{6.65}$$

We can also define $\bar{e}^*$ as

$$\bar{e}^* = e^{sT}\frac{E}{Z}\bar{v} \tag{6.66}$$

So that

$$\bar{\phi} = \bar{\phi}^* + \bar{e}^* \tag{6.67}$$

This emulator can be incorporated in a feedback loop, replacing the output in a conventional control law. In other words if $1/Q(s)$ represents the controller, the

feedback control strategy will be

$$\bar{u}(s) = \frac{1}{Q(s)}[\bar{w}(s) - \bar{\phi}^*(s)] \tag{6.68}$$

where $\bar{w}(s)$ represents the set point.

A block diagram illustrating the above can be seen figure 6.4.

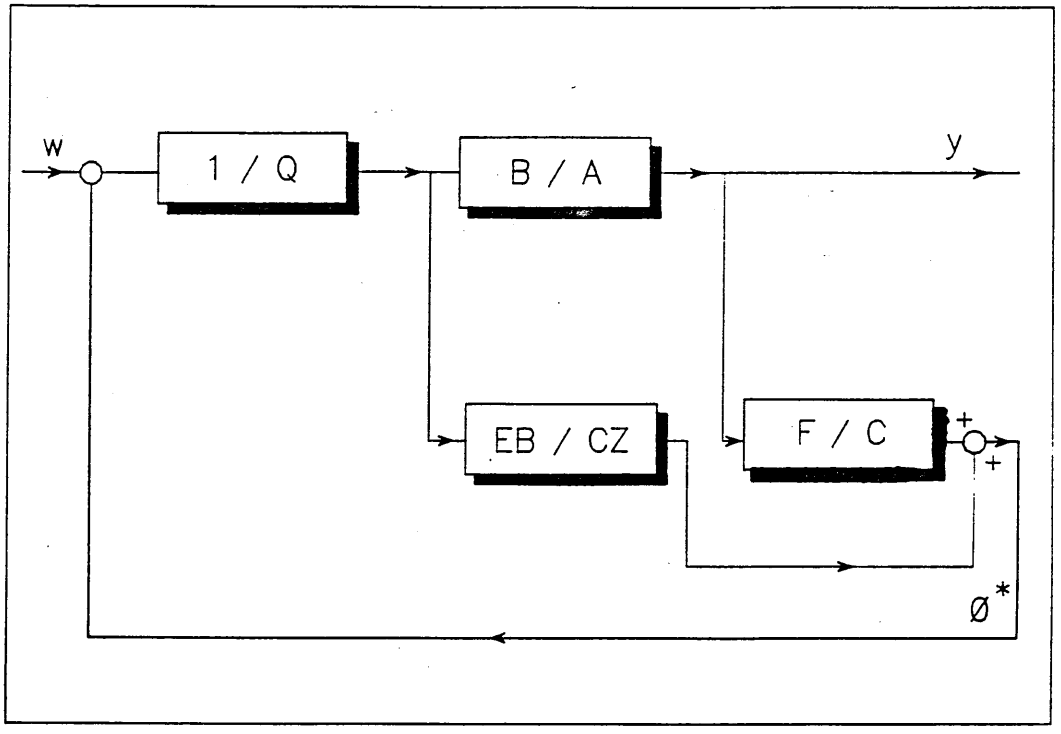


Figure 6.4: Feed back loop representing the emulator

It is easier to analyse the emulator, by replacing the explicit expression for the emulator represented in equation 6.65 by 6.67 in an equivalent feedback loop. The expressions for the properties of the closed loop system are:

Notational loop gain

$$L = \frac{1}{Q}\frac{PB}{ZA} \tag{6.69}$$

Closed loop system output

$$\bar{y} = \frac{BZ}{PB + QZA}[e^{-sT}\bar{w} + \bar{e}^*] + Q\frac{ZC}{PB + QZA}\bar{v} \tag{6.70}$$

Closed loop system input

$$\bar{u} = \frac{L}{1 + L}\frac{ZA}{PB}\bar{z} \tag{6.71}$$

where $\bar{z}$ is the equivalent set point equal to

$$\bar{z} = \bar{w} - \frac{F}{A}\bar{v}$$

A block diagram representation of the equivalent feedback loop is shown in figure 6.5.

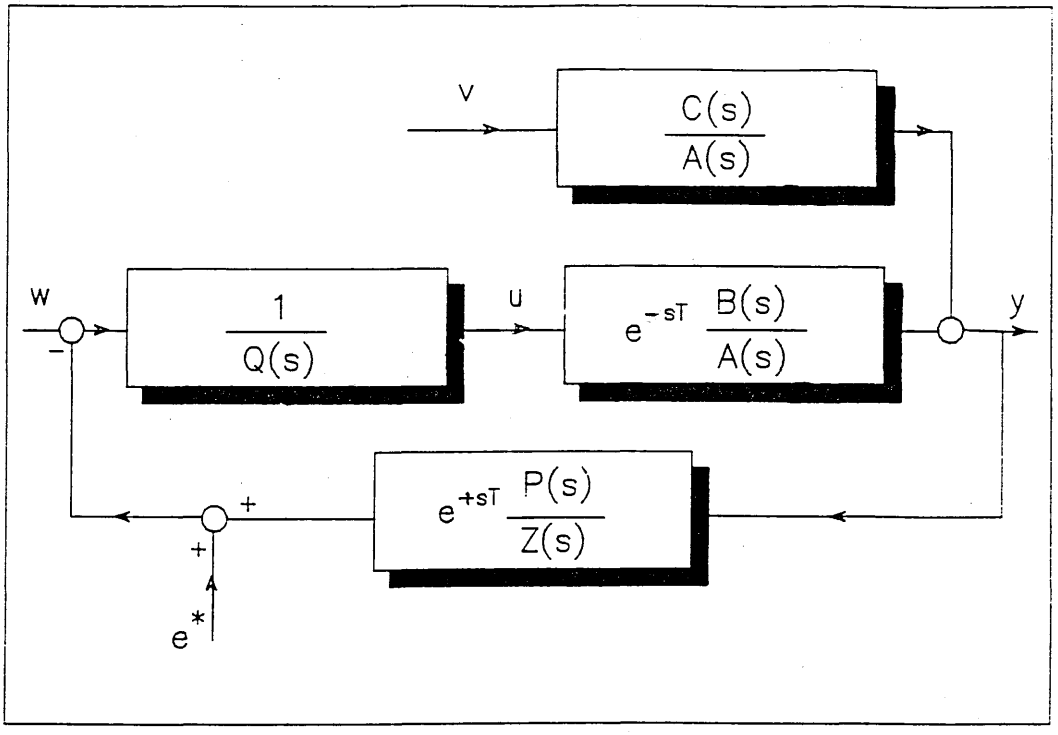


Figure 6.5: The equivalent feedback loop

By selecting particular values of P, Q, C and Z, particular control laws can be chosen, for example

- Smith's method by selecting $P = 1$, $Z = 1$, and $C = A$
- Model reference control by selecting $Z = 1$, and $Q = 0$
- Pole placement control by selecting $Z = B$, and $Q = 0$
- Weighted pole placement and model reference control by selecting Q to be non zero in both cases, typically small at low frequencies to give exact model matching, and large at high frequencies.

When the system is not exactly known, the emulator does not emulate the desired unrealisable transfer function directly, hence poor and unstable closed loop performance results.

In this case, the emulator is combined with a parameter identification algorithm to give a *self tuning emulator*. To do this the emulator equation is rewritten in the linear-in-the-parameter form:

$$\bar{\phi}^* = \frac{EB}{ZC}\bar{u} + \frac{F}{C}\bar{y} = \underline{X}(s)\underline{\theta} \tag{6.72}$$

where *data vector* $\underline{X}$ and the *parameter vector* $\underline{\theta}$ are given by

$$\underline{X} \triangleq \begin{bmatrix} \underline{X}_u \\ \underline{X}_y \\ \underline{X}_i \end{bmatrix} \; ; \; \underline{\theta} = \begin{bmatrix} \underline{\theta}_u \\ \underline{\theta}_y \\ \underline{\theta}_i \end{bmatrix}$$

where

$$\underline{X}_u = \frac{1}{C}\begin{bmatrix} s^n \\ s^{n-1} \\ \vdots \\ 1 \end{bmatrix} u \; ; \; \underline{X}_y = \frac{1}{C}\begin{bmatrix} s^n \\ s^{n-1} \\ \vdots \\ 1 \end{bmatrix} y \; ; \; \underline{X}_i = \frac{1}{C}\begin{bmatrix} s^n \\ s^{n-1} \\ \vdots \\ 1 \end{bmatrix} v_i$$

and $\underline{\theta}$ is given by

$$\underline{\theta}_u = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_n \end{bmatrix} \; ; \; \underline{\theta}_y = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} \; ; \; \underline{\theta}_i = \begin{bmatrix} i_0 \\ i_1 \\ \vdots \\ i_n \end{bmatrix}$$

Issues arising from the above approach including robustness, stability etc. can be found in detail in [24].

This method will be used in a later section in order to introduce a model based variable structure continuous time self tuning controller, in which the model of the manipulator is used in a feedback-feedforward linearisation method of the nonlinear system and then a continuous time variable structure self-tuning control is applied to each joint of the manipulator.

6.5.4 Some approaches based on the self-tuning method

In self tuning control, the controller gains are adjusted at the sampling instances on the basis of current measurements, and sensory information which is available at a sampling time in a suitable form can be utilised in the control strategy, hence the controller can adapt to environmental variations.

Its past performance on a task can be used to improve the present response and performance.

As far as robot manipulators are concerned, the task cycle and as a result the duration of robot's motion is usually finite which is not quite the same as usual self tuner notion. This affects the stability studies, where in self tuning control usually the asymptotic behaviour of the system as the time approaches infinity is studied and these are local in nature.

The stability of a manipulator system with self-tuning controller has not been addressed and for a general six joint manipulator the determination of the global stability in the sense of Lyapunov for example is not within the realm of the designer, at least at present [49].

When the manipulator contains a self tuning controller as a sole controller with no cancellation of nonlinear terms, the stability of the system can hardly be proven.

A self-tuning control scheme for manipulators was proposed by Koivo et al [48] in which coupling between the joints are expressed by a forcing term. These interactions are not fully compensated. In this controller, R and Q in the cost function are restricted to be constant. Desired joint positions and velocities are used, but no systematic computation of the feedback gains is given.

In this approach, an autoregressive model is proposed to model the motion of a manipulator and to design a controller for the system, motivated by the form of the multivariable discrete time model which can be obtained if the mathematical model of the robot is properly linearised about a nominal trajectory and then discretised by Euler's method.

The performance criterion to be minimised is

$$I^k(u) = E\{\|y(k+d) - y^d(K+d)\|_Q^2 + \|u(k)\|_R^2/\phi(k-1)\} \tag{6.73}$$

where $\|\cdot\|_R$ indicates the norm with weight R ($\|u\|_R^2 = u^T R u$), and R is a positive semidefinite symmetric matrix, Q is a positive definite symmetric matrix, and $y^d(\cdot)$ describes the desired path vector as a sequence of discrete points, relative to the admissible controls while satisfying the constraint equation

$$y(k) = \hat{\theta}^T(k)\phi(k-1) + e(k) \tag{6.74}$$

Least squares is used for parameter estimation.

An adaptive controller for interacting joints (MIMO) was also presented, but they showed by simulation that when trajectories are compared, in the cases in which most estimated parameters had reached the steady state values, with the SISO case no relative improvements were observed, indicating that adaptive feature of the closed loop single variable model accounts at least partly for interacting effects.

Koivo [47] also introduced a MIMO discrete-time stochastic model to represent the motion of a robot gripper in cartesian coordinates. It is important to note that planning the manipulator trajectories in cartesian coordinates is preferred to joint interpolated trajectory planning. Amongst other reasons, this is due to the fact that in various applications, the trajectory is specified in terms of the end effector movement.

The R and Q of the cost function were extended to polynomials by Liu [60] and the parameters of these polynomials were adjusted on-line to achieve a closed loop pole assignment scheme with better performance. This scheme was then modified [59] by minimising the variance of a generalised cost function and estimating the parameters directly. The weighting factor in the cost is adjusted on-line to assign closed-loop poles. This was further improved, by introducing the nominal torque feedforward compensation of non-linear coupling among the joints along the desired trajectory.

A pole/zero placement approach was suggested by Linde et al [58] in which the inverse model of the manipulator is also incorporated and the robot plus the inverse show an almost linear behaviour. The resulting system is second order and system parameters are estimated and used to tune a pole/zero placement. It is noted that if B polynomial has a non-minimum phase character, the system becomes unstable.

Another pole placement STR was described in which a 'black box' approach for modelling a robot arm was used [39] and a third order model was considered appropriate. The model was then discretised. The resulting model is of non-minimum phase. In this approach no knowledge of the components of the robot is requires and the model accounts for system elasticity and other system characteristics. However the couplings between the joints are neglected.

Trajectory control is dealt with in [80] based on invertibility and functional reproducibility results. Both joint angle and position trajectory controls are applied. In the latter case it is noticed that, the control has singularities on the boundary of the reachable zone. Robustness issues for small deviations is considered.

Anticipatory action in the form of including subsequent desired joint positions (two step ahead) was introduced by [44]. This adaptive pole-assignment control scheme for robotic manipulators was deduced from the well established computed torque method. In computed-torque, the cross couplings and non-linearities are compensated for by a non-linear feedback law and the residual is controlled by state feedback. It should be noted that desired velocities and accelerations are also needed. The model is recursively identified to allow for the imperfections in the cross couplings and non-linearities.

Leininger [55] adapted Wellstead's pole placement self-tuning algorithm to the closed loop control of multi degree of freedom manipulators, claiming that, as for non-minimum phase systems, the self tuners are very sensitive to system changes and pole shifting controllers developed by Wellstead do not attempt pole/zero cancellation and are thus "more robust" than the methods of Astrom and Clarke

and Gawthrop.

Koivo had used the self-tuning method of Clark and Gawthrop [14] which can be shown to be more robust than the model reference methods discussed earlier, and applied it successfully to joint control of the stanford manipulator, but Leininger suggests that it is not appropriate for control of flexible manipulators or low mass (inertia) manipulators which may exhibit open loop unstable characteristics (Lyapunov sense).

He used a modified version of the autoregressive model

$$A(q^{-1})y(t) = B(q^{-1})u(t-k) + h(t-k) \tag{6.75}$$

where all the symbols are used in their usual representation, and $h(t-k)$ represents all unmodeled effects related to gravity torque and other joint coupled interactions. Then this term is included in the unknown parameter vector of the parameter identification to be estimated, and according to the estimate $\hat{h}(t-k)$ the joint torque input $u(t)$ is modified to eliminate the effects as

$$u'(t) = u(t) - \hat{h}(t-k)/b_0 \tag{6.76}$$

The method is claimed to be suitable for the control of non-linear continuous time systems.

He then extended the approach to task level [54], where the dynamic interaction among the coordinate directions in response to joint actuation are learned on line, hence is in theory capable of correcting for link flexibility, joint compliance, friction and other fundamental nonlinear coupled interactions.

The proposed method assumes that error measurements in tool position, orientation, force and/or moment are available as required.

6.6 Robust Control

Generally when there is a need for faster response in the course of variation of preferably known range of parameters, a robust control design as opposed to adaptive control is sought. This seems quite a useful method to handle the problem of robot control.

There are various approaches for different applications, varying from robust high gain feedback control in which process uncertainties are explicitly dealt with, based on Horowitz's observation that a system in which the Nyquist curve is close to a straight line through the origin can tolerate a significant change of gain, to variable structure systems, with discontinuous feedback, and the salient feature of occurrence of the so-called *sliding mode* on a switching surface where the system remains insensitive to parameter variations and disturbances.

In the case of high gain feedback control, it is impossible to know the attainability of the desired closed-loop specifications before hand, and it leads to a trial-and-error method. This robust technique and others such as adjustment of gain matrices in the LQG scheme (keeping the loop gain less than one at high frequencies), either have not been thought suitable or simply have not been applied to control robot manipulators.

However a number of authors have reported quite reasonable results based on the application of Variable Structure Systems.

6.6.1 Variable Structure Control

Control algorithms based on the theory of Variable Structure Systems (VSS) are designed in such a way that all trajectories in the state space are directed toward some switching surfaces, and once there, slide along them. The system response then only depends on the gradient of the switching surfaces and remains insensitive to parameter variations and disturbances. This property is important in rejecting effects due to Coulomb and viscous friction.

The method consists of a non-linear feedback control that switches discontinuously on a specified surface, not allowing deviation of the natural trajectory of the open-loop system from this surface.

The VSC method is referred to as *robust* since it requires complete knowledge only of the terms in the existing dynamics that may become unbounded. In other words, the design of a variable structure control which includes sliding mode doesn't require accurate modelling and it is sufficient to know only the bounds of the model parameters.

For a variable structure control system of the form

$$\dot{x}(t) = \mathcal{F}(t, x, u); \quad x \in \mathcal{R}^n, u \in \mathcal{R}^m \tag{6.77}$$

control u is piecewise-continuous and is given by

$$u_i(t,x) = \left\{ \begin{array}{ll} u_i^+(t,x) & \text{for } s_i(x) > 0, \\ u_i^-(t,x) & \text{for } s_i(x) < 0, \end{array} \right\} \quad i = 1, 2, \cdots, m \tag{6.78}$$

where s is the switching function, that can also be time dependent. Discontinuity surfaces are defined as $S_i = \{x: s_i(x) = 0\}$ and the intersection of an arbitrary number of them is the sliding surface when it attracts all motions originating in a neighbourhood (locally asymptotically stable with respect to the dynamics) [7].

If we consider each joint of a manipulator actuated by permanent magnet DC motors as a second order linear time invariant system and ignore the coupling of the joints, joint i will be represented by

$$x^T = (x_1, x_2) \qquad x_1 \text{ is angle, } x_2 \text{ velocity}$$

$$\left. \begin{array}{l} \dot{x}_1 = x_2 \\ \dot{x}_2 = ax_2 - bu, \quad a, b > 0 \end{array} \right\} \tag{6.79}$$

The control law is of the form $u = u_i x_1$, where u_i is defined by 6.78. As it is a second order system the switching line is defined as

$$s = cx_1 + x_2 = 0, \qquad c > 0 \tag{6.80}$$

where c is a design parameter, determining the response speed in sliding mode.
The phase trajectories of the two linear time-invariant systems with $u_i = u_i^+$ and with $u_i = u_i^-$ are combined to form the phase plane trajectories of 6.79 under the feedback control law.
As Utkin [94] points out, a property of variable structure systems is that an asymptotically stable system may consist of two structures, neither of which is asymptotically stable.
In the case of the above system, the phase plane trajectories are shown in figure 6.6, and as can be seen, one of the structures is marginally stable and the other is unstable.

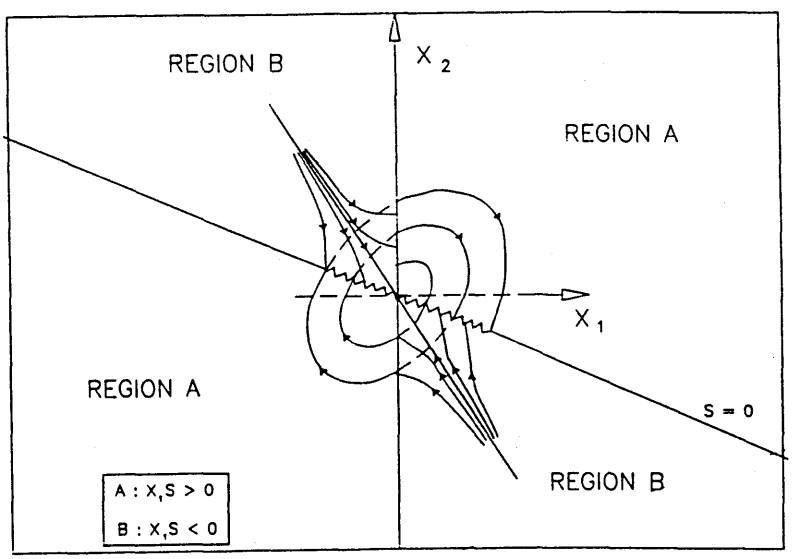


Figure 6.6: Phase plane trajectory of a 2^{nd} order VSS

If u_i is selected such that s and $\dot{s}$ have opposite signs in the neighbourhood of $s = 0$, motion constrained on the sliding line (sliding motion) occurs on $s = 0$, which is not part of trajectories of either linear structures.
The equation of the sliding mode then can be got from 6.80

$$\dot{x}_1 = -cx_1$$

Robot dynamic equations are highly nonlinear, and to design a variable structure system systematic use of the rigid body model has to be made.

For nonlinear systems, there is an additional factor to consider, and that is to ensure in the design that the system trajectories reach the switching line. This can be achieved by including the condition

$$\dot{s}s < 0 \tag{6.81}$$

A number of applications of sliding mode control to robot manipulators have been reported in the literature.

Young [107] applied what is known as the hierarchical approach, whereby sliding mode occurs on the switching planes that are higher in the hierarchy. That is to say initially sliding mode occurs on the switching plane $s_1 = 0$, and then on the intersection of the switching planes $s_1 = 0$ and $s_2 = 0$ and so on until switching mode occurs on the intersection of all the switching planes. This method was provided in [94], to give robustness for VSS with multiple input, as in such systems the existence of sliding mode on the intersection of the switching planes can not easily be guaranteed. The hierarchy of control method replaces the multi-input problem by a sequence of single input problems.

The equation of the sliding mode that is derived is in terms of position errors and derivatives, and not position and velocity. It consists of six uncoupled first order linear systems, each representing the dynamics of a single degree of freedom when system is in sliding mode. The nonlinear interactions are eliminated when in sliding mode.

Young concluded that despite the discontinuous nature of the control signal, the manipulator filtered out some of the high-frequency behaviour caused by delay.

One shortcoming of this method is that when hierarchical control is used, and it is assumed the trajectory lies in the intersection of all preceding sliding surfaces, there is a possibility that convergence to each sliding surface may only be asymptotic which renders the assumption invalid as pointed out by Slotine et al. [81]. Also the resulting control torques are excessive.

The problem of excessive torque can also be seen in the work of Morgan and Ozguner [68] who employed a dynamic coupling compensation.

Slotine and Sastry [81] introduced the concept of time varying sliding surfaces, by defining a solution concept for piecewise continuous dynamical systems with the surface of discontinuity varying with time, based on the results of Fillipov [22] on solution concepts of discontinuous differential equations. They used this to address tracking rather than stabilisation problems.

An undesirable characteristic of the behaviour of systems under sliding mode control is *chattering* which is attributed to non-idealities in control such as time delay, hysteresis etc. This can excite high-freqency unmodelled dynamics.

Slotine et al. modified their method by approximating the discontinuous control laws to obtain continuous control laws with smaller component of high frequency signal and lower control activity, at the expense of lower accuracy in tracking. This method was also faced with the problem of excessive torques.

Later Xu et al. [102] simplified the balance conditioning of Slotine [82] which varies the boundary of the control input interpolation, that smooths out the control discontinuity in a thin boundary layer neighbouring the switching surface, hence removing the chattering, according to the balance condition, to cut down the heavy computational requirements.

Bailey and Arapostathis [7], introduced a simple control law to ensure the stability of the intersection of the surfaces without necessarily stabilising each individual one, hence avoiding the high computational power and speed requirements of the methods above that attempt to make each surface attracting in order to guarantee the asymptotic stability of their intersection. In their method Lyapunov's second method applied to differential equations with a discontinuous right hand side is utilised in establishing asymptotic stability. They also utilise the structure of the manipulator dynamics to establish a sliding surface on the intersection of the switching surfaces.

Briefly, similar to the state space representation of the robot dynamic equation

6.2, the state vector of the dynamics is taken to be

$$x = [\theta \ \ \dot{\theta}]^T$$

Letting $\theta_{di}(t)$ and $\dot{\theta}_{di}(t)$ represent the desired position and velocity of the i_{th} joint of the manipulator respectively, assumed continuously differentiable functions of time, the objective is to make the actual position and velocity track these values. The switching surface is chosen to be

$$s_i(t,x) = s_i(t,\theta,\dot{\theta}) = c_i(\theta(t) - \theta_{di}(t)) + (\dot{\theta}(t) - \dot{\theta}_{di}(t)) \ ; \quad i = 1,2,\cdots,n \quad (6.82)$$

where $c_i > 0$ are constants.

Then introducing a Lyapunov-function candidate to be

$$V(t,x) = \frac{1}{2}s^T M s$$

where M is the inertia matrix of the manipulator model

and using Koditschek's identity [45] and the norm-equivalence property of the euclidean space, both asymptotic stability of the switching surface and the property that every trajectory reaches the sliding surface in finite time is shown.

Yeung and Chen [103] proposed a scheme that takes advantage of the symmetric positive-definiteness of the inertia matrix to develop a control law that does not need taking the inverse of the inertia matrix.

They used the Lyapunov function $V = s^T M s$ and not the condition 6.81 in their VSS controller design. They introduces an auxiliary set of planes to prove that the switching planes were stable, and then transformed the coordinates to get the results.

They used Slotine's approach discussed above to deal with chatterings about the set point, and during the transient phase they eliminated chattering by confining the trajectories within a sector of the phase plane.

The approach presented in the next section will include a model based controller which is augmented with a continuous-time variable structure self tuning control.

6.7 Model-Based Adaptive Control with Load Mass Estimation

Unknown or varying masses of the payloads that the manipulator carries, render dynamic model of the robot highly inaccurate creating varying operating conditions as the inertias vary.

A way of estimating the mass of any load held by a manipulator gripper was developed in chapter 4.

A dynamic model of the robot in the *linear in the parameters* form is obtained by considering the gross motion of the first three links and an additional finite mass of zero length representing the unknown load mass, and symbolically manipulating the N-E equations of motion

$$\tau - (M_2\ddot{\theta} + Q_2) = (M_1\ddot{\theta} + Q_1)m_{LoadMass} \tag{6.83}$$

the symbols represent the usual notation and suffix 1 shows dependence on the load mass of the elements of the equation.

or

$$\Psi = Xm_{LoadMass}$$

and then recursive least squares can be used to estimate the load mass on line.

The method is based on the state-variable filter approach that is used to estimate the parameters of continuous-time transfer functions as described in [27] etc.

Incorporating a varying load mass in the dynamic model of a robot and estimating the value of it on-line, results in a better representation of the behaviour of the dynamic system. This certainly means that when a model-based control strategy is implemented the resulting overall system is more accurately decoupled and linearised.

A block diagram representation of a model-based adaptive controller with load mass estimation is shown in figure 6.7

As can be seen from the figure, the dynamic model of the robot includes a

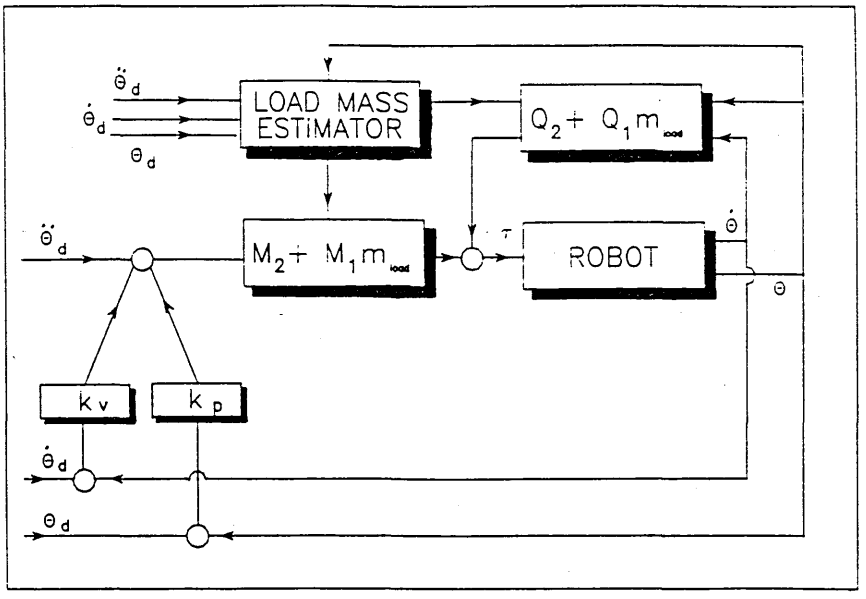


Figure 6.7: Model-based adaptive control with load mass estimation

parameter representing the varying load mass m_{load}. The load mass estimator takes values of the joint angles as well as the desired values of joint positions, velocities, and accelerations at each sample and using the algorithm discussed in chapter 4 provides estimates for m_{load} which is then passed to the routines that compute the dynamic equations. The remaining steps are the same as the computed torque method.

It should be noted that the actuator dynamics are also included in the model and in the control implementation for an MA3000 robot which will be discussed in the next chapter, the value of input voltages that need to be applied to the power amplifiers and in turn to the DC motors are calculated.

In chapter 4 using data extracted from an MA3000 robot, it was shown that the estimated load masses were reasonably close to their actual values.

In the next chapter, the improvements of this method as compared with the computed torque without load mass estimation will be demonstrated by means of implementing both schemes on a robot and comparing the resulting trajectory tracking errors, using a network of transputers.

6.8 Model-Based Continuous-time Variable Structure Self-Tuning Control

In this section a control strategy which is thought appropriate for robotic manipulators is developed and in the next chapter its implementation on an MA3000 robot using a parallel architecture is described. Experimental results in the next chapter provide a valuable means of comparing this scheme with the computed torque and the adaptive model-based method with load mass estimation of the previous section.

First of all, a model based controller is used to cause the cancellation of nonlinear terms in the dynamic model of the robot manipulator. As a result the coupling between the joints will be allowed for and each joint can be considered as a linear system.

This statement would be valid only if the dynamic model of the robot was known very accurately, which due to various factors is not realistic. Consequently if we were to employ a linear controller for each joint, the resulting performance will not be satisfactory.

To cope with the uncertainties and modelling inaccuracies, employment of an adaptive controller is quite an appropriate choice.

In this way variations from the modelled behaviour can be noted via measurements and after the system performance relative to the desired objectives are evaluated, corrective action will be taken in the system.

Adaptive controllers and their application to robotics were discussed earlier, and the advantages of a continuous-time design approach were briefly stated. The adaptive controller that will be employed is the continuous-time self tuning control of Gawthrop.

Robust control and in particular variable structure control schemes for robot manipulators were also looked at in the previous section, and as it has been shown that they are quite suitable schemes for coping with parameter uncertainties, it

will be useful to evaluate their performance on a real robot.

One area in the VSS context that has not been considered in the robotics applications is when feed-back control law uses *output feedback* with *observers* as opposed to full state feedback. This was pointed out by Slotine et al. [81].

In the emulator-based control that was discussed previously, where an emulator (kind of observer) is used to overcome the unrealisability of output feedback, as is shown in [18], if we let the control weighting equal zero it has the effect of replacing it by a relay with the system operating in the sliding mode.

However robustness properties of the above can be improved by using the detuned version of the algorithm, where control weighting is chosen to be nonzero [24]. This is also described in [18], where the control weighting is fed back around the relay, giving the control law 6.68. This is depicted in figure 6.8.

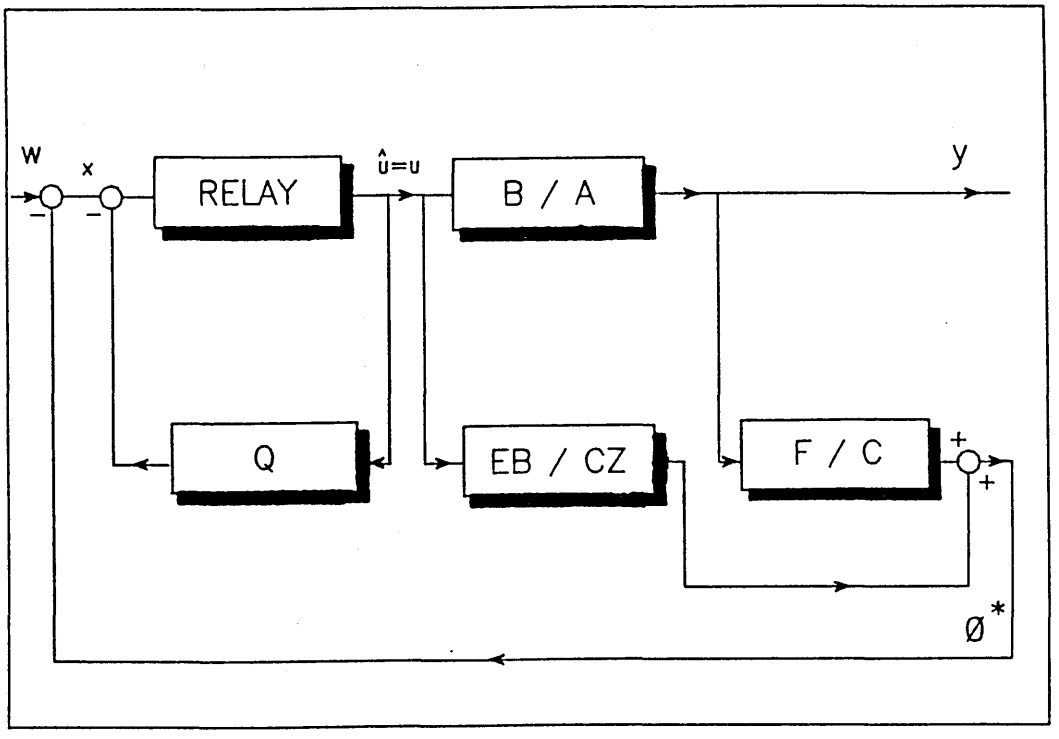


Figure 6.8: Detuned relay control

Sliding mode is obtained when transfer function $Q + G/CZ$ has unity relative order. A first order low pass filter is added in the relay loop if Q has zero relative order.

The block diagram of the overall model based continuous-time VS self-tuning method is shown in figure 6.9.

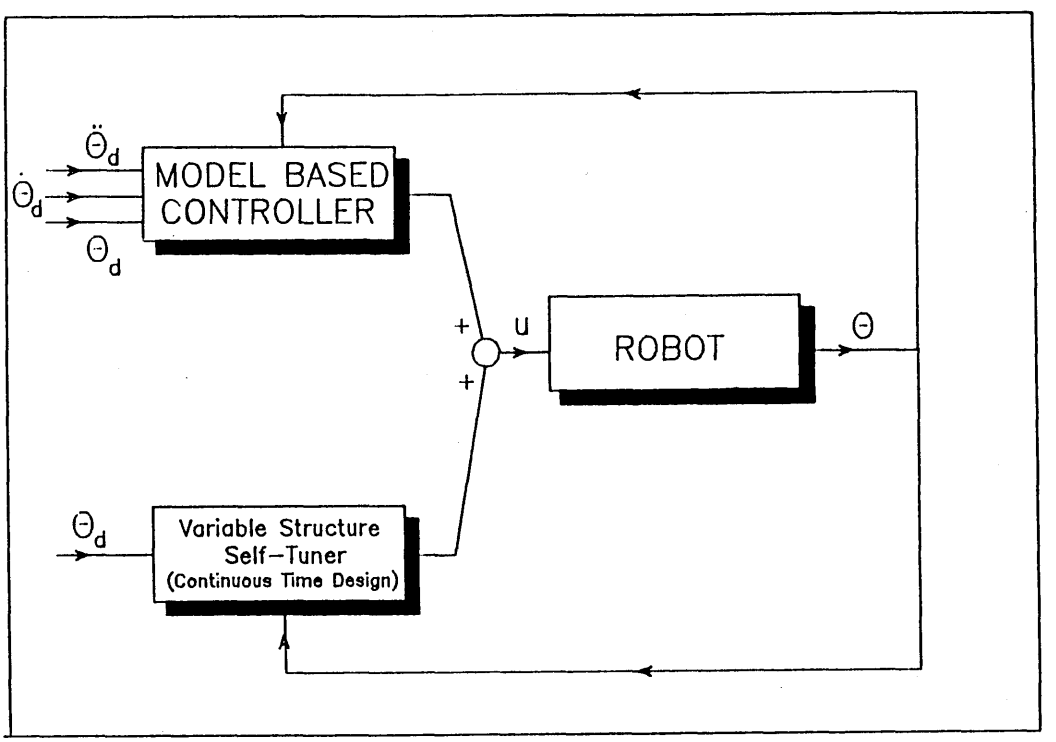


Figure 6.9: Model-Based CVS Self-Tuning Control

The continuous time self tuning part of the algorithm will be *implicit off-line design*, details of which is described in [24], due to its robustness properties and the fact that an off-line design approach is used for implementation.
In the next chapter, the steps for implementation of this method will be included.

Robustness and stability

The affect of the model based part of the algorithm will hopefully be to allow us treat individual joints as linear second order subsystems, in which case standard linear system analysis can be carried out for robustness and stability of the continuous time variable structure self tuners.
Compared to model reference adaptive control which tries to match the model at high frequencies [25], the continuous time self tuner can be made robust by using

control weighting at high frequencies, but not at low frequencies.

An input-output approach as opposed to the state space Lyapunov and Hyperstability approaches to stability is presented in [24] for the continuous time self-tuning controller.

Gawthrop in this reference points out that the advantage of using this approach is the availability of standard textbook proofs.

More specifically complete robust stability results are given for the implicit off-line design when the realisability filter (see next chapter for definition) is equal to one, and partial results are given when this is not the case.

Under ideal conditions, the continuous time self tuning scheme is shown to be globally stable [72] and in the presence of unmodeled dynamics, $\mathcal{L}_\infty$ stability is preserved if a transfer function which depends on the control weighting polynomial, the system and other filters, satisfies a small gain condition.

6.9 Some other methods for control of robot manipulators

Lim and Eslami [57] proposed two adaptive controllers for robot manipulators by using the Lyapunov method first, and then by introducing an auxiliary input applied to the input stage, a new adaptive controller is proposed with better transient response and fast convergence speed as shown by simulation.

Seraji [78] developed a control scheme, structure of which was derived from linear multivariable theory with a direct adaptation law of model reference adaptive control type based on the Lyapunov method. The approach is composed of a feedback controller to provide a stable closed loop system with poles at desired locations in the complex plane and, a feedforward controller to cause the position vector to track the reference trajectory, plus an auxiliary input. It is applied in Cartesian space and measurements of the end effector position and velocity vectors in the

Cartesian space. In this method only simulation results are shown.

Han *etal.* [29] presented an adaptive algorithm with a nonlinear reference model for an N link planar robot with an unknown load. The stability of the error dynamics resulting from the nonlinearity of the plant and the model, is shown using the extended Lyapunov second method for persistent disturbances. To implement the scheme, it will be computationally very demanding.

Luo and Saridis [62] proposed an additional feedback implementation as minor compensating loops built around individual joints in the form of derivatives of the state variables by analogue techniques. This they claim helps to obtain robustness against variations in open-loop dynamics and decreases the effect of nonlinear terms due to couplings in suboptimal control systems with quadratic performance indexes.

Tourassis and Neuman [92] compared the computed torque and the direct-design methods in the context of nonlinear feedback control for robots.

They point out that, in the direct design the reference signal is proportional to the desired joint coordinate vector, which results in a system of uncoupled transfer functions, the gains of which are chosen to ensure that the characteristic polynomials are stable and meet the specified performance.

In the computed torque method, the reference signal is defined to be a linear combination of the desired joint position, velocity and acceleration vectors, which introduces zeros to cancel the poles of the closed loop system and leads to a unity transfer function. The closed loop transfer function in the case of the computed torque indicates that the method is ideally suited for trajectory tracking applications. They showed the validity of this by simulation.

Then they showed that neither approach is robust in the presence of modelling inaccuracies, unmodelled dynamics and parameter errors, and introduced the α-computed-torque that is meant to reduce the driving vector of the linear computed torque error equation and diminish the tracking error, and as a result enhance the

performance of the system, by adding a compensating control signal to the commanded acceleration.

The efficiency and applicability of the method for cylindrical robots was shown.

Khosla and Kanade [43] compared the trajectory tracking performance of the computed torque and independent joint controls. This is one of the very few real-time implementation of the computed torque method, or indeed any model based scheme for manipulators.

Outperformance of the computed torque over the conventional independent joint control scheme in which no acceleration feedforward is introduced, in the absence of torque saturation is demonstrated .

Their experiments also show that even at low speeds the effect of Coriolis and centrifugal forces introduce trajectory tracking errors.

An, Atkeson and Hollerbach [3], presented some experimental results using the three link MIT Serial Link Direct Drive Arm to evaluate trajectory tracking of various controllers.

They established that feedforward control can improve the trajectory following accuracy significantly, especially at high speed movements. They also found that for light links, the unmodelled dynamics including the motor dynamics and friction, become significant.

In addition to the above, numerous other methods have been proposed for controlling robot manipulators. Some of these methods tend to address the specific requirements of robots whereas others only apply methods from the established control theory without specific references to robotics.

Three controllers namely the computed torque, the adaptive model-based controller with load mass estimation, and the model-based variable structure self-tuner, were presented in this chapter and their simulations and experimental evaluations are the subject of the next chapter. These serve to demonstrate:

- The effectiveness of using a model-based approach to decouple and linearise the manipulator dynamics.

- Further enhancements resulting from estimating the payload and additionally augmenting the model-based controller with a self-tuner.

Chapter 7

Control Implementation using Transputers

SUMMARY

In this chapter parallel computations will be considered with particular emphasis on robot dynamic calculations and model based robot control algorithms which utilise the dynamic equations of the robot.

Three robot control schemes are implemented on an MA3000 robot using a network of transputers, and the results are compared based on the trajectory errors produced.

7.1 Introduction

A great majority of Robot control schemes have not yet been tried on real Robots to verify their effectiveness. The true effectiveness of these design theories can only be determined by way of actual implementation and experimental evaluation. By so doing relative merits between many design approaches can be meaningfully established.

The functions of a robot controller can consist of planning, organisation, coordination and decision making at the top of control hierarchy and, joint angle control

at the bottom end. In chapter 2 robot control languages were discussed and a new scheme for comparison of these languages and a method of choosing an appropriate one possessing all the needed attributes for a particular application was presented. The combination of *Transputers* and *OCCAM* was suggested as an ideal way of dealing with the parallel nature of controlling robot manipulators.

In robotics, the computational needs for processing of the external data and implementation of complex control strategies for tasks involving precision path following at high speeds, high band-width compliance and adaptive control, are quite intensive.

However, partitioning of the computational tasks between various independent processors, which can operate in parallel and communicate with each other when necessary, can help alleviate the burden and allow real-time implementation of a desired control algorithm.

For control problem decomposition in general, although some parts of the algorithm can be considered to be efficiently mapped to a standard parallel architecture, a better performance can be achieved, by exploiting the inherent parallelism of the algorithm first and then devising a suitable processor topology, which matches the number of processes and has identical communication structures, in relation to the developed parallel algorithm.

Usually for robot control, the subtasks obtained from a decomposition tend to be fewer in number, but more complex than those obtained in other context, for example signal processing. As a result coarse-grained parallelism, in which there is a small number of processors, each fairly powerful and loosely coupled seem more suitable.

The ability of different processors to execute different instructions at any given point in time i.e. MIMD (Multiple Instruction Multiple Data) is also desirable.

The Transputer, being general purpose, capable of high speed processing with extensive capabilities is a relatively low cost processor which suitably fulfills the robot control computational requirements stated above.

In a previous chapter, the Robot-Transputer Network Interface was described. Here the system is used to study the performance of the transputers, for robot control applications. Different parallel architectures are utilised, to achieve required system performance, with a reduced computational time.

Some potentials of the Transputer as a fast single processing unit and its capabilities as part of a parallel network, for implementation of advanced robot control algorithms are presented.

As a result, the possibility of implementing advanced controllers at relatively low costs which are demanded by high precision applications, but robot manufacturers have not yet responded to, is demonstrated.

The superiority of the Transputer in terms of cost/performance compared to other processors has been shown extensively in the literature, however the decomposition of the computing load into tasks to be allocated to individual Transputers in a parallel network is application dependent. Even for a specific application different network architectures can result in variation in performance.

In this work, no complex task scheduling is involved, rather the parallelism inherent in the task and regular network topology which is suitable for VLSI implementation is sought. Different parallel topologies for computation of some control algorithms are employed to investigate their suitabilities.

Following a brief look at parallel computations, a multiprocessing approach to calculation of dynamic equations of motion of manipulators including a review of the methods used is presented.

Most of these methods are based on N-E formulation for articulated chain dynamics, which yields a set of recursive equations for the dynamic components consisting of forward and backward phases.

The two basic approaches are dedicated multiprocessor approach, breaking the computation into a series of subtasks for execution on independent loosely coupled processors, and the systolic architecture approach, where tightly coupled processing produces the computations.

Then a suitable parallel topology of a transputer network is presented for the computed torque of a 3 degree of freedom robot manipulator, taking into account the disadvantages of large processing times of complex scheduling schemes and looking at the combination of speed up and efficiency.
This architecture can be used for part of the controllers that use the dynamic model of the manipulator for decoupling the dynamics and allow for nonlinearities, such as the model-based variable structure self-tuning controller discussed in the previous chapter.
Then the multi Transputer based control algorithms that are implemented on the MA3000 robot will be discussed and compared.
The control schemes considered are, the Computed Torque, the adaptive scheme in which the load mass carried by the Robot end effecter is estimated on line presented in chapter 4 and, the model based continuous time VS self tuning controller.

7.2 Parallel Computations

Construction of multiprocessor systems with high concurrency is a way of dealing with computationally intensive tasks. This approach has been made more attractive, with the availability of high performance and inexpensive VLSI chips such as the Transputer.
Generally speaking when more processors are allocated to execution of a program, it runs faster. However the law of diminishing returns applies, which means that at a certain point the extra allocated processors can not be utilised efficiently.
A lot of research has concentrated on exploiting the concurrency of of multiprocessor systems in a variety of fields, ranging from numerical methods problems [8] where vector and matrix computations, parallelisation of iterative methods and algorithms for systems of linear equations are considered, to dynamic partitioning of multiprocessor systems [74].

7.2.1 Parallel algorithm representation

One way of representing the precedence relationships between tasks in a concurrent program that can also contain information about the length of time [1] that each task may take to process, is using graphs with directed links.

An example of such representation is shown in figure 7.1.

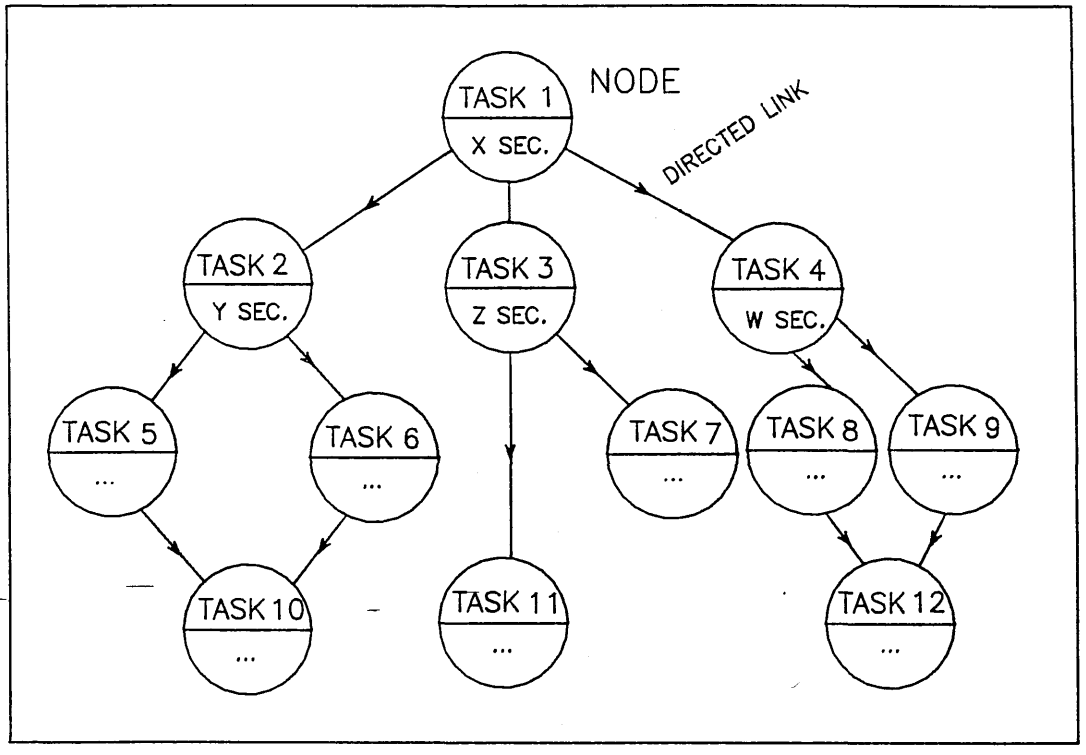


Figure 7.1: An example of graph representation of parallel programs

The graph can be referred to by its set of nodes which represent operations of the program in the form of subtasks and the set of directed links which show data dependencies.

The type and size of subtasks can vary from elementary operations such as addition and multiplication to execution of part of a program.

It will be useful that after specifying the subtasks in a particular order within the constraints (i.e. a processor can only perform one subtask at a time and this is after the subtasks that it depends on have been executed and data from them

[1] The time element is not usually included and simple task precedence graphs or Directed Acyclic Graphs (DAG) are used

has been supplied) imposed, the time taken for each subtask should be measured individually and introduced to the graph, so that decisions can be made as to how effectively subtasks can be assigned to processors or when initially one processor per subtask is designated, the underutilised processors can be identified and their tasks combined.

For programs that need to be partitioned into elementary subtasks, the graph representation might not be very appropriate as it is difficult to represent loops and conditional branches. However for programs that need only be divided to a small number of subtasks without the need for branching and loops, it is ideal as based on the precedence and time for each subtask, rearrangements can be made as to achieve maximum efficiency. In addition enhanced performance can be obtained by allocating the under utilised processors to another parallel subtask.

From the above argument it is obvious that a task is not specified by only one graph representation, but rather various graphs that can represent a task are compared in order to decide which one is most appropriate, using the measures that will be discussed in the next subsection.

7.2.2 Performance characterisation

To quantify the utilisation of computational resources which can be the basis for comparison of various architectures, various measures can be used.

These measures can be in the form of the complexity in terms of the number of processors used, the overall time taken for the algorithm to be executed and communication aspects.

Two concepts that have been used in relation to algorithm comparisons are *Speedup* and *Efficiency*.

Supposing that a parallel algorithm uses n processors, and it takes T_p seconds to complete, where as the best possible algorithm that can run on one processor returning the same results takes T_s seconds, then one definition of speedup of the

algorithm would be

$$S(n) = \frac{T_p}{T_s}$$

Now dividing the speedup by the number of processors used gives the efficiency of the algorithm

$$E(n) = \frac{S(n)}{n}$$

which measures the fraction of time that a typical processor is usefully employed. The ideal situation is that no processor does any work which is unnecessary or remains idle for any length of time and hence the value of efficiency is one and, speed up is equal to the number of processors. Realistically speaking however the aim should be to keep the efficiency bounded away from zero as we increase the number of processors.

It should be noted that there is always a trade-off between speedup and efficiency. An observation known as *Amdahl's law* [2] can be used to obtain the quantification of parallel computational utilisation. It basically expresses that some sections of a program that are inherently sequential can create bottlenecks, when the rest of the program is able to utilise concurrency in a large scale and be executed fast, but having to wait for a long time for the sequential parts to terminate.

This means that, the ratio of the amount of time that is sequential to the total execution time is significant in bounding the execution time of a parallel algorithm. If we call this ratio f, then the speedup is limited by

$$S(n) \leq \frac{1}{f + (1-f)/n} \leq \frac{1}{f}$$

where n is the number of processors as above.

The trade-off between efficiency and speedup can be quantified by introducing the average number of busy processors during the execution of a parallel program when an infinite number of processors is available. This is referred to as *Average parallelism* A and can be used in determining the number of processors that should be allocated to a task in order to achieve an optimal trade-off.

Another definition, of average parallelism is the total service required by the

computation to the length of the longest path in the subtask graph.

The upper and lower bounds on speedup in terms of A are

$$\frac{nA}{n+(A-1)} \leq S(n) \leq min(n, A)$$

Useful as it might be, determination of average parallelism for a system that does not have (as assumed) unlimited number of processors is difficult.

Average parallelism can also be obtained from histogram of the number of active processors over the execution time of a parallel program that is monitored. And consequently this can be used to estimate the appropriate number of processors to allocate.

Number of processors n to be allocated can be found from

$$n = A + \frac{V}{(M-A)(A-m)}[h(m, M, \rho) - A]$$

where V, M, and m are variance in parallelism, maximum and minimum parallelism for the program respectively.

The term in square brackets represents the amount of deviation of processor allocation from A when the parallelism variability is high. ρ is the average processor utilisation.

Eager et al [20] used average parallelism.

In task allocations that is to follow, speedup and efficiency are considered as well as the *average activity* of each processor.

By monitoring individual processors and measuring the percentage of the time that they are busy (i.e. calculating their average activity), in the case of different suitable architectures for a particular problem, one can modify the processor assignment so as to increase the overall average activity and hence efficiency.

7.2.3 Dynamic programming

Although dynamic programming is not going to be applied in this work, a brief mention of it is not quite irrelevant.

Dynamic programming deals with the issue of making optimal decisions sequentially for data routing in a given network of processors.

If graph representations are used, once a decision is selected at a node then the next node is chosen according to a known probability distribution that depends on the selected decision. This leads to a model involving a finite-state Markov chain [2], the transition probabilities of which are influenced by the choice of decision.

A special case of dynamic programming is what is known as the shortest path problem which is to find a desirable path (minimum cost path) for routing data. A cost is resulted from every decision made and each decision affects the options of subsequent decisions. Therefore future situations where high costs might result, should be avoided, while a low cost for the present decision is aimed for.

Unlike dynamic programming, there is no unpredictability of the next node being chosen, once a decision is made at a given node in the case of shortest path problem.

Another concept in parallel programming is the notion of dynamic partitioning. This deals with reconfiguration of multiprocessor systems to accommodate additional programms to the network by optimal partitioning of the system. Markovian models can be used for dynamic partitioning.

[2] A discrete time, finite state, homogeneous Markov chain is a sequence $\{X_k | k = 0, 1, 2, \cdots\}$ of random variables that take values in the finite set (state space) such that

$$Pr(X_{k+1} = j | X_1, \cdots, X_k = i) = p_{ij} \quad \forall k \geq 0$$

where each P_{ij} is a given non negative scalar. In particular the probability distribution of the next state depends on the past only through the current state and since the coefficients p_{ij} do not depend on the time index k, the transition probabilities $Pr(X_{k+1} = j | X_k = i)$ are independent of k.

7.3 Multiprocessor implementation of Dynamic Equations of Manipulators

To achieve better performance in task execution of Robot Manipulators, advanced real-time control algorithms are needed to deal with the non-linear system dynamics and the uncertainties in the robot model.

When implementing robot control algorithms, simplifications are usually made, the most common form of which is to ignore the centrifugal and coriolis forces created by the coupling of the dynamics and each joint is treated as a decoupled system. The dynamic equations for robot manipulators which can be used in advanced control algorithms, to enhance their capability, are computational intensive, but by efficient formulation and exploiting the power of fast processors and parallel processing, the on-line implementation of the control schemes based on these equations are possible.

7.3.1 Review of the work in Parallel Processing for calculating Robot Dynamic Equations

There has been a number of publications that focus on the utilisation of parallel processing to alleviate the computational burden which is faced, when calculating the dynamic equations of Robots. In some specific instances , parallel algorithms are developed to deal with real-time calculation of the manipulator inertia matrix, where as in other cases concentration is on the time-optimum scheduling problem. Majority of these approaches are based on the Newton-Euler (N-E) formulation of Manipulator Dynamics. A parallel processing system (N-E based) was suggested by Luh et al[61] in which one CPU is utilised for each link of the manipulator. With this architecture, as a result of the precedence relations that appear among the subtasks to be executed due to the dynamic coupling between adjacent links and the large number of choices of alternative subtasks, the effort of searching

for minimum-time schedule is complicated. To overcome this, they developed a "variable" branch-and-bound method which discards the branches whose conservatively estimated execution times are longer than the current established upper bound. Also to bring the total processing time to a manageable level, the subtasks need to be arranged so that the precedence relations no longer exist. The usefulness of the scheduling problem was later demonstrated on an experimental processing system by Kasahara et al [40] They considered the optimal assignment of tasks to varying number of processors.

Two parallel algorithms executed on special-purpose processors were proposed by Lathrop [51], one a linear parallel N-E and another logarithmic parallel N-E. In this work inertial information via coordinate transformation between successive pairs of aggregate bodies are consolidated and as a result of this a reduction in the time requirement is accomplished by avoiding a serial recursion. Another multiprocessor system was proposed by Nigam and Lee [70] which consisted of available microprocessors arranged in such a way as to permit piplining at a macro level with parallelism within each macro block. In this work there is no reference to the fact that the time requirement of a piplined algorithm is governed by the number of machine cycles rather than arithmetic operations. In spite of algorithmic constraints imposed by the pipelined computations Wander and Tesar [99] showed that the number of arithmetic operations can be reduced by expressing the time-varying inertia content of a manipulator in terms of kinematic influence coefficients which are represented by explicit functions of only the generalised coordinates.

More recently with the availability of the Transputers which in terms of cost and performance have superiority over many other processors, being classed as one of the fastest 32-bit floating-point processors available, a few have attempted to show their potential applicability in the Robot Control Applications. A new

parallel algorithm for inverse dynamics based on Kane's formulation [3] for manipulator dynamics was introduced by Hashimoto and Kimura [30] which was said to be suitable for VLSI implementation. The algorithm was implemented using 4 T800 Transputers with a good response time. Two architectures for parallel computation of N-E equations were simulated by Jones and Entwistle [38] using Transputers. First, one processor per link which results in loss of parallelism which was shown to be poor in terms of processor utilisation. And second, using a processor farm which has the advantage of being able to change the number of processors used, but the performance is not good due to the large scheduling times required. An interesting observation in this work is that they showed, the Transputer's effective link transfer rate is slower than the quoted values and is dependent on the size of data block transferred. For example for T800-20, the quoted value is 20 Mbits/s, whereas they found it to be 6.5 Mbits/s to pass blocks of 32 bit words. it should be noted that in the algorithms this is a small percentage compared to the time taken to compute the tasks.

7.3.2 Parallel calculation of robot dynamic equations using transputers

Three different Transputer network architectures for computing the torques that result from robot dynamic equations are shown in Figure7.2 for the case when three links of the manipulator are considered.

The OCCAM TIMER was used to calculate the overall time and in one case the time taken by individual processors. From this the performance of each is evaluated in terms of speed-up and efficiency. The results are shown in the table 7.1.

In the case where 3 Transputers are used plus 2 for communication, a speed up of 1.478 and an efficiency of 0.49 is achieved. Each row of the inertia

[3] Kane's formulation describes the N-E in a different way, resolving the task of kinematic and dynamic computations into a set of subtasks

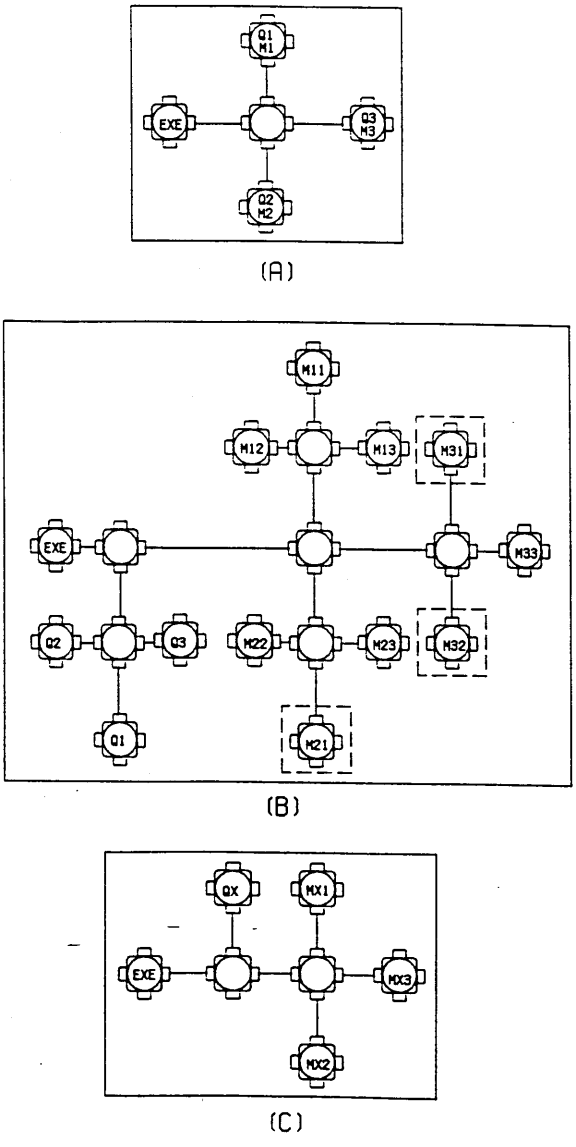


Figure 7.2: Three Transp. Architectures

No. of Procs	Time (ms)	Speed up	Efficiency	Ave. activity
one EXE	6.797			
3 plus 2 communication	4.6	1.478	0.49	
(M11 M12 M13 Q1)				95.65 %
(M21 M22 M23 Q2)				43.48 %
(M31 M32 M33 Q3)				21.72 %
13 plus 6 communication	2.832	2.4	0.18	
10 plus 6 communication			0.24	
4 plus 3 communication				
(M11 M12 M13 Q1)	2.75	1.47	0.37	
(M21 M22 M33 Q2)	1.175	1.43	0.36	
(M31 M32 M33 Q3)	0.733	1.24	0.31	

Table 7.1: Table of comparison for diff. topologies

matrix plus one member of the force vector are assigned to one Transputer.
Looking at the average activity of individual processors, it can be seen that the computational intensity decreases from top to bottom. As a result perhaps a different architecture which combines two of the processors with lower average activity can be used to increase the efficiency. However as each row corresponds to a particular joint torque, having taken account of the coupling, the topology which was explained earlier (ie. one transputer dedicated to one joint which could further be replaced by a suitable number of processors to increase efficiency) is not suitable because of the communication overheads.
When 13 Transputers with an additional six for communication is used, the speed up does not really justify the loss of efficiency. Even when the symmetrical format of the inertia matrix is exploited in the 10 processor case, there is not much improvement. This architecture might be relevant when a 6 x 6 inertia matrix is computed when six joints of the robot are modelled. The final set up shows the comparison of dividing the computation of each of the rows of the inertia matrix and one member of the force vector between 4 Transputers. As can be seen efficiencies can be achieved with reasonable speed ups. Again this might be taken advantage of for larger inertia matrices.

7.4 Parallel Implementation of some Control Algorithms on an MA3000 robot using Transputers

Taking the bending mode frequencies of manipulators into account, sampling frequencies of 100HZ and above are quite sufficient for manipulator control implementation. Employing one transputer is more than sufficient for implementing a PID controller. Model based controllers in general require more computation time

and as a result, by utilising parallel architectures, the required sampling frequencies can be achieved.

To implement the following control methods, a three segment trajectory planning scheme in joint coordinates is used for the gross motion (motion of the first three links) of the manipulator, to give a realistic representation of a real situation in robotic applications.

Usually trajectories are specified in cartesian coordinates in the form of straight line segments which when converted to joint coordinates are not straight line any longer. Having to convert individual points from cartesian to joint space, results in problems of large memory requirements etc., and as a result the usual practise is to approximate the joint movements by polynomial functions.

The specified path together with constraints imposed by the manipulator's dynamics and the path constraints should satisfy a set of boundary conditions to ensure continuity of position, velocity and acceleration of the joints, for a smooth movement.

Depending on the specifications, a different order polynomial can be used and this is dependent on the type of task the manipulator is employed for, for example when position and velocity of the joint is specified at the initial and final times, they provide boundary conditions that can be satisfied by a third order polynomial with four unknown parameters.

The resulting equations can be simplified considerably by introducing a normalised time.

The approximate function used for the first segment (i.e. when the movement starts) is a third order polynomial of the form

$$\left.\begin{aligned} p_1(t) &= [\delta_1 - v_0 t_1 - \tfrac{a_0 {t_1}^2}{2}]t^3 + [\tfrac{a_0 {t_1}^2}{2}]t^2 + (v_0 t_1)t + p_0 \\ v_1 &= \tfrac{3\delta_1}{t_1} - 2v_0 - \tfrac{a_0 t_1}{2} \\ a_1 &= \tfrac{6\delta_1}{{t_1}^2} - \tfrac{6v_0}{t_1} - 2a_0 \end{aligned}\right\} \qquad (7.1)$$

Then the second segment is represented by a fifth order polynomial of the following form

$$\left.\begin{aligned} p_2(t) &= [6\delta_2 - 3v_1t_2 - 3v_2t_2 - \tfrac{a_1t_2^2}{2} + \tfrac{a_2t_2^2}{2}]t^5 \\ &+[-15\delta_2 + 8v_1t_2 + 7v_2t_2 + \tfrac{3a_1t_2^2}{2} - a_2t_2^2]t^4 \\ &+[10\delta_2 - 6v_1t_2 - 4v_2t_2 - \tfrac{3a_1t_2^2}{2} + \tfrac{a_2t_2^2}{2}]t^3 \\ &+[\tfrac{a_1t_2^2}{2}]t^2 + (v_1t_2)t + p_1 \\ v_2 &= \tfrac{3\delta_n}{t_n} - 2v_f + \tfrac{a_ft_n}{2} \\ a_2 &= \tfrac{-6\delta_n}{t_n^2} + \tfrac{6v_f}{t_n} - 2a_f \end{aligned}\right\} \quad (7.2)$$

And finally a third order polynomial is used for the last segment.

$$\left.\begin{aligned} p_n(t) &= [\delta_n - v_ft_n + \tfrac{a_ft_n^2}{2}]t^3 + (-3\delta_n + 3v_ft_n - a_ft_n^2)t^2 \\ &+[3\delta_n - 2v_ft_n + \tfrac{a_ft_n^2}{2}]t + p_2 \end{aligned}\right\} \quad (7.3)$$

In the above equations p represents position, v velocity, and a acceleration, t is normalised time, $t \in [0, 1]$ and is equal to

$$t = \frac{\tau - \tau_{i-1}}{t_i}; \quad \tau \in [\tau_{i-1},\ \tau_i]$$

where $t_i = \tau_i - \tau_{i-1}$ is the real time required to travel through the i^{th} segment, and τ represents the real time. δ is the difference between successive positions. A graph of combination of the three functions for arbitrary positions and durations can be seen in figure 7.3.

After set points are generated from the above trajectory planning method, the actual resulting trajectory of the joints under the controller commands are subtracted from the desired values and the resulting errors are graphically shown for each control scheme. This way the methods can be compared.
The first controller implemented is the Computed Torque method, then a model based controller based on load mass estimation and finally a model based continuous time variable structure self tuning.

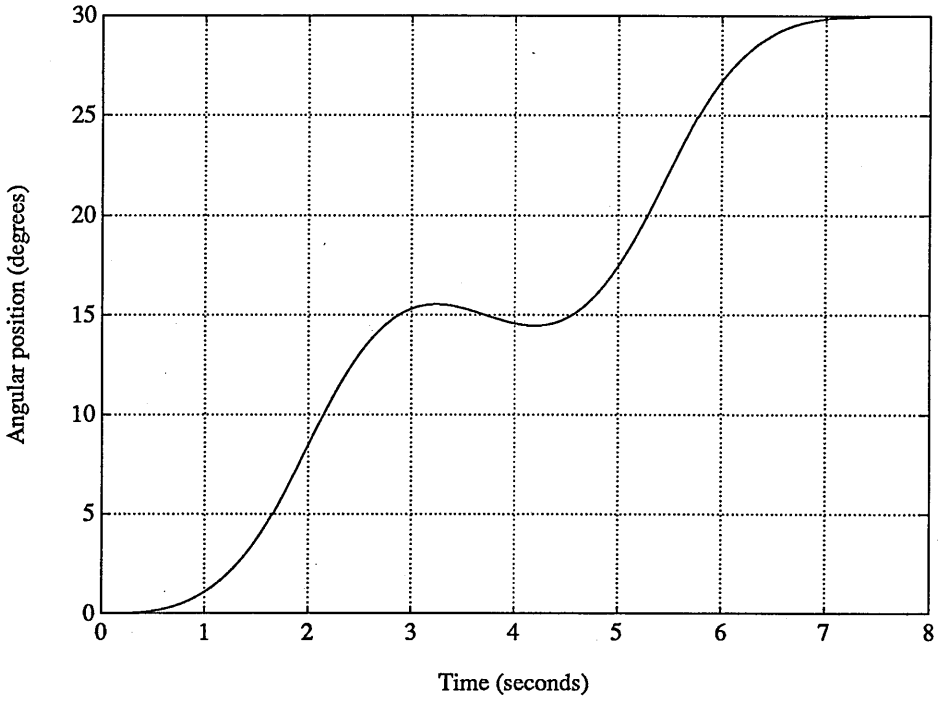


Figure 7.3: A three segment (3-5-3) polynomial approx. joint trajectory

7.4.1 Computed Torque

Parallel implementation of the computed torque scheme is discussed here.

Based on the findings of the previous section the most appropriate architecture for the computed torque method when the dynamic equations correspond to three joints of the manipulator is option (A) of Figure 7.2. In other words by using one processor to calculate the three members of the inertia matrix and the member of the force vector that correspond to one joint, the sampling frequency needed can be achieved and it leaves room for extra calculations to be carried out when controllers with added demands on computing are to be implemented without going above the sampling frequency.

This is particularly useful since the next two control strategies to be implemented are model based and use the dynamic equations of the computed torque and as such having one processor dedicated to a joint cuts down the amount of alterations

necessary and in turn reduces the complications that might arise when communication aspects are considered. Furthermore additional processors can be used with minimum effort for computationally demanding tasks as shall be seen.

Figure 7.4 shows the basic set up for implementing the computed torque method.

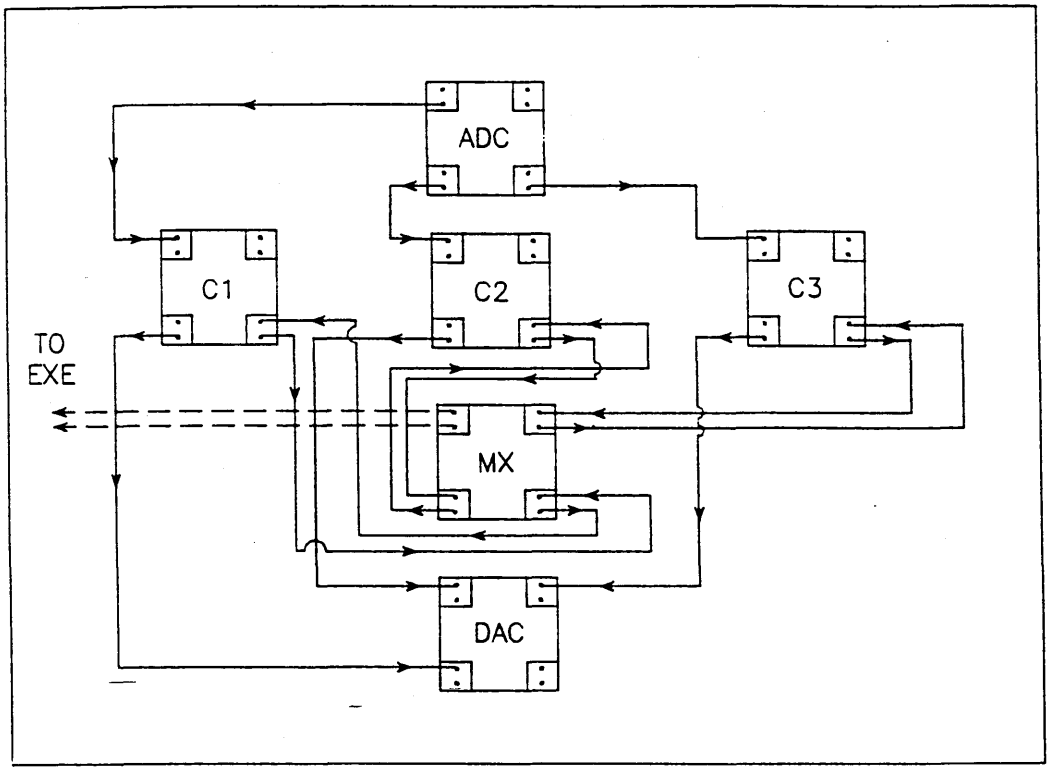


Figure 7.4: Basic transputer topologoy for the computed torque

The EXE process which sits on the host processor is used to signal the start of the control operation. The MX process running on the root processor acts as a multiplexer, recieving set point trajectory data from the EXE and distributing it to the controller processors for individual joints C1, C2, and C3 and then after the controllers have completed their operations, it receives the trajectory errors from them and passes these to the EXE to be filed.

The control processes receive angular data at each sample from the ADC and using these calculate the voltages for each motor and pass the values on to the DAC which in turn sends the analogue signals to motor amplifiers.

The calculations are based on the customised dynamic equations which were developed as discussed in chapter 3. An occam library called robdy was created to

implement these equations in occam and was used by the three controllers.
In Appendix A part of the occam code of the EXE and PROGRAM for the implementation of the computed torque is shown.
As it was mentioned in chapter 5, due to the fact that ADC and DAC are only available on the Parallax, there is a need for the Meiko and the Parallax to communicate.
A Transputer Subsystem Interface (TSI) software allows a transputer domain on the Meiko to have an external dummy processor which can communicate with the rest of the transputers on the domain using the Inter-Module Control Interface provided on the *TO SLAVE* 9 way connector mounted on the rear panel and Inter-Module Link Interface of the same connector or a 37 inter cabinet link.
For convenience and as it did not increase the computation time drastically, it was decided to only use the TO SLAVE. As a result only one output and one input link exist for communication between the Meiko and the Parallax hence four extra processors are used for a smooth multiplexing.
The transputer set up used with extra processors is similar to the one shown in Figure 7.6 with the exception of the processor for filtering and connections for Load Mass Estimation (LME) processes.
A TSI hardware allows ECL-TTL and TTL-ECL translation to be performed as described in chapter 5, for Meiko - B008 compatible board communication.

7.4.2 Model-based control with load mass estimation

The adaptive manipulator control schemes often ignore the effect of the payload mass on the inertia characteristics of the manipulators, in order to make simplifications.
What is clear is that, modelling of the payload can improve the accuracy of dynamic representation of the manipulator and hence the schemes that rely on the

dynamic equations to decouple the model and allow for nonlinearities, can be implemented more accurately.

The approach taken to include the load mass held by the gripper in the manipulator dynamic model was described in the previous chapter. To implement this, in addition to the customised robot dynamic equations similar to those for the computed torque method which are implemented in OCCAM, the numerical solution of the state variable filter given by the n^{th} order differential equation

$$\frac{d}{dt}\underline{X}^c(t) = A\underline{X}^c(t) + Uu \tag{7.4}$$

which is the controllable state-space form of the differential equation for a strictly proper subsystem, is required.

As explained in [24], if the subsystem is not strictly proper, an extended state vector can be used. The above differential equation is written as an n first order differential equations plus an algebraic equation. The Laplace transform of the extended state vector with zero initial conditions is:

$$\underline{X}^c(s) = \frac{1}{A'(s)} \begin{bmatrix} s^n \\ s^{n-1} \\ \vdots \\ 1 \end{bmatrix} \bar{u}(s) \tag{7.5}$$

In this formulation the states are all derivatives of the partial state $\bar{x}_n^c$.

The above is fully explained in [24] and its implementation in Pascal is discussed in [26] as a procedure in an emulator-based self-tuning control program (CSTC). The program includes both continuous-time and discrete-time implementations of the state-variable filter. As the use of continuous-time version has more advantages [26], this is the one which is used.

Briefly, equation 7.4 is integrated between two consecutive time points and two of the terms are expanded in a truncated Maclaurin series and as a result a recursive scheme is obtained.

In a modified version of the continuous-time state variable filter that is compiled

as a self contained program, Functions StateOutput, Delayed, DelayFilter, and Procedure TimeDelayInitialise are not included. In addition modifications are made to allow communication with the OCCAM programs that addresses the ADC and DAC and implements the customised dynamic equations of the robot. A listing of the program can be seen in Appendix B.

The communication between the Pascal and OCCAM is described shortly.

Using the Pascal program, a discrete-time approximation to a continuous-time transfer function is achieved, the accuracy of which can be changed by varying the sample interval and the approximation order.

The input to the Pascal program is the joint angles at each sample and the results for each angle input, assuming a second order filter are:

$$FilterState.State[i] = \frac{s^{-i}}{C(s^{-1})} \times \theta$$

where θ represents the angle input.

In other words filtered angle, filtered angular velocity, and filtered angular acceleration are:

$$FilterState.State[2]; \quad FilterState.State[1]; \quad FilterState.State[0]$$

respectively.

Simplifications on the basis of the sparseness of the A matrix to reduce calculations is noticeable.

Communication between OCCAM and Pascal (OCCAM harness)

The Pascal program was compiled using a Prospero Pascal compiler designed for the Meiko Computing Surface. Both the compiler and the generated programs require the T800 transputer for execution. MeikOS, CS Tools, and occam operating environments are supported. The assembler and linker needed to produce an executable program form part of the operating system and as a result the executable programs produced run independently of the Pascal software.

Besides the features of Standard Pascal, it contains a number of extensions such as an additional data type 'channel' which is a general channel type occupying one word.

One feature of Prospero Pascal which was utilised for communication with OCCAM programs is access to the operating system by calling routines using the C calling conventions. It has a language extension to allow C routines to be called directly. In this way, attributes of the transputer such as efficient execution of many concurrent processes and management of communication between them using its links and input/output facilities can be employed.

A C routine is declared as being a procedure or function by using the C directive provided and is used in a similar manner to an external Pascal procedure. The body of the procedure must not be given and it must be declared at the outer level. The procedure can not be passed as a procedural parameter. A procedure or function declared as a C function obeys the C calling convension, and in particular, the parameter list does not need to be reversed. When the C routine is declared in the Pascal program, an underscore must prefix the name of the routine.

A C library exists which provide routines that enable C procedures that are called from the Pascal to make use of the transputer features. Messages are sent and received on channels using two routines in particular that provide support for the transputer architecture, namely cwrite and cread respectively. The parameters of these routines are

- The channel for sending or recieving messages
- A message buffer
- The size of data to send or receive

As explained in a previous chapter, communication using channels is always point to point and synchronous.

Channel access is provided in the C routine by including a header file *chanio.h* which defines the type CHAN and the C macro function CHAN *occam channel();. The macro indexes the channels which correspond to an OCCAM procedure's array parameter representing channels (chanArray). The elements of this array can be passed as parameters to functions within the C program, but chennels can only be accessed using cread and cwrite routines.

One undocumented point to note with Prospero Pascal is that a single astrix can not be included as a string character in for example comments, or this results in unexpected errors.

Although CS Tools could have been used to combine the OCCAM and the Pascal, as a majority of the code was written in occam, an occam harness seemed more appropriate. The way this was implemented was to create a 'shell' procedure in occam which does nothing except call the Pascal program and then an OCCAM harness was written which connects the shell to the computing surface. When the Pascal program and in turn the C procedures are called from occam, an array of channels is passes to it in the chanArray parameter.

One way to implement the harness is to include it in a Seperately Compiled (SC) program and run it in parallel with other SCs. An EXE program then is needed to allow keyboard, screen, and filing system access. A terminal emulator EXE program is provided with the Meiko Occam Programming System (OPS) to perform communication with the SC programs in the network and supply a run-time environment including connection of the program to the Computing Surface supervisor bus.

As using the terminal emulator EXE requires addition and running of a number of system routines in parallel with the shell in the created harness and it also means that there is no flexibility when the standard terminal emulator is used, an EXE was written for screen, keyboard and filing system access and also performing the trajectory planning stage of the control operation similar to that of the previous

subsection. Messages to and from the harness were then routed through transputer channels.

The SC program that performs the filtering is shown Appendix B. The Pascal program for carrying out the state-variable filter is called *tering* and after compilation etc., the library created is *tering.li8*, which is included in the occam by USE. For every trajectory point, three joint angles are input to the program through channel in1. Then the values of each angle are sent to the Pascal state-variable filter procedure and filtered values of the angle, velocity and acceleration is received and is output from the program on channel out1.

The function of various parameters are explained in the program by comments. It should be noted that arguments of the Pascal programs are standard and the value of heap space should not be less than what is indicated in the program or it halts. When compiling the Pascal program the amount of workspace, stacksize, and Freestore ought to be increased depending on the program requirements.

There are three C procedures which should be linked with the Pascal. First there is a dummy program that returns -1 and 1 for standard checks that are made when a program needs to be run under OPS.

```
/*  dummy.c */
#include <errno.h>
int fstat (int fd,char *buffer)
{
        errno = ENOTDIR;
        return(-1);
}
int link(char *name,char *name1)
{
        errno = EINVAL;
        return(-1);
}
```

```
int isatty(int fd)
{
       return(1);
}
```

Then it is the C program for reading from a channel:

```
/*  cfilti.c */
/*  To be used in con. with tering.pas */
#include <stdio.h>
#include <chanio.h>
#define chan_in 1
cfilti(float *a)
  {
     cread (occam_channel(chan_in), a, sizeof(float));
  }
```

And finally the C program for writing to a channel:

```
/*  cfilto.c */
/*  To be used in con. with tering.pas */
#include <stdio.h>
#include <chanio.h>
#define chan_out 0
cfilto(float a)
  {
     cwrite(occam_channel(chan_out), &a, sizeof(float));
  }
```

Part of the basic transputer architecture for the adaptive model-based control method with load mass estimation is shown in Figure 7.5.

Filtering of angles is performed using the processor FILTER, and the results are passed to the processor which implements the Load Mass Estimation (LME)

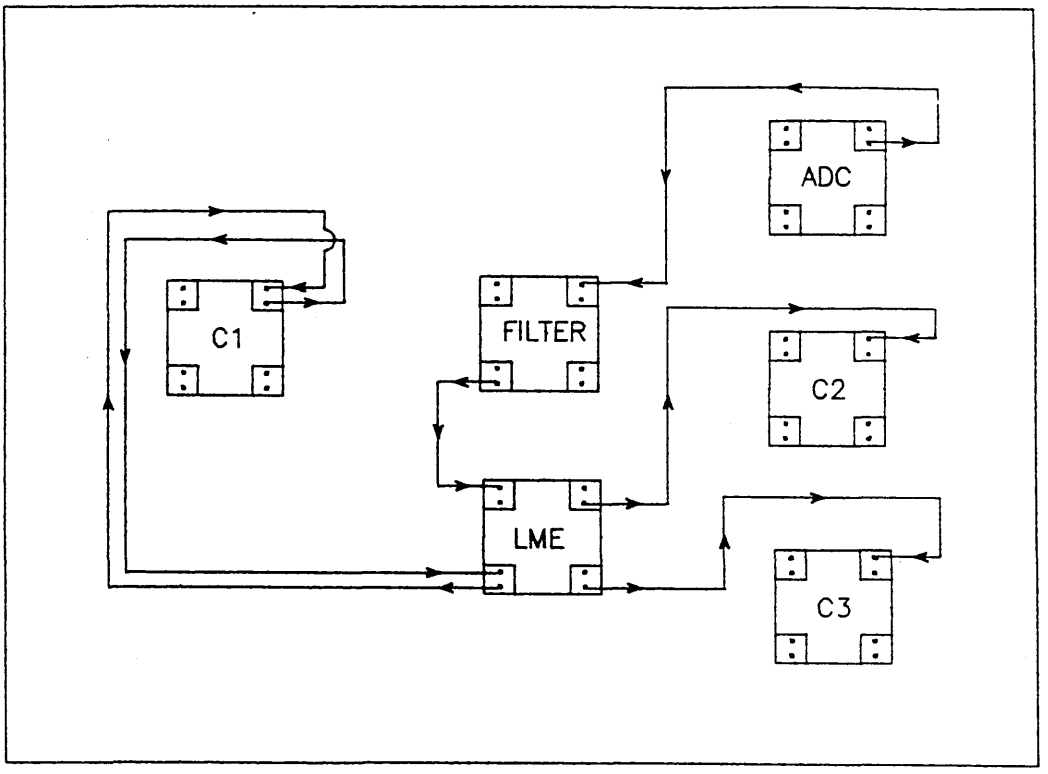


Figure 7.5: Part of the basic architecture for Adaptive MBC with LME

algorithm (discussed in chapter 4). In addition another filtering procedure for the value of the voltage is included in the controller C1, as the first joint or waist model is used for estimating the load mass, the filtered value of voltage for the first joint is passed to LME. After estimating the value of the load mass by LME, this value is sent to the three controller processors, so that the model can be updated. The LME process takes roughly the same length of time to calculate as the robot dynamic model. Although the structure is different, the computational times for the part of the equation that is suitable for allocation to separate processors is not a great percentage of the total. A higher speed up is achievable by allowing the linear in the parameter dynamic equations to be calculated on a separate Transputer or even three, similar to the dynamic model calculations of the computed torque scheme discussed, but this is not necessary for the case of equations for only three joints.

The rest of the set up is the same as for the computed torque method.

The discussion about extra processors to enable the use of the TO SLAVE connector for communication between the Meiko and the Parallax applies here too. The topology used is shown in Figure 7.6.

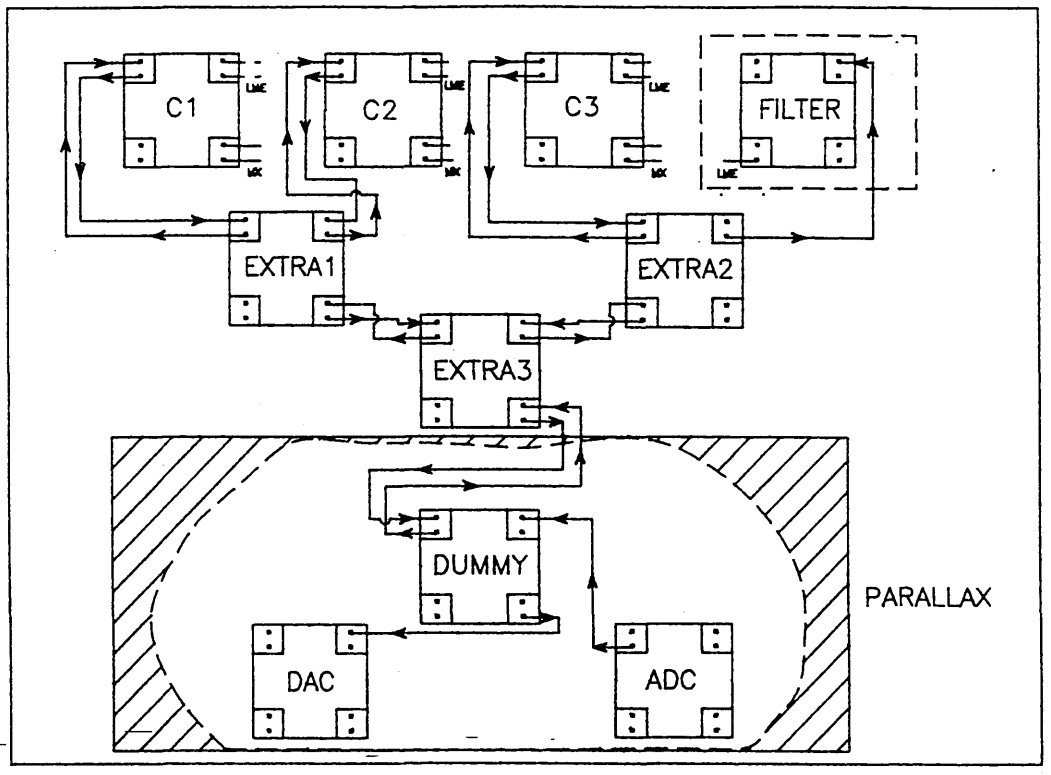


Figure 7.6: Transputer Topology when *TO SLAVE* is used

7.4.3 Model-Based VS Self-Tuning Control

Implementation of a VS self-tuning controller which will be combined with the model based control discussed above is explained here.

The proposed approach is a combination of a continuous time variable structure self tuning control, based on the idea of Variable Structure Systems and emulator based self tuning control of Gawthrop [24], and the model-based adaptive control with load mass estimation of the previous section. The method was discussed in chapter 6.

The objective of using the VSS approach is to achieve a performance that is robust with respect to disturbances and modelling errors, while it provides accurate

tracking. This results in rejection of Coulomb and viscous friction effects.

The method presents a control law that forces every trajectory to eventually come in contact with and remain on the intersection of switching surfaces in the joint space of the manipulator, with each surface being described by a linear combination of a joint position error and joint velocity.

Physical implementations of the method have given rise to *chattering*, which is highly undesirable, as it results in excitation of underdamped unmodeled dynamics. This problem is addressed in [102] where the discontinuous switching rule is replaced by a smooth version in which tracking precision is sacrificed for improved transient response. This was also discussed in chapter 6.

Koditschek [46] points out that the objective function in the Sliding Mode is explicitly time varying and as a result, natural control strategies cannot be applied with confidence so, instead the existing dynamics are cancelled or forced toward those desired by making systematic use of the rigid body model.

He shows that all trajectories originating away from the objective surface tend towards it asymptotically.

In the proposed method here, the non-linear terms in the dynamic equation of the robot manipulator

$$M(\theta)\ddot{\theta} + Q(\theta, \dot{\theta}) = \tau \tag{7.6}$$

where M is the $n \times n$ inertia matrix, Q is the vector of centrifugal, coriolis and gravitational forces, and τ is applied torques for rotational joints, can be cancelled, thereby decoupling the plant.

Once the plant is decoupled, a VSS self tuning is then applied to individual joints. In the VSS self-tuning control, implementation of which is in Pascal (CSTC) and is documented in [26], only measurements of the system output are required as opposed to measurements of the system states. The implementation of the switching surface is achieved by employing a self tuning emulator in place of unrealisable derivatives.

The basic idea behind CSTC is to unify a number of self-tuning algorithms and

with this in mind the major emphasis is on a continuous-time approach where controller design is carried out based on a continuous-time representation of the system.

A modified version of CSTC is used in the implementation of MBVSSTC. A brief description of CSTC's operation will be given, as well as pointing out the modifications that are made to allow three programs to be used for three robot joints to run in parallel and allow communication of the programs with the OCCAM sections of the controller code. And finally transputer implementation is described.

In CSTC transfer functions are represented as ratios of polynomials that are in turn defined by their degrees and corresponding coefficients. A large number of polynomial manipulation routines are provided which are used by various procedures.

As it is stated in [26], the implementation of the self-tuning controller is in the form of a self tuning emulator in a feedback loop. It consists of a tunable feedback controller and a parameter identification phase. Employment of an adaptive controller as compared to its non-adaptive counterpart means that the system parameters (coefficients of $A(s)$, $B(s)$ and T) need not be known in advance for achieving a desired performance. This is the main reason for replacing a fixed emulator by a self tuning one.

Emulators are used as is described in [24] for:

- Reducing the relative degree of the system.
- Reducing the number of non-minimum phase system zeros.
- Reducing system time delay.

In the first case Markov recursion is used for emulating linear combination of output derivatives.

To divide the Laplace transform of derivatives of impulse response of the system into *proper* and *improper* parts, Markov parameter representation is used. To

do this, the transfer function is expressed in terms of s^{-1} and the relative order, then repeated algebraic long division is used to express the transfer function as a polynomial in s^{-1}, the coefficients of which are the system Markov parameters. This process with some algebraic manipulation can be expressed in a recursive form.

In the second case, there is a need to detect common factors between the open-loop system denominator $A(s)$ and the notationally non-realisable part of the design parameter $Z(s)$, i.e. $Z^-(s)$. This is implemented using Euclid's algorithm to find the Greatest Common Divisor (GCD) of two polynomials.

Then the Diophantine equation of the form

$$P(s)C(s) = E_2(s)A(s)Z^+(s) + F_2(s)Z^-(s) \tag{7.7}$$

are solved for $E_2(s)$ and $F_2(s)$ by finding the GCD of $Z^-(s)$ and $A(s)Z^+(s)$ and deducing E_2 and F_2 by solving

$$1 = e(s)A(s)Z^+(s) + f(s)Z^-(s) \tag{7.8}$$

and using $e(s)$ and $f(s)$ to solve the equation recursively.

In the third case, Pade polynomial is used as an approximation to time delay.

The emulator based control law is either in explicit form

$$\hat{u} = \frac{1}{Q(s)}[R(s)\bar{w}(s) - \bar{\phi}^*(s)] \tag{7.9}$$

where $\frac{1}{Q(s)}$ is strictly proper and as can be seen the right hand side does not depend on the control signal, and is implemented by feeding the error signal inside the square brackets into a filter implementing $\frac{1}{Q(s)}$, or it is in the implicit form

$$\bar{\phi}^*(s) + Q(s)\hat{u}(s) - R\bar{w}(s) = 0 \tag{7.10}$$

where the numerator and the denominator of $\frac{1}{Q(s)}$ have the same degree. In this case the value of the control signal is put equal to zero once and then unity

$$\hat{u} = 0 \quad \Rightarrow \quad \kappa_0 = \bar{\phi}_0^*(s) - R\bar{w}(s) \tag{7.11}$$

$$\hat{u} = 1 \quad \Rightarrow \quad \kappa_1 = \bar{\phi}_1^*(s) - R\bar{w}(s) \tag{7.12}$$

If we draw κ against $\hat{u}$, and try to find the value of $\hat{u}$ when κ is zero, similar triangles can be used to obtain the following equality:

$$\frac{-\kappa_0}{\hat{u}} = \frac{-\kappa_0 + \kappa_1}{1}$$

Now replacing from 7.11 and 7.12 for κ_0 and κ_1 the value of $\hat{u}$ can be found from

$$\hat{u} = \frac{w - \bar{\phi}_0^*(s)}{\bar{\phi}_1^*(s) - \bar{\phi}_0^*(s)}$$

To avoid saturation, the control action of the feedback controller is limited by including a non-linearity between the control signal and the $\frac{1}{Q(s)}$ transfer function, hence an important factor emphasised in [24] of emulator operating on the control signal before the saturation as opposed to the signal before saturation is taken account of.

It is possible to implement variable structure control by setting the control signal to a minimum or a maximum whichever is closest to the value of the signal.

For self-tuning control the emulator equation needs to be represented in the linear-in-the parameters form

$$\phi^*(t) = \underline{X}_e^T(t)\underline{\Theta}_e$$

where in the Laplace transform

$$\bar{\underline{X}}_e(s) = \begin{bmatrix} \bar{\underline{X}}_u(s) \\ \bar{\underline{X}}_y(s) \\ \bar{\underline{X}}_i(s) \end{bmatrix} \quad ; \quad \bar{\Theta}_e = \begin{bmatrix} \bar{\Theta}_u \\ \bar{\Theta}_y \\ \bar{\Theta}_i \end{bmatrix}$$

are data and parameter vectors respectively.

This is not explicitly implemented, instead, after generating the data vectors, inner product of the data vectors are formed from the coefficients of respective polynomials and in this way the emulator output is found.

The emulator output value is then estimated and this replaces the emulator output.

For parameter identification, CSTC uses the discrete-time algorithm for recursive least squares estimation which is regarded as an approximation to the continuous-time algorithm discussed in [24]. It should be mentioned that with this algorithm, continuous time parameters are estimated.

In [24] it is shown that discrete-time estimation of continuous-time parameters does not introduce sampling errors which means estimation sample rate and controller sample rate can be independent from one another.

The particular Self Tuning Control (STC) scheme used for the purpose of controlling the robot joint movements is the *Implicit control-weighted model-reference with off-line design* as explained in [24]. It is shown in [26] that the control signal is reduced when control weighting is not zero, and although model following is not exact, but no steady-state offset is ensured by using the $Q(s)$ design rule.

A detuned (control-weighted) version of this algorithm, where the control weighting is finite is used for a robust control, with a moderate penalty that the algorithm will no longer be quite as exact.

The closed-loop transfer function generating the system input when there is no time delay, control weighting or setpoint filter is

$$\bar{u}(s) = \frac{Z^+(s)A(s)}{P(s)B^+(s)}(\bar{w}(s) - \frac{F(s)}{A(s)Z^+(s)}\bar{v}(s))$$

For model-reference control $B^-(s) = Z^-(s) = 1$. The control signal is stable if $B(s)$ is stable.

When we include a set point filter (i.e. $R(s) \neq 1$), $R(s)$ has no effect on the feedback loop, but then it can affect the set point response the same way that $\frac{1}{P(s)}$ can (i.e. it can be set equal to the desired model). However $P(s)$ alters the disturbance response and close-loop sensitivity. The requirement of a unity steady-state system gain from setpoint to output imposes $R(s) = 1$

In the Implicit STC, the emulator parameters are directly tuned and the design phase is avoided. The off-line design means that the emulator design parameter,

the control weighting and the set point filter are chosen off-line before the self-tuning starts. There are two phases associated with the off-line design, one the a-priori design phase and the other on line tuning phase.

The steps to implement the off-line phase are to choose

- Emulator polynomials $P(s)$, $Z^+(s)$, $Z^-(s)$, and $C(s)$
- Control Weighting filter $Q(s)$
- Set point filter $R(s)$
- System order
- Realisability filter $\Lambda(s)$ [4], typically $e^{-sT}\frac{Z(s)}{P(s)}$

Depending on the type of approach, the choice of emulator polynomials and the realisability filter might not be as above. For example if the Boolean variable *UsingLambda* is set to *TRUE*, then $\Lambda(s)$ is chosen to be $e^{-sT}\frac{Z(s)}{P(s)}$, otherwise it is equal to unity.

In the on-line phase if using lambda filter :

- The quantity $\phi_\Lambda(t)$ is generated.
- The control signal $u(t)$ and the system output $y(t)$ are filtered by the realisability filter.
- Using the filtered signals of system output and control signal, and the following differential equation [5]

$$\frac{d}{dt}\underline{X^c} = \underline{A}\underline{X^c} + \underline{U}u$$

also discussed in the previous subsection , generate the emulator data vector $\underline{X}_\Lambda(t)$.

[4]Many emulators $\phi(t)$ which are used in the continuous time approach to self tuning are not realisable and, a *realisability filter* $\Lambda(s)$ is appended so as to obtain a realisable signal $\phi_\Lambda(t)$

[5]The controllable state space representation of the filter is obtained and with $\underline{X}$ as input, the first state is the output, second the first derivative etc.

- The emulator parameter estimate vector $\underline{\hat{\theta}}e(t)$ is updated using discrete least squares from the linear in the parameters model.

- The emulator signal $\phi(t)$ is generated using

$$\hat{\phi}(t) = \underline{X}_e^T(t)\underline{\hat{\theta}}_e(t)$$

- The control signal in the Laplace-transform terms is generated

$$\hat{u}(s) = \frac{1}{Q(s)}[R(s)\bar{w}(s) - \hat{\phi}(s)]$$

In the step before the last, emulator signal is generated using a modified control signal which is the actual signal the plant receives and not the saturated value. This is due to a procedure that limits the value between a maximum and a minimum and also allows implementation of variable structure control.
The emulated signal also updates the corresponding emulator states.
The value of the emulator design parameters chosen as well as other parameters such as the initial values $A(s)$ and $B(s)$ etc. can be seen in Appendix C.

Modifications

A number of procedures were not included such as Input, output, Polynomial output, WriteLnData, WriteData etc. for simplicity; as they are not used in the transputer program. These procedures are basically used for prompting for and inputting the value of various parameters such as the ones shown in Appendix C in the initial interaction with CSTC before the self-tuning starts. This requires access to the keyboard, screen, and the filing system which if was to be done for three joints (three separate programs), the data routing can get a bit complicated, not to mention the time taken to input three lots of parameters some of which hardly need to be changed anyway. In addition it was decided beforehand as to what type of self-tuning algorithm was to be used. Therefore the initialisation stage was performed explicitly within the program.

In addition the only data which was of interest was the trajectory error data which needed to be recorded and as a result the data recording in files of the emulator parameters, model output etc. that is possible in CSTC were not performed.
A procedures *GetData* and *PutData* in CSTC were modified to *ConveyData* and *DeliverData* that are shown in Appendix C. The procedures take the values of the setpoint, system input, and system output from an occam program using a C procedure crec, and after the self-tuning is performed deliver the new input to the occam program using the C procedure csend, so that it could be combined with the model-based input and then be passed to the DAC and in turn to the motor amplifiers.
Besides other modifications which were made to exclude writing to the screen or a file or reading from the keyboard and modifications which exclude unnecessary steps such as the ones involving External data etc. or functions and procedures such as procedure *HighGainControl* which were not necessary for the particular self-tuning implementation in mind for clarity and simplicity, the following modifications were also made: Limiting the chapters only to two (6, 7) and not including the function for selection of appropriate chapter, and repeating the modified procedure *OneTimeStep* as many times as there were trajectory points specified by the occam program so that self-tuning is performed at each sample for the specified trajectory until the run is finished based on one set of initial conditions. The value of real variable LastTime should be twice the value of SampleInterval so that the number of runs would match the number of setpoints provided.
The occam harness for the Pascal was similar to the one described for the adaptive model-based control of the previous subsection. However the value of fast stack arrays and the heap space had to be increased to 1M bytes in the occam program and when compiling the same increase applied to workspace, stacksize, and the freestore.

Transputer architecture

The transputer architecture for implementing the MBVSST algorithm is the same as the one used for the adaptive model-based control of the previous subsection. The variations in processes assigned to processors are: each controller processor C1, C2, C3, in addition to dynamic equation calculations carry out the self-tuning control for individual joints on one processor per joint basis. Measuring the time taken to carry out the whole process shows that by doing this the target sampling frequency is still achievable. In fact without introducing any delay, the sampling frequency was measured to be 125 HZ. It should be noted that for the adaptive model-based control of the previous section the the sampling frequency is 167 HZ and for the computed torque it is 217 HZ.

Another variation is that C1 sends the value of the voltage to LME and the filtering of this voltage is performed on LME processor.

Further speed up can be achieved by dividing the self tuning to Identification and Control and allowing each stage to be performed in parallel on separate transputers.

Simple graph representation can be used, representing data dependencies of various operations, allowing for input-output, communications and path sequences of subtasks. From this, complexity measures (both time and communication complexity plus number of processors) can be analysed and as a result a suitable topology can be designed. In addition, speed up and efficiency considerations should be looked at and where possible subtasks should be allocated to the most lightly loaded processor to increase efficiency.

In the next section, the transputer network topologies and the programs that were discussed above are used to carry out a number of simulations and experiments, the results of these are presented and discussed.

7.5 Results and discussion

Prior to the actual implementation of the control algorithms on the robot, some simulations were performed to assess the suitability of these schemes as well as verifying how close the simulation results, which are based on the model of the robot, are to the experimental outcome using the robot itself.

One point should be made at this stage: the motor speeds of the MA3000 robot are slow. For the case of the waist and shoulder, the maximum speed is quoted as being 22.5 deg/sec and for the elbow the value is 45 deg/sec maximum.

All the experiments were started when the shoulder and elbow were at a vertically upright position which corresponds to absolute 140 degrees. Movement of each joint was from this initial position through 30 degrees according to the polynomial trajectory described earlier; shown in figure 7.3, or application of a square wave setpoint with time period of 4.9 s. Only the deviations of the measured trajectories from the desired ones (trajectory errors) are shown for clarity.

The input voltages to the joint motors at each sample could vary between 88.5V and -88.5V. These correspond to the preamplified values output from the DAC of 10V and -10V respectively. As the voltages calculated from the model-based algorithms that are presented can exceed the maximum value or are at times below the minimum or even are below the threshold voltage that can actually result in motion, the signals were restricted to either maximum or minimum.

An additional consideration is that, when the model-based controllers are combined with variable structure self-tuners, the limitation of VS self tuners to have only maximum and minimum does not lead to domination of one signal over the other and both signals will have equal effects.

To perform the simulations in line with the experiments, the same conditions as for the experiments were applied.

For each controller simulation or experiment, in addition to the trajectory error profiles for each joint; individual control signals over the time duration will be

graphically shown and in the adaptive cases, the graph of estimated values of the load mass against time will be displayed.
All the robot model parameters for the controllers are based on the model of the MA3000 robot developed in chapter 3 and the values are presented there.

7.5.1 Simulations

Using the same transputer topologies as for the real experimental implementations, the processes and the corresponding processors for Analog to Digital and Digital to Analog conversions (the communication routes between the circuit that drives the robot and the network of transputers) were replaced by a robot simulator. The simulator was itself implemented in OCCAM on the transputer network. In the robot simulator, the model of the robot is used to calculate the resulting joint accelerations for given input voltages at each sample. Then the accelerations are integrated once to find the joint velocities and twice to find the joint angles. Values of joint velocities and angles are are then used in the next sample. The process includes a matrix inversion of the 3×3 mass matrix at each sample.

Computed Torque

The first simulation performed involved Computed Torque control with a square wave set point. The torques were computed on the basis of a correct system model.
The diagonal elements of the gains K_p and K_v of the characteristic equation which describes the suppression of errors in the control system:

$$\ddot{e} + K_v\dot{e} + K_p e = 0$$

were experimentally adjusted for the critically damping case to be

$$K_p = 1600 \qquad K_v = 80$$

The trajectory errors for the waist, shoulder, and elbow can be seen in figures 7.7 and 7.8.

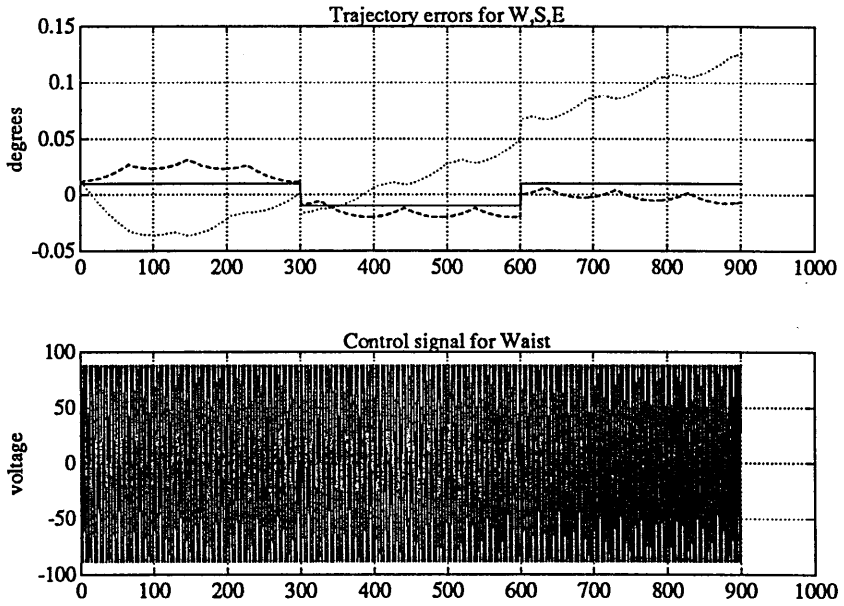


Figure 7.7: Computed Torque with SW setpoints

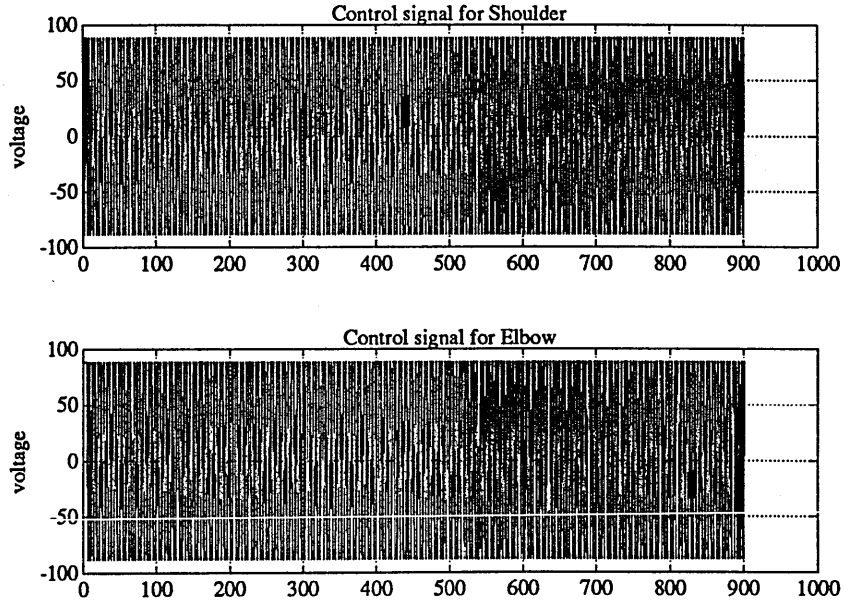


Figure 7.8: Computed Torque with SW setpoints

As can be seen the errors are quite acceptable.

Next simulation was again the computed torque, but this time the setpoints were defined according to a polynomial trajectory.

The trajectory errors and the control signals are shown in figures 7.9 and 7.10.

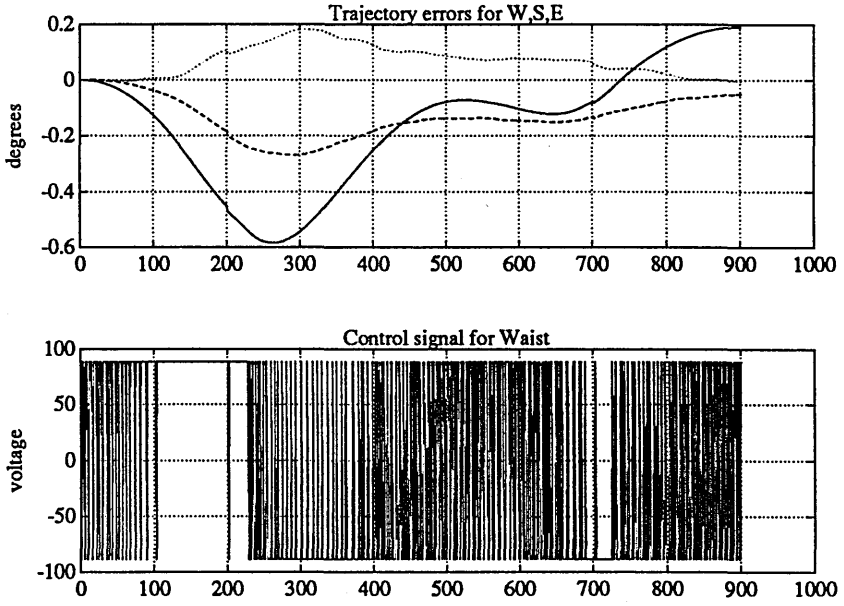


Figure 7.9: Computed Torque with Polynomial Traj.

The errors are slightly higher than the previous case but still quite reasonable. There is less switching in the control signals in comparison to the previous case.

Model-Based Adaptive Control with LME

The model-based adaptive controller with load mass estimation was simulated next. The initial estimate for the load mass was chosen to be 1.5kg (according to the results presented in chapter 4 regarding the load mass estimation method, the discrepancy between the initial and the actual value should not be too large). Firstly square wave setpoints were applied, the results of which can be seen in figures 7.11, 7.12, and 7.13 and then polynomial trajectory setpoints, results shown in figures 7.14, 7.15, and 7.16. In the latter case improvements in reducing the trajectory error for the waist is noticeable.

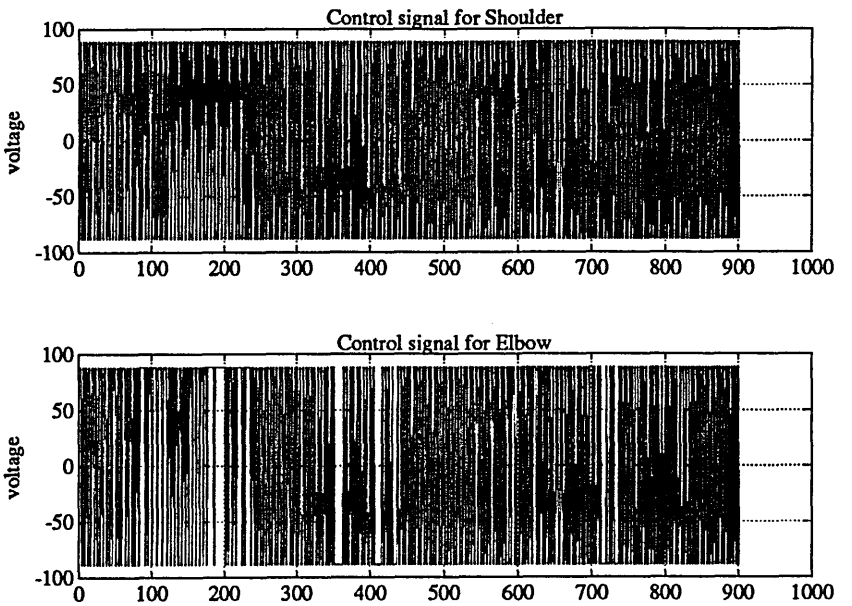


Figure 7.10: Computed Torque with Polynomial Traj.

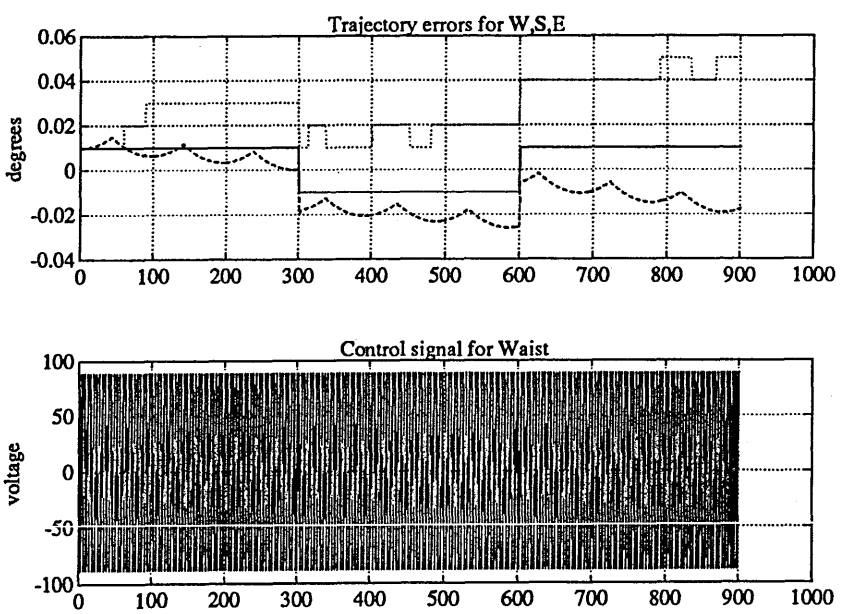


Figure 7.11: Adaptive model-based with SW setpoints

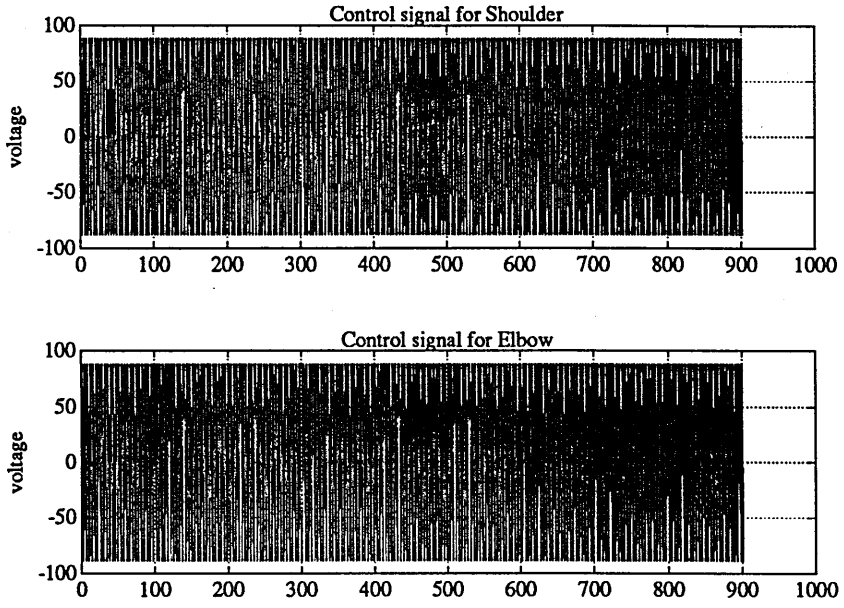


Figure 7.12: Adaptive model-based with SW setpoints

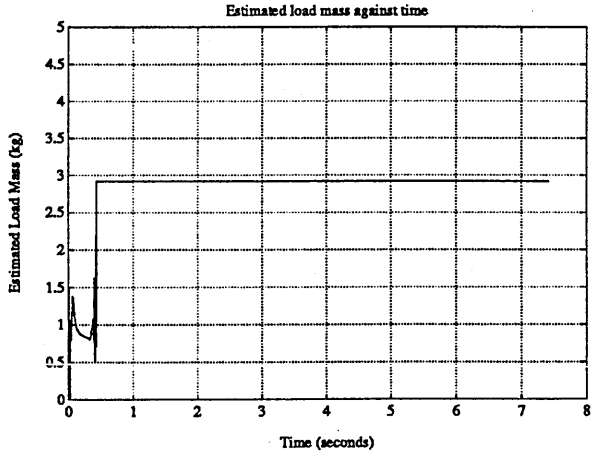


Figure 7.13: Adaptive model-based with SW setpoints

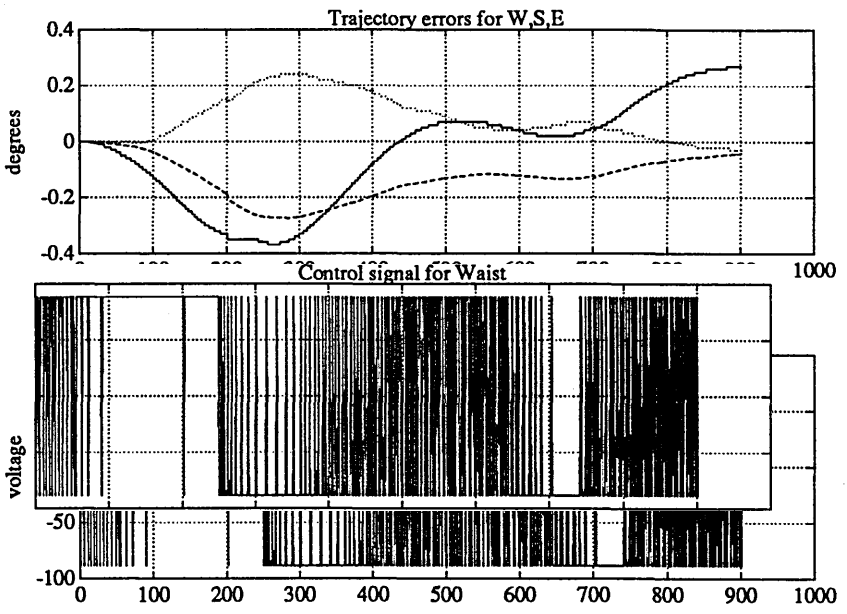


Figure 7.14: Adaptive model-based with Polynomial Traj.

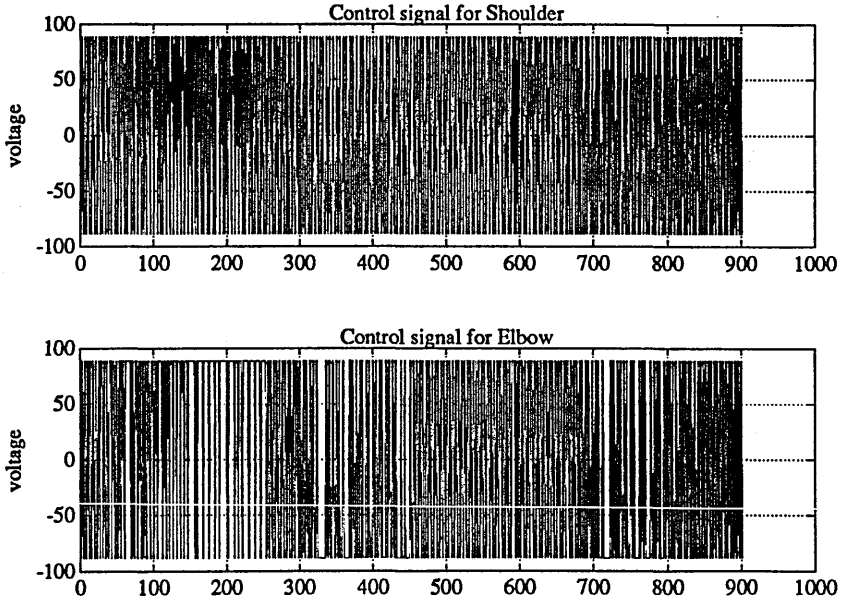


Figure 7.15: Adaptive model-based with Polynomial Traj.

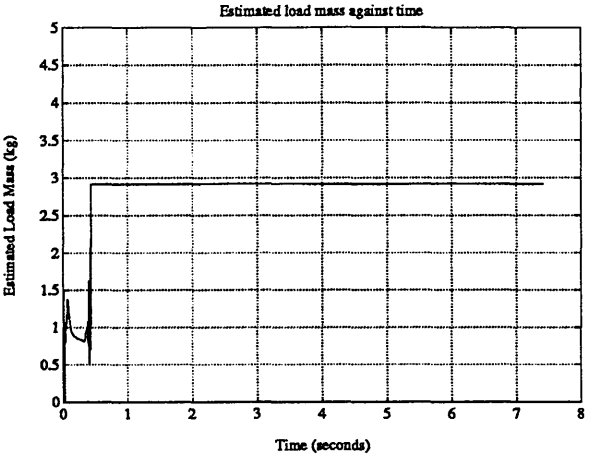


Figure 7.16: Adaptive model-based with Polynomial Traj.

Model-Based VS Self-Tuner

Using the adaptive model-based controller with load mass estimation to linearise and decouple the system and then applying a variable structure self-tuner was a subject of discussion in the previous chapter. This combined controller was simulated and the results can be seen in figures 7.17, 7.18, and 7.19. The particular type of VSST used was implicit model reference with control weighting $Q = 0$. The trajectory errors for shoulder and elbow are reduced considerably, where the control signal switching for the model-based adaptive part is reduced.

The same controller was simulated this time with square wave setpoints. The results are shown in figures 7.20, 7.21, and 7.22. The errors compared to when only a model-based adaptive controller was used with square wave set points are considerably reduced.

When the control weighted version of the above self-tuner was applied with a polynomial trajectory and $Q(s) = \frac{s}{s+1}$, the errors increased as can be seen from the results shown in figure 7.23.

Addition of a setpoint filter $R(s) = \frac{0.5s+1}{s^2+\sqrt{2}s+1}$ did not improve the situation as can be seen from figure 7.26

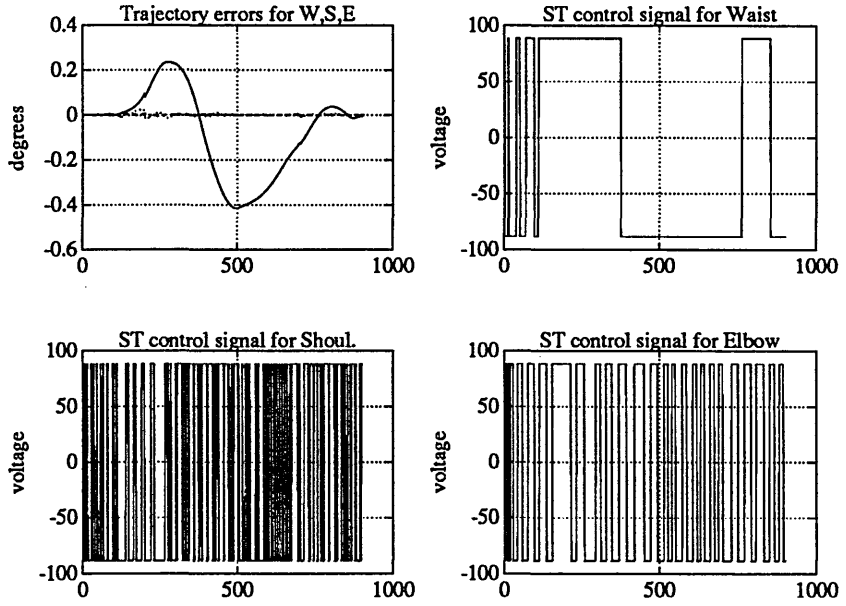


Figure 7.17: Model-Based VSST with Polynomial Traj.

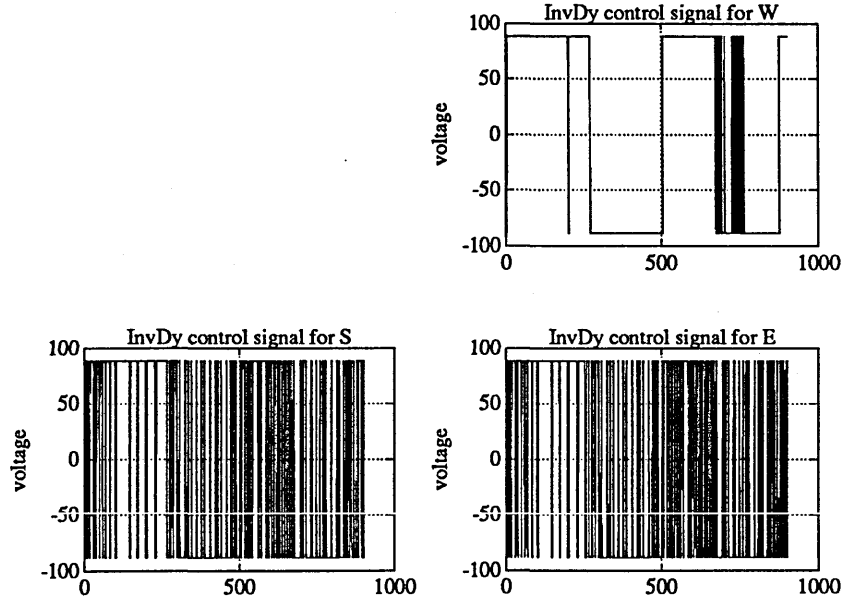


Figure 7.18: Model-Based VSST with Polynomial Traj.

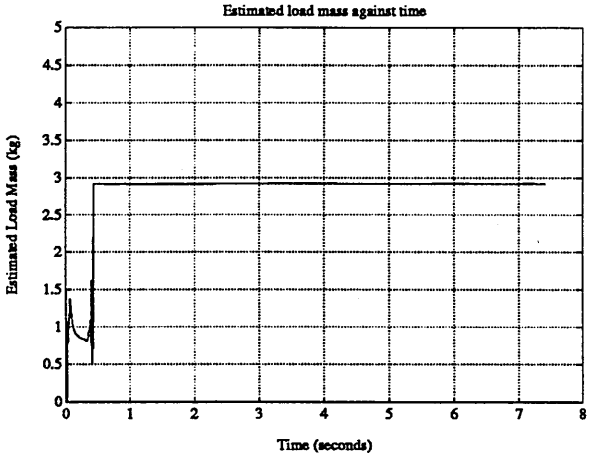


Figure 7.19: Model-Based VSST with Polynomial Traj.

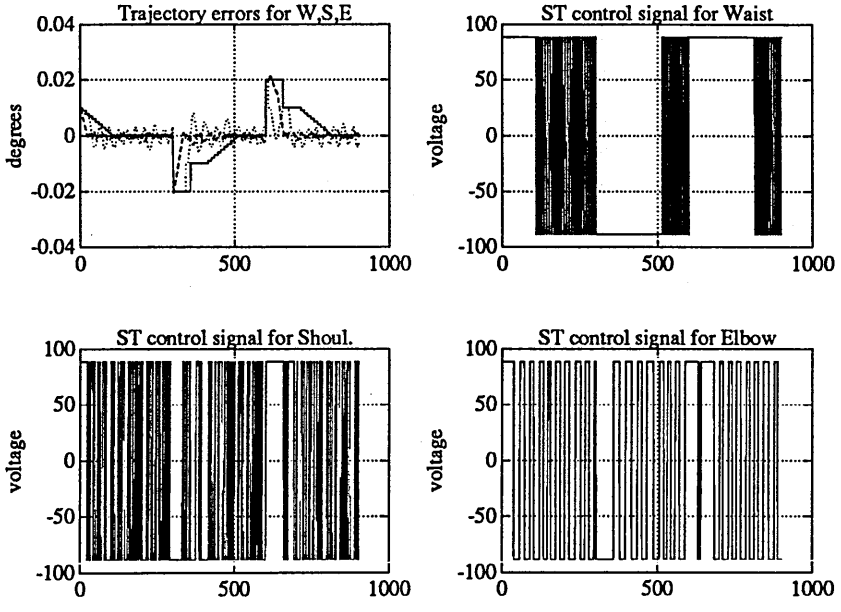


Figure 7.20: Model-Based VSST with SW setpoints

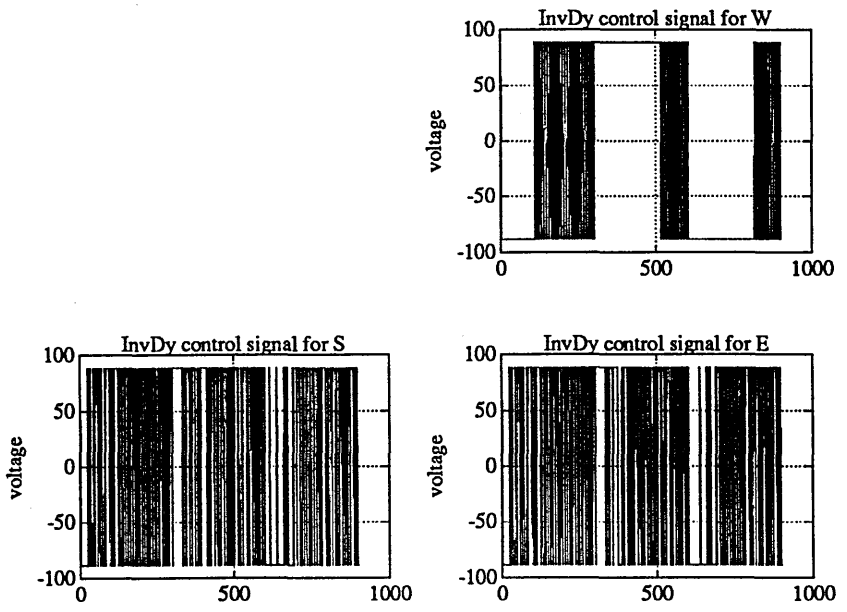


Figure 7.21: Model-Based VSST with SW setpoints

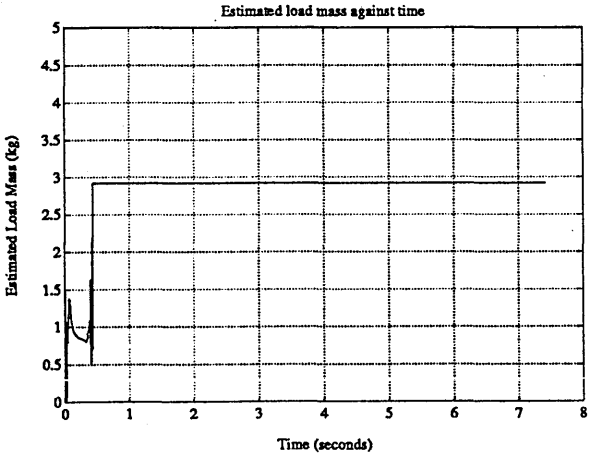


Figure 7.22: Model-Based VSST with SW setpoints

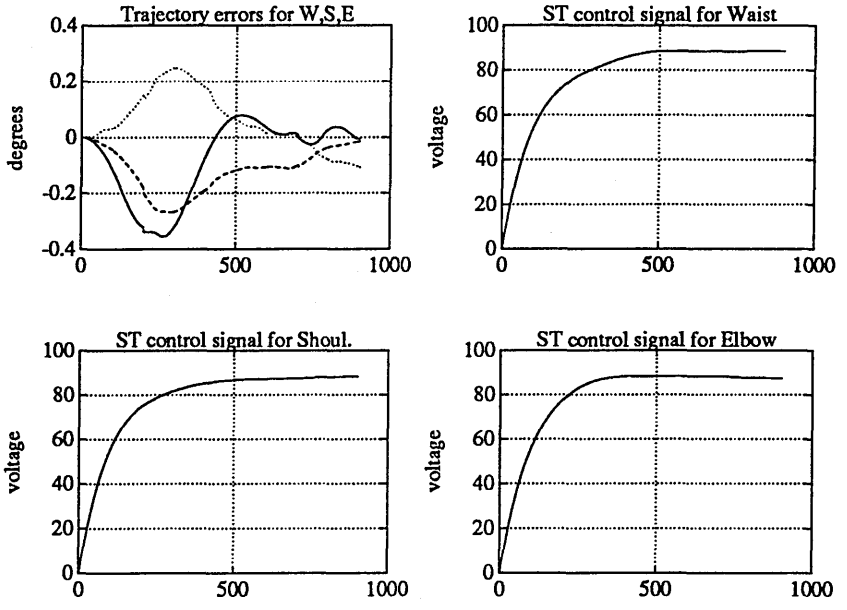


Figure 7.23: Model-Based Cont. Weigh. VSST with Polynomial Traj.

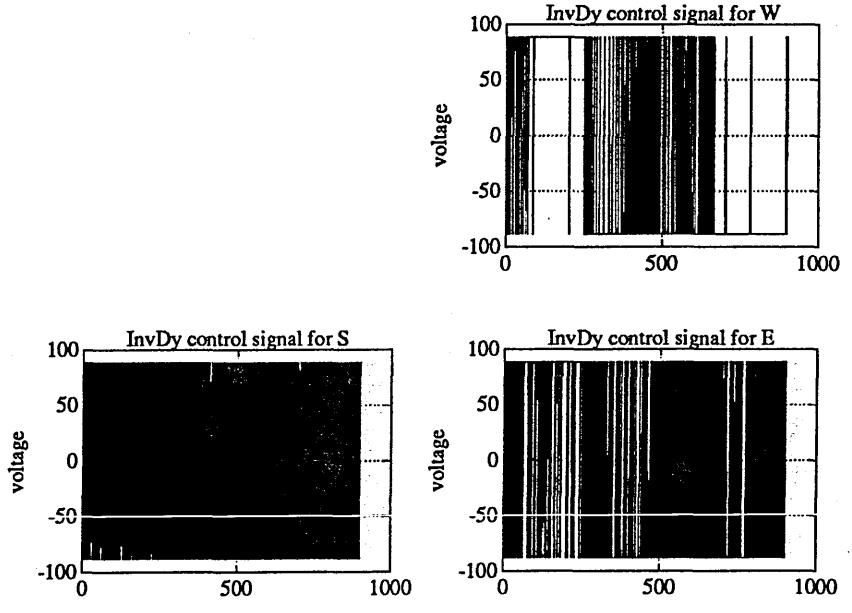


Figure 7.24: Model-Based Cont. Weigh. VSST with Polynomial Traj.

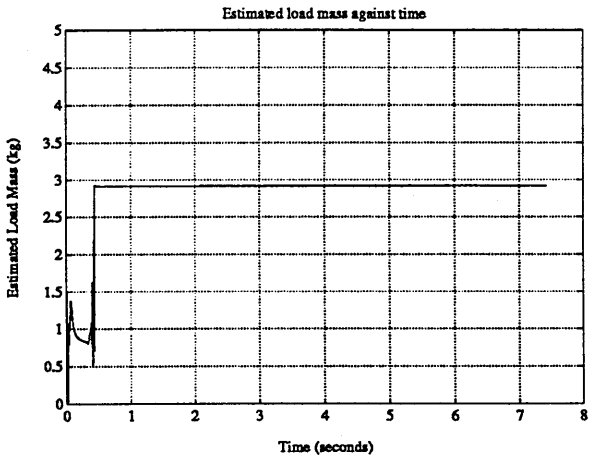


Figure 7.25: Model-Based Cont. Weigh. VSST with Polynomial Traj.

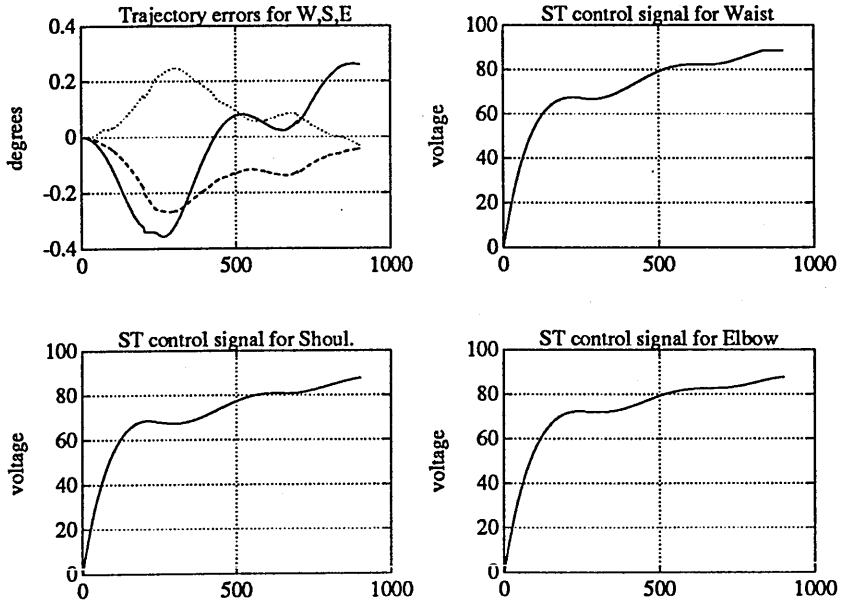


Figure 7.26: MB Cont. Weigh. with setpoint filter VSST (Poly. Traj.)

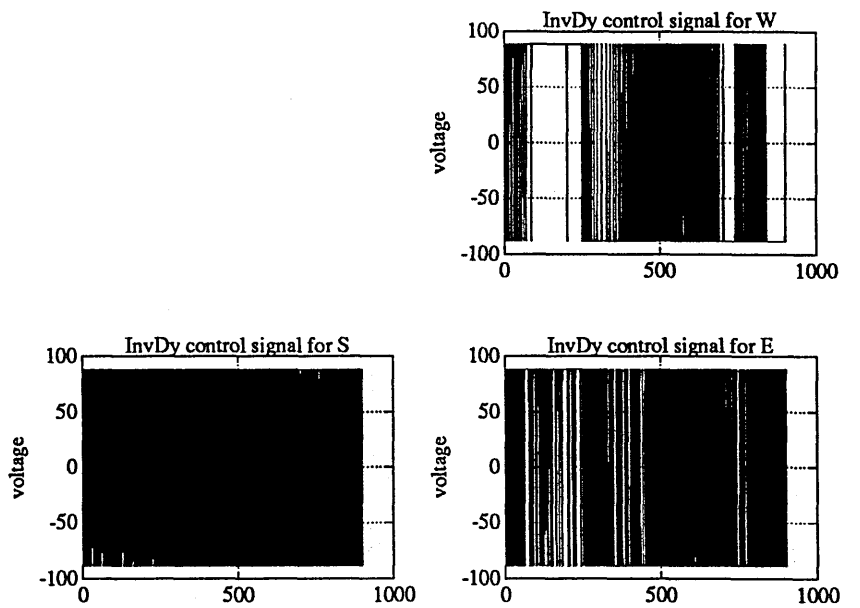


Figure 7.27: MB Cont. Weigh. with setpoint F VSST with Poly. Traj.

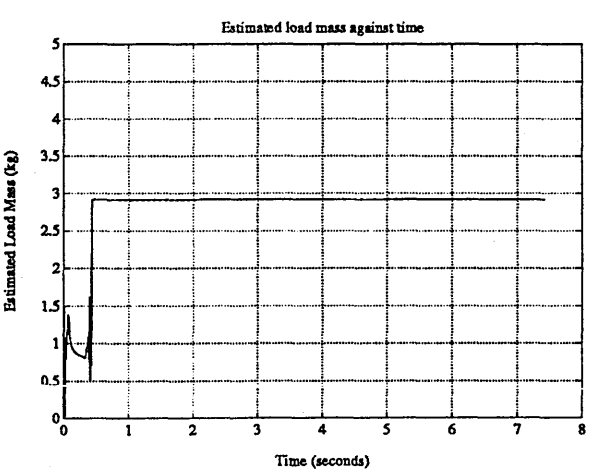


Figure 7.28: Model-Based Cont. Weigh. VSST with Polynomial Traj.

7.5.2 Controller experiments

The sampling frequency achievable to implement the computed torque on the transputer network topology discussed earlier is 215 HZ and the values for the model based adaptive controller with load mass estimation and the model based variable structure self-tuner are 167 HZ and 121 HZ respectively.

To keep all the conditions for the three controllers the same, delays were introduced so that the lowest sampling frequency i.e. 121 was applied for all controllers. The overall duration for each control scheme to go through 900 points according to the polynomial trajectory discussed earlier was 7.4s.

Computed Torque

Firstly the computed torque scheme was implemented. The trajectory errors and the corresponding control signals for all the three joints are shown in figures 7.29 and 7.30.

The errors are certainly larger compared to the simulated results, but still quite reasonable since a polynomial trajectory setpoint is applied.

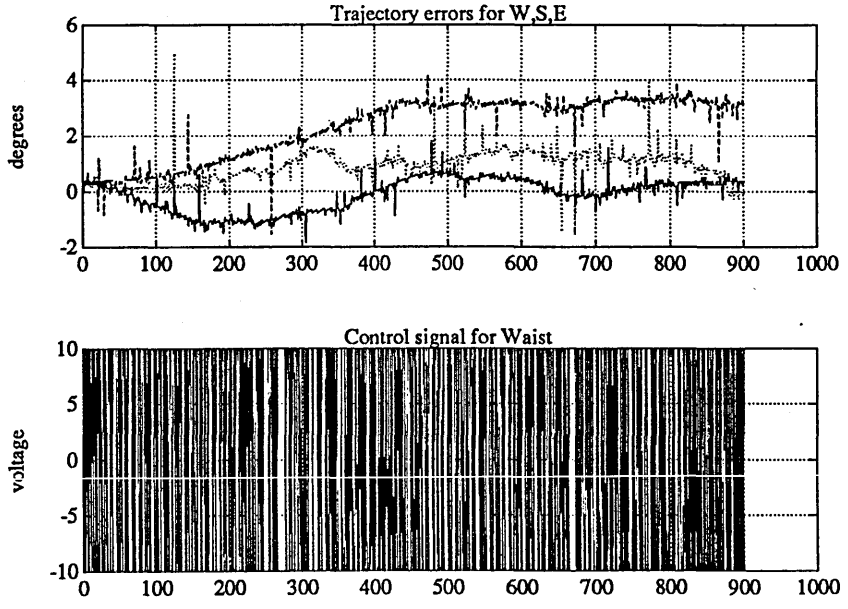


Figure 7.29: Computed Torque (real implementation)

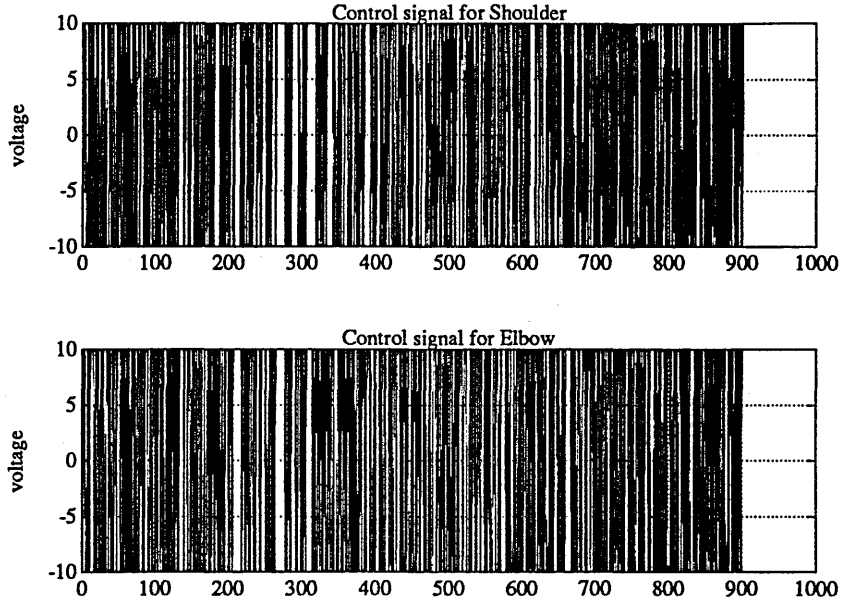


Figure 7.30: Computed Torque real implementation

Model-Based Adaptive Control with LME

When the model-based adaptive controller with load mass estimation was applied the errors for the waist were reduced which is consistent with the simulated results. This can be seen in figure 7.31.

The graph of estimated load mass against time is shown in figure 7.33.

It can be that the effect of the load mass on the waist is higher compared with the other joints and as a more accurate value of the load mass is included in the model, the model gets closer to the real representation of the actual robot (note that the gripper is part of the load in this case).

Furthermore as the load estimation routine only uses the dynamic equations of the waist (as its feasibility was discussed in chapter 4), this could also play a part in the outcome of the results.

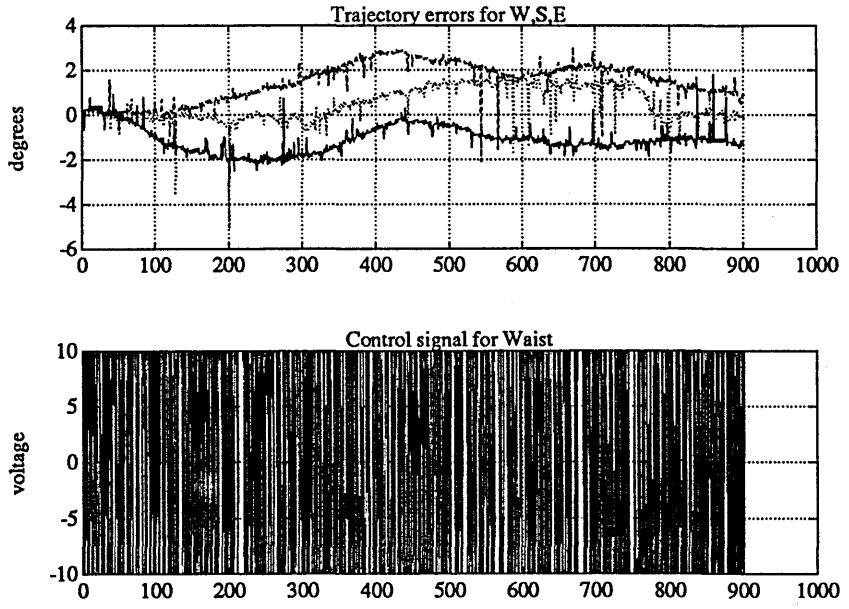


Figure 7.31: Model-Based Adaptive Control with LME (real implementation)

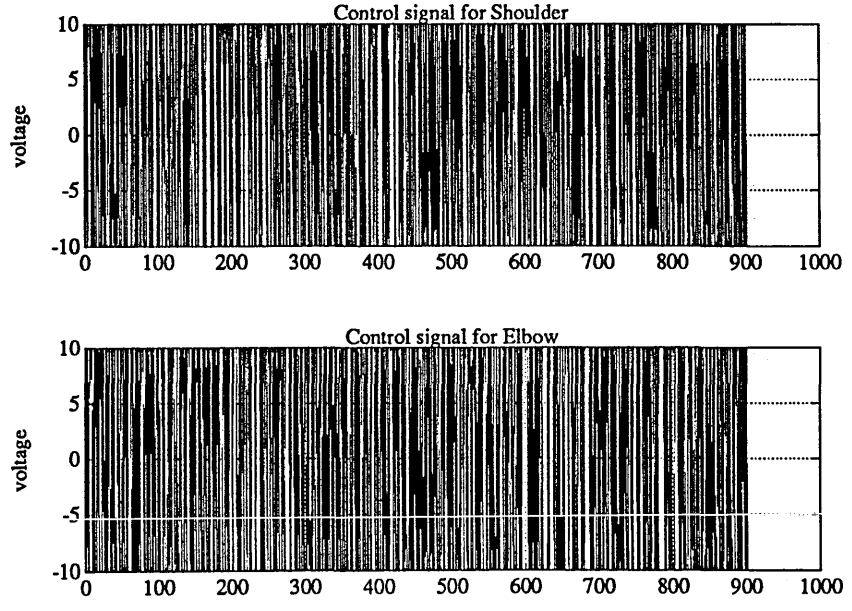


Figure 7.32: Model-Based Adaptive Control with LME (real implementation)

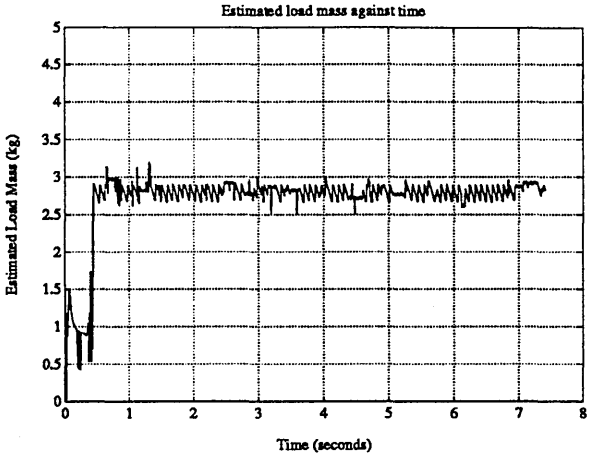


Figure 7.33: Model-Based Adaptive Control with LME (real implementation)

Model-Based VS self-tuning control

From simulation results and by carrying out a number of experiments, the most suitable (in terms of producing the least trajectory errors) type of variable structure self-tuner to combine with a model-based controller was found to be a model reference self-tuner with no control weighting.

When the self-tuner was applied by itself without linearising or decoupling the system first, the overall results were not acceptable due to trajectory errors produced being large. When the self-tuner was applied to individual joints with all other joints not moving, good performance was only exhibited in the case of the waist as can be seen from figure 7.34.

Combination of the VS self-tuner and the model-based adaptive control with load mass estimation produced the best results of all.

The trajectory errors are quite small as can be seen from figure 7.35 and in comparison with the other controllers, the results are much more improved.

The load mass estimates can be seen in figure 7.37.

7.5.3 Conclusions

The conclusions drawn from the simulation and experimental results and also from what has been discussed so far can be summerised as follows:

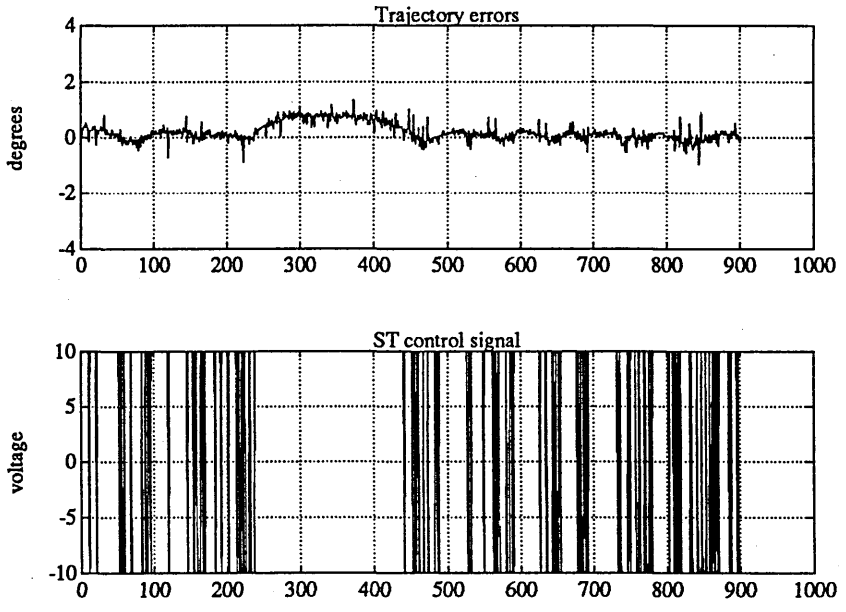


Figure 7.34: VS Self-Tuning applied to waist (real implementation)

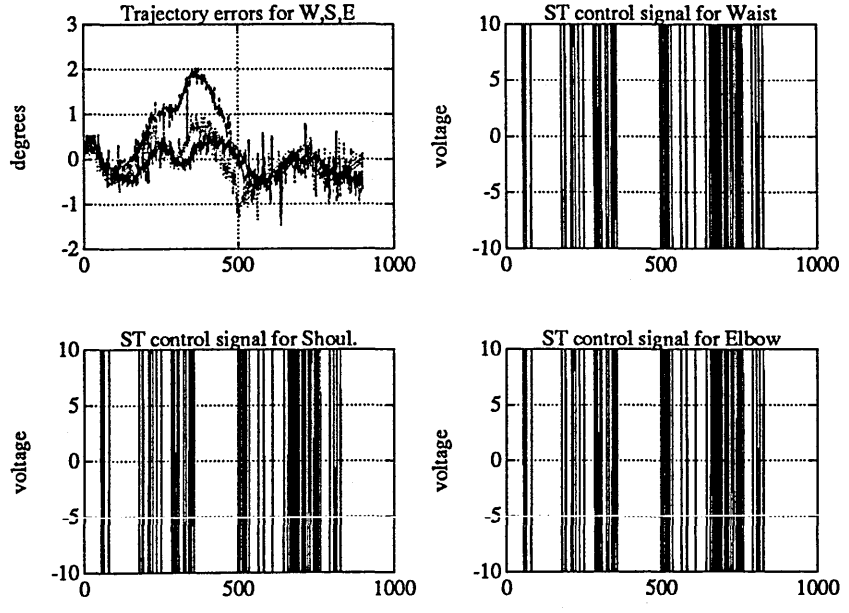


Figure 7.35: Model-Based VS Self-Tuner with LME (real implementation)

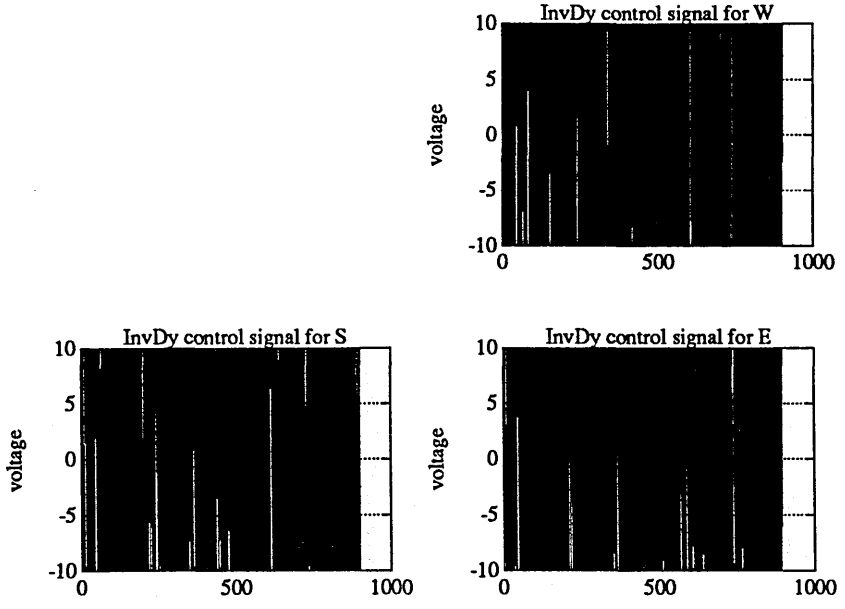


Figure 7.36: Model-Based VS Self-Tuner with LME (real implementation)

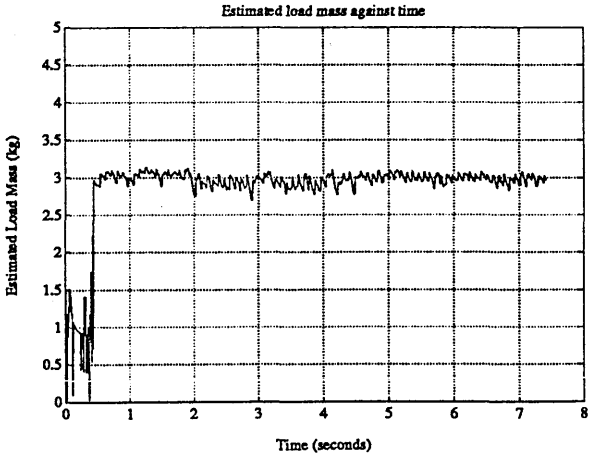


Figure 7.37: Model-Based VS Self-Tuner with LME (real implementation)

- There is a need for using transputers and parallel processing to implement the type of controllers that were used, if required sampling frequencies is to be achieved.
- Experiments are needed in addition to simulations which are purely based on a model that might not be accurate enough.
- Experimental results help verify the validity of a developed model if they are consistent with simulation results.
- Model-based control is certainly effective if a good model is available.
- The load mass estimation method introduced in chapter 4, incorporated into a model based control scheme results in a more accurate model and hence the model can be used more effectively to decouple and linearise the system.
- Combining a model based controller with a suitable variable structure self-tuner results in an excellent controller as far as trajectory following is concerned.

Chapter 8

Conclusions

During the course of the research, a number of hypotheses and concepts regarding model-based control of robot manipulators for gross motion trajectory tracking have been brought under scrutiny and most of the relevant issues in the area have been considered.

The following are the issues considered:

- Kinematics and dynamics of manipulators.
- Modelling of manipulators including the actuation systems.
- Obtaining parameters of the overall model, using experimental data from the robot and using a CAD-based approach.
- Model validation.
- Load mass estimation.
- Computing requirements for real-time control implementation and choice of parallel processing as a suitable option to meet these requirements.
- Requirements of a robot control programming system.
- Hardware and software interfaces needed for experimental evaluation.

- Comparison of various manipulator control methods.
- Realistic simulation and experimental evaluation of the control strategies.

Experimental evaluations provided some valuable insights for better understanding of various aspects of robot control as well as giving credibility to the conclusions drawn. For example

- The ability of model-based controllers to produce a reasonable trajectory tracking accuracy, provided that the robot dynamic model is sufficiently accurate.
- The importance of cancelling coriolis and centrifugal forces, even at low speeds, to reduce tracking errors.
- The improvements that can be achieved, as far as trajectory tracking is concerned, by estimating the mass of the payload carried by the manipulator gripper.
- The unsuitability of employing a linear self tuning controller for each joint of the manipulator without decoupling and linearising the system first (especially for trajectory tracking applications).
- The trajectory error reduction, resulting from addition of suitable self-tuners to decoupling and linearising model-based controllers, to reduce the effect of having an inexact dynamic model.
- The suitability of using a variable structure self-tuner which uses output feedback with observers as opposed to full state feedback in the self-tuning part of the control law.
- The increase in the trajectory errors, as a result of using a detuned version of the self-tuner (nonzero control weighting), although its robustness properties are desirable.

Employment of model-based algorithms for robot control is essentially to compensate the nonlinear dynamics of the manipulator at the initial stages especially at high speeds. Having this in mind, the limiting factor in the experimental evaluations was the relatively low speeds of the MA3000 robot which was used. However this limitation served to provide a suitable ground for showing that compensation for coriolis and centrifugal forces and joint interactions led to improved trajectory tracking accuracy even at low speeds.

To achieve better tracking accuracy, some have embarked on development and use of direct-drive robots to overcome the domination of the motor and drive system dynamics over joint dynamics and to avoid the friction of the gear train. However there are two points to note: firstly that the former argument does not necessarily imply that joint dynamics may be neglected in the presence of large motor and drive dynamics as was experienced in this work; and secondly, by inclusion of a friction model in the dynamic equations, the later argument is no longer an important factor at least in the case of non compliant motion.

The modelling of the robot does not need to be extremely accurate, although a CAD approach proved to be quite acceptable, as long as finer adjustments are made at a later stage using self-tuning methods. In other words a dynamic model of the robot that includes the motor and drive system dynamics can be used to reduce the nonlinear and coupling effects and then a self-tuner is employed to provide enhanced accuracy in tracking. The most appropriate self-tuner in relation to the above was found to be a variable structure one with no control weighting as described in chapters 6 and 7. Both the simulation and experimental results show the effectiveness of this approach.

Any deviations from the set trajectory due to variations in the payload of the manipulator can be suppressed by using an estimation method which was introduced in chapter 4 to allow payload adaptation. The effectiveness of the method which is developed by symbolic algebraic manipulation of the dynamic equations and is based on the state-variable filter approach and least squares estimation, was

demonstrated experimentally.

It should be noted that as the load mass estimation algorithm is based on the model of the robot, accuracy of this model is of prime importance for good results.

Use of symbolic algebraic manipulation also resulted in a more efficient set of dynamic equations which meant less computational effort in implementing the model-based controllers. However even with this reduced set of equations, inherent coarse parallelism of the control algorithms had to be exploited and to carry out the computations in real-time and to achieve the required sampling rates, parallel processing and fast processors (transputers) had to be employed.

Suitability of a transputer network for implementation of a robot programming system was also argued.

Suggestions for future research

It would be useful to pursue two areas as an extension to the work presented here; firstly an investigation into the cartesian space control along the same lines as the joint space approach that was dealt with in this work; and secondly look at the effects of adding a velocity self-tuner to the control system.

An interesting area of research in the field of robotic control is to develop suitable controllers for assembly operation applications. This entails considering multiple arm interactions, compliant motion and force control.

Appendix A

Part of the OCCAM code for comp. torque

THE EXE

```
PROC Controller (CHAN OF INT keyboard, CHAN OF ANY screen,
  [4]CHAN OF ANY from.user.filer, to.user.filer)
  #USE userio
  #USE interf
  #USE maths32.lib
  VAL linkout IS [0,1,2,3] :
  VAL linkin  IS [4,5,6,7] :
  CHAN OF ANY to.first, from.first :
  PLACE to.first AT linkout[1] :
  PLACE from.first AT linkin[1] :

  SEQ
    PROC datavalues (CHAN OF ANY screen)
      INT i,j:
      [1000]REAL32 data:
```

```
[1000]REAL32 data1:
[1000]REAL32 data2:
[1000]REAL32 time:
[1000]REAL32 error1:
[1000]REAL32 error2:
[1000]REAL32 error3:
[1000]REAL32 contsig1:
[1000]REAL32 contsig2:
[1000]REAL32 contsig3:
REAL32 p0, p1, p2, pn, t, t1, t2, tn, p2A, p2B:
REAL32 v1, a1, v2, a2:
REAL32 tp1, tp2, tp3, tp4, tp5, t1p2, t2p2, tnp2:
REAL32 delta1, delta2, deltan:
REAL32 tau, seg1, seg2, segn:
REAL32 tau0, tau1, tau2, taun:
REAL32 theta0,theta1,theta2:
SEQ
  write.text.line (screen,"Program running ...")
  seg1:= 2.0(REAL32)
  seg2:= 7.0(REAL32)
  segn:= 9.0(REAL32)
  p0:=0.0(REAL32)
  p1:=5.0(REAL32)
  p2:=25.0(REAL32)
  pn:=30.0(REAL32)
  delta1:=p1-p0
  delta2:=p2-p1
  deltan:=pn-p2
  t1:=2.0(REAL32)
```

```
t2:=5.0(REAL32)
tn:=2.0(REAL32)
tau0:=0.0(REAL32)
tau1:=2.0(REAL32)
tau2:=7.0(REAL32)
taun:=9.0(REAL32)
i:=0
j:=0
SEQ
  SEQ
    t1p2:= POWER (t1,2.0(REAL32))
    t2p2:=POWER (t2,2.0(REAL32))
    tnp2:=POWER (tn,2.0(REAL32))

    tau:=0.0(REAL32)

    WHILE tau < seg1
      SEQ
        t:=(tau-tau0)/t1
        tp2:=POWER (t,2.0(REAL32))
        tp3:=POWER (t,3.0(REAL32))
        tp4:=POWER (t,4.0(REAL32))
        tp5:=POWER (t,5.0(REAL32))
        p1 := (delta1 *tp3) + p0
        v1 := (3.0(REAL32) * delta1) / t1
        a1 := (6.0(REAL32) * delta1) / t1p2

        data[i] := p1
        data1[i] := v1
```

```
      data2[i] := a1
      time[i] := tau
      tau:=tau+0.01(REAL32)
      i:=i+1

tau:= seg1
WHILE (tau >= seg1) AND (tau < seg2)
  SEQ
    t:=(tau-tau1)/t2
    tp2:=POWER (t,2.0(REAL32))
    tp3:=POWER (t,3.0(REAL32))
    tp4:=POWER (t,4.0(REAL32))
    tp5:=POWER (t,5.0(REAL32))
    p2A:= (((((((6.0(REAL32) * delta2) +
          (-(3.0(REAL32) * (v1 *t2))))+
          (-(3.0(REAL32) * (v2 * t2)))) +
          (- (0.5(REAL32) * ( a1 * t2p2 )))) +
          (0.5(REAL32) * (a2 * t2p2)))) * tp5) +
          (((((((-(15.0(REAL32) * delta2)) +
          (8.0(REAL32) * (v1 * t2))) +
          (7.0(REAL32) * (v2 * t2))) +
          (1.5(REAL32) * (a1 * t2p2))) +
          (-(a2 * t2p2)))) * tp4)

    p2B:=(((((((10.0(REAL32) * delta2) +
          (-(6.0(REAL32) * (v1 * t2)))) +
          (-(4.0(REAL32) * (v2 * t2)))) +
          (-(1.5(REAL32) * (a1 * t2p2)))) +
          (0.5(REAL32) * (a2 * t2p2)))) * tp3) +
```

```
            ((0.5(REAL32) * (a1 * t2p2)) * tp2)) +
            (((v1 *t2) * t) + p1)
        p2 := p2A+p2B
        v2 := 3.0(REAL32) * (deltan / tn)
        a2 := (-(6.0(REAL32))) * (deltan / tnp2)

        data[i] := p2
        data1[i] := v2
        data2[i] := a2
        time[i] := tau
        i:=i+1
        tau:=tau+0.01(REAL32)

tau:= seg2
WHILE (tau >= seg2) AND (tau < segn)
  SEQ
    t:=(tau-tau2)/tn
    tp2:=POWER (t,2.0(REAL32))
    tp3:=POWER (t,3.0(REAL32))
    tp4:=POWER (t,4.0(REAL32))
    tp5:=POWER (t,5.0(REAL32))
    pn :=((deltan * tp3) +
            ((- (3.0(REAL32) * deltan)) * tp2))+
            (((3.0(REAL32) * deltan )* t) + p2)

    data[i] := pn
    data1[i] := 0.0(REAL32)
    data2[i] := 0.0(REAL32)
    time[i] := tau
```

```
            i:=i+1
            tau:=tau+0.01(REAL32)
      SEQ
        to.first ! data
        from.first ? error1;contsig1 ; error2;contsig2 ; error3;contsig3

        WHILE j<= 900
          SEQ
            write.real32 (screen, error1[j], 6, 4)
            write.real32 (screen, contsig1[j], 7, 6)
            write.real32 (screen, error2[j], 6, 4)
            write.real32 (screen, contsig2[j], 7, 6)
            write.real32 (screen, error3[j], 6, 4)
            write.real32 (screen, contsig3[j], 7, 6)
            INT pos:
            [80]BYTE text.line:
            SEQ
              pos := 0
              write.text.line (screen, [text.line FROM 0 FOR pos])
              j:=j+1
:

INT kchar:
INT error:
SEQ
  write.full.string(screen,"Do you want to file the output? ")
  read.echo.char (keyboard, screen, kchar)
  newline(screen)
  -- mask off alphabetic case
```

```
      VAL bchar IS BYTE (kchar /\ #5F):
      IF
        bchar = 'Y'
          CHAN OF ANY fromprog, tofile:
          INT foldnum:
          PAR
            SEQ
              datavalues (fromprog)
              write.endstream (fromprog)
            SEQ
              scrstream.fan.out (fromprog, tofile, screen)
              write.endstream (tofile)
            SEQ
              scrstream.to.file (tofile, from.user.filer[0],
                    to.user.filer[0], "datafile", foldnum, error)
              IF
                error = 0
                  SKIP
                TRUE
                  STOP
        TRUE
          datavalues (screen)
    write.full.string(screen, "Type ANY to return to TDS")
    INT any:
    read.char(keyboard, any)
    newline(screen)
:
Controller (keyboard, screen, from.user.filer, to.user.filer)
```

THE PROGRAM

```
VAL linkout IS [0,1,2,3] :
VAL linkin  IS [4,5,6,7] :
CHAN OF ANY fromROOT, toROOT :
[16]CHAN OF ANY links:

{{{  SC MX (CHAN OF ANY inn, outt, in1,out1, in2,out2, in3,out3)
{{{  MX (CHAN OF ANY inn, outt, in1,out1, in2,out2, in3,out3)
PROC MX (CHAN OF ANY inn, outt, in1,out1, in2,out2, in3,out3)

  [1000]REAL32 error1:
  [1000]REAL32 error2:
  [1000]REAL32 error3:
  [1000]REAL32 contsig1:
  [1000]REAL32 contsig2:
  [1000]REAL32 contsig3:
  [1000]REAL32 TRAJdata:
  SEQ
    inn ? TRAJdata
    PAR
      SEQ
        out1 ! TRAJdata
      SEQ
        out2 ! TRAJdata
      SEQ
        out3 ! TRAJdata
      SEQ
        in1 ? error1 ; contsig1
```

```
      SEQ
        in2 ? error2 ; contsig2
      SEQ
        in3 ? error3 ; contsig3
    outt ! error1;contsig1 ; error2;contsig2 ; error3;contsig3
:
}}}
}}}
...  SC position(CHAN OF ANY out1, out2, out3)
...  SC SIGNAL(CHAN OF ANY in1, in2, in3)
...  SC C1 (CHAN OF ANY in1, in2,out2, out )
...  SC C2 (CHAN OF ANY in1, in2,out2, out )
...  SC C3 (CHAN OF ANY in1, in2,out2, out )

PLACED PAR
  PROCESSOR 0 T8
    PLACE fromROOT AT linkin[2] :
    PLACE toROOT   AT linkout[2] :
    PLACE links[11]   AT linkin[3] :
    PLACE links[8]   AT linkout[3] :
    PLACE links[12]   AT linkin[0] :
    PLACE links[9]   AT linkout[0] :
    PLACE links[13]  AT linkin[1] :
    PLACE links[10]  AT linkout[1] :

    MX(fromROOT,toROOT,links[11],links[8],links[12],
             links[9],links[13],links[10])

  PLACED PAR
```

```
    PROCESSOR 1 T8
      PLACE links[0]  AT  linkout[0] :
      PLACE links[1]  AT  linkout[2] :
      PLACE links[3]  AT  linkout[3] :

      position(links[0], links[1], links[3])

  PLACED PAR
    PROCESSOR 2 T8
      PLACE links[4]  AT  linkin[2] :
      PLACE links[5]  AT  linkin[0] :
      PLACE links[6]  AT  linkin[1] :

      SIGNAL(links[4], links[5], links[6])
  PLACED PAR
    PROCESSOR 3 T8
      PLACE links[0]  AT  linkin[0] :
      PLACE links[8]  AT  linkin[3] :
      PLACE links[11] AT  linkout[3] :
      PLACE links[4]  AT  linkout[2] :

      C1(links[0], links[8], links[11], links[4])

  PLACED PAR
    PROCESSOR 4 T8
      PLACE links[1]  AT  linkin[0] :
      PLACE links[9]  AT  linkin[3] :
      PLACE links[12] AT  linkout[3] :
      PLACE links[5]  AT  linkout[2] :
```

```
    C2(links[1], links[9], links[12], links[5])

PLACED PAR
  PROCESSOR 5 T8
    PLACE links[3]  AT  linkin[0] :
    PLACE links[10] AT linkin[3] :
    PLACE links[13] AT linkout[3] :
    PLACE links[6]  AT  linkout[2] :

    C3(links[3], links[10], links[13], links[6])
```

Appendix B

Pascal and OCCAM SC progs for Filtering

THE PASCAL

```
PROGRAM Filter(Input, Output);
  PROCEDURE _cfilto ( x : real ) ; c ;
  PROCEDURE _cfilti ( var y : real ) ; c ;
  CONST
    Version = 'Version 1.1';
    NumberParameters = 3;
    MaxDegree = 4;
    MaxState = 2;
    MaxDimension = 6;
    Criterion = 0.01;
  TYPE
    Degree = - 1..MaxDegree;
    Polynomial =
      RECORD
        Deg: Degree;
```

```
      Coeff: ARRAY [0..MaxDegree] OF REAL
    END;
  StateVector = ARRAY [0..MaxState] OF REAL;
  TypeFilterKnobs =
    RECORD
      SampleInterval: REAL;
      ApproximationOrder: INTEGER;
      ConstantBetweenSamples: BOOLEAN;
    END;
  TypeFilterState =
    RECORD
      State: StateVector;
      Old: REAL;
    END;
  Dimension = 1..MaxDimension;
  Vector = ARRAY [Dimension] OF REAL;
  Matrix =
    RECORD
      Rows: Dimension;
      Columns: Dimension;
      Element: ARRAY [Dimension, Dimension] OF REAL;
    END;
VAR
  InData: TEXT;
  Psi, ControlSignal, SystemOutput: REAL;
  ControlState, OutputState: TypeFilterState;
  X, XPsi, Parameter: Vector;
  XX: Matrix;
  Cs: Polynomial;
```

```
    FilterKnobs: TypeFilterKnobs;
    Order: INTEGER;
    Done: BOOLEAN;
    aa : REAL;
    ii : INTEGER;
  PROCEDURE Initialise;
    VAR
      i, j: INTEGER;
    BEGIN {Initialise}
      WITH ControlState DO
        BEGIN
        FOR i := 0 TO MaxState DO State[i] := 0.0;
        Old := 0.0;
        END;
      WITH OutputState DO
        BEGIN
        FOR i := 0 TO MaxState DO State[i] := 0.0;
        Old := 0.0;
        END;
      WITH XX DO
        BEGIN
        Rows := NumberParameters;
        Columns := NumberParameters;
        FOR i := 1 TO NumberParameters DO
          FOR j := 1 TO NumberParameters DO
            Element[i, j] := 0.0;
        END;
      FOR i := 1 TO NumberParameters DO XPsi[i] := 0.0;
      WITH Cs DO
```

```
      BEGIN
      Deg := 2;
      Coeff[0] := 0.01;
      Coeff[1] := 0.2;
      Coeff[2] := 1;
      END;
    WITH FilterKnobs DO
      BEGIN
      SampleInterval := 1 / 121;
      ApproximationOrder := 5;
      ConstantBetweenSamples := FALSE;
      END;
  END {Initialise} ;

PROCEDURE FilterData;
  PROCEDURE StateVariableFilter
       (u {Signal to be filtered} : REAL;
        C: Polynomial;
        FilterKnobs: TypeFilterKnobs;
        VAR FilterState: TypeFilterState);
    VAR
      k, Index: INTEGER;
      Sum, hk: REAL;
      Increment: StateVector;
    BEGIN { StateVariableFilter }
      WITH FilterKnobs DO
        BEGIN
        IF ConstantBetweenSamples THEN
          FilterState.Old := u;
```

```
IF C.Deg = 0 THEN
  FilterState.State[0] := u / C.Coeff[0]
ELSE
  BEGIN
  FilterState.State[0] := 0.0;
  FOR Index := 0 TO C.Deg DO
    Increment[Index] := FilterState.State[Index];
  FOR k := 1 TO ApproximationOrder DO
    BEGIN
    Sum := 0.0;
    hk := SampleInterval / k;
    FOR Index := 1 TO C.Deg DO
      Sum := Sum - C.Coeff[Index] *
             Increment[Index];
    FOR Index := C.Deg DOWNTO 2 DO
      Increment[Index] := hk * Increment[Index -
                          1];
    Increment[0] := Sum / C.Coeff[0];
    IF k = 1 THEN
      Increment[0] := Increment[0] +
                      FilterState.Old /
                      C.Coeff[0];
    IF k = 2 THEN
      Increment[0] := Increment[0] + (u -
                      FilterState.Old) /
                      C.Coeff[0];
    Increment[1] := hk * Increment[0];
    FOR Index := 0 TO C.Deg DO
      FilterState.State[Index] := FilterState.
```

```
                                        State[Index] +
                                        Increment[Index]
                                        ;
          END;
        END;
      FilterState.Old := u;
      END {WITH FilterKnobs}
  END { of StateVariableFilter } ;

BEGIN {FilterData}
      ii := 1;
      WHILE ii < 901 DO
        BEGIN
          _cfilti(SystemOutput);
          StateVariableFilter(SystemOutput, Cs, FilterKnobs,
                              OutputState);

          X[1] := OutputState.State[0];
          X[2] := OutputState.State[1];
          X[3] := OutputState.State[2];

          _cfilto(X[1]);
          _cfilto(X[2]);
          _cfilto(X[3]);
          ii := ii + 1;
        END;
END {FilterData} ;

BEGIN {Filter}
```

```
    Initialise;
      BEGIN
      FilterData;
      END;
  END.
```

THE OCCAM SC

```
{{{  SC filter(CHAN OF ANY in1, out1)
{{{  filter(CHAN OF ANY in1, out1)
PROC filter(CHAN OF ANY in1, out1)
  CHAN OF ANY Aang, Bang, Cang, filA, filB, filC:
  {{{  fil(CHAN OF ANY Ain, Aout)
  PROC fil(CHAN OF ANY Ain, Aout)
    #USE globals.lib
    #USE streams.lib
    PROC foccam( [2] CHAN OF ANY chans )
      #USE userio.lib
      REAL32 fang, fvel, facc, angle:
      in  IS chans[0] :
      out IS chans[1] :
      INT size, i, noOFtrajectoryPOINTS:
      SEQ
        i := 0
        noOFtrajectoryPOINTS := 901
        WHILE i < noOFtrajectoryPOINTS
          SEQ
            Ain ? angle
            out ! angle
            in ? facc
```

```
            in ? fvel
            in ? fang
            Aout ! fang ; fvel ; facc
            i := i + 1
    :
    VAL bufSize   IS 256 :     -- size of string buffers
    [bufSize]BYTE args :       -- arguments to PASCAL
    INT result :               -- result from PASCAL
    [1]BYTE fStack :           -- dummy fast stack arrays
    [20 * 1024]BYTE heap :     -- heap space
    [2]CHAN OF ANY chans :     -- dummy inter-process channels
    CHAN OF ANY debug, toFSys, fromFSys:
    CHAN OF ANY screen, keyboard:

    #USE "tering.li8"
    SEQ
      args[0] := '*#00'
      PAR
        tering(fromFSys,  toFSys,
               keyboard,  screen,
               debug,     args,
               result,    chans,
               fStack,    heap)
        foccam(chans)
:
}}}
PAR
  SEQ
    REAL32 angle1, fang1, fvel1, facc1:
```

```
      REAL32 angle2, fang2, fvel2, facc2:
      REAL32 angle3, fang3, fvel3, facc3:
      INT iii:
      SEQ
        iii := 0
        WHILE iii < 901
          SEQ
            in1 ? angle1 ; angle2 ; angle3
            Aang ! angle1
            Bang ! angle2
            Cang ! angle3
            filA ? fang1 ; fvel1 ; facc1
            filB ? fang2 ; fvel2 ; facc2
            filC ? fang3 ; fvel3 ; facc3
            out1 ! fang1;fvel1;facc1 ; fang2;fvel2;facc2 ;
                   fang3;fvel3;facc3
            iii := iii + 1
    SEQ
      fil(Aang, filA)
    SEQ
      fil(Bang, filB)
    SEQ
      fil(Cang, filC)
:
}}}
}}}
```

Appendix C

Parameters and two procs for MBVSST

Values of parameters needed in the initial stage

```
{ ======== Assumed system  ========} {SystemInitialise}

 A.Deg := 2;
 A.Coeff[0] := 1.000000;
 A.Coeff[1] := 2.000000;
 A.Coeff[2] := 1.000000;
 B.Deg := 0;
 B.Coeff[0] := 1.000000;
 NumberInteractions := 0;
 D.Deg := 0;
 D.Coeff[0] := 0.000000;
 Delay := 0.000000 ;
 TuningInitialConditions := FALSE ;
```

```
 {======== Emulator design ======== } {DesignInitialise}
 ZHasFactorB := FALSE;
 ZMinusPlus := 1.000000;
 ZPlus := 1.000000;
 LQ := FALSE;
 {================================= }
 P.Deg := 1;
 P.Coeff[0] := 1.0;
 P.Coeff[1] := 1.0;
 C.Deg := 1;
 C.Coeff[0] := 1.0;
 C.Coeff[1] := 1.0;
 PadeOrder := 0;
 Small := 0.000100;

{ ======== Filters          ========} {InitFilterKnobs}
 SampleInterval :=  0.008251;
 ApproximationOrder := 5;
 ContinuousTime := TRUE;

{ ======== Identification  ======== } {TunerInitialise}

 InitialVariance := 100000.000000 ;
 ForgetTime := 1000.000000;
 DeadBand := 0.000000;
 On := TRUE ;
 TuneInterval := 1;

{ ================================= } {IdentifyInitialise}
```

```
 Cs.Deg := 1;
 Cs.Coeff[0] := 1.000000;
 Cs.Coeff[1] := 1.000000;
 IdentifyingRational := TRUE;
 IdentifyingDelay := FALSE;

 {======== Controller      ======== } {ControlInitialise}
 qNumerator.Deg := 0;
 qNumerator.Coeff[0] := 0.0;
 qDenominator.Deg := 0;
 qDenominator.Coeff[0] := 1.0;
 rNumerator.Deg := 0;
 rNumerator.Coeff[0] := 1.0;
 rDenominator.Deg := 0;
 rDenominator.Coeff[0] := 1.0;

{ ======== PutDataInit  ======== } {PutDataInitialise}
 Max := 10.0 ;
 Min := -10.0;
 Switched := TRUE;

 {======== STC type       ======== } {KnobsInitialise}
 Explicit := FALSE;
 UsingLambda := TRUE;
 IdentifyingSystem := TRUE;

{ ======== Control action  ========} {STCInitialise}
 Auto := TRUE;
 IntegralAction := TRUE;
```

```
{ ======== Data Source      ========} {RunInitialise}
 ExternalData := FALSE;
 LastTime := 0.016502;
 PrintInterval := 1;

{ ======== Actual system    ======== } {tSystemInitialise}
 A.Deg := 2;
 A.Coeff[0] := 1.000000;
 A.Coeff[1] := 2.000000;
 A.Coeff[2] := 1.000000;
 B.Deg := 0;
 B.Coeff[0] := 1.000000;
 D.Deg := 0;
 D.Coeff[0] := 0.000000;
 Delay := 0.000000 ;
```

ConveyData and DeliverData

```
PROCEDURE ConveyData(VAR ThisLoopVAR: TypeLoopVAR;
                     VAR LoopVAR: LoopVARs;
                     VAR Time: REAL;
                     RunKnobs: TypeRunKnobs;
                     FilterKnobs: TypeFilterKnobs);

  PROCEDURE Receive;

    VAR
      uD: REAL;
```

```
    j, Loop: INTEGER;
    Time, u, y, w : REAL;
    aa, bb, cc : REAL;
  BEGIN {Receive}
    WITH ThisLoopVAR, RunKnobs DO
      BEGIN
      _crec3(aa, bb, cc);
      u := aa;
      IF NOT Cascade OR (ThisLoop = 1) THEN uD := u
      ELSE uD := LoopVAR[ThisLoop - 1].y;
      WITH tSystemKnobs, tSystemState DO
        uD := MultiLag(uD, Lags, LagTimeConstant,
                       Interactive, FilterKnobs,
                       LagState);
      y := bb;
      j := 0;
      FOR Loop := 1 TO Loops DO
        IF NOT (Loop = ThisLoop) THEN
          WITH tSystemKnobs DO
            BEGIN
            j := j + 1;
            y := y + Filter(Interaction[j],
                            BInteraction[j], A,
                            FilterKnobs,
                            InteractionState[j + 1]);
            END;
      IF NOT Cascade OR (ThisLoop = Loops) THEN
       BEGIN
        w := cc;
```

```
          END
        ELSE w := LoopVAR[ThisLoop + 1].u;
        END;
    END {Receive} ;

  BEGIN {ConveyData}
     Receive;
  END {ConveyData} ;

PROCEDURE DeliverData(VAR u: REAL;
                      PutDataKnobs: TypePutDataKnobs);
  BEGIN {DeliverData}
    WITH PutDataKnobs DO
      BEGIN
      IF u > Max THEN u := Max
      ELSE IF u < Min THEN u := Min;
      IF Switched THEN
        IF Abs(u - Min) < Abs(u - Max) THEN u := Min
        ELSE u := Max;
      _csend(u);
      END;
  END {DeliverData} ;
```

Appendix D

Mass Matrix (M), and (Q) Vector

Definition of Symbols

(a) and (d) represent Denavit-Hartenberg parameters, (w) represents the waist (link1), (s) represents shoulder (link2), (e) represents elbow (link3). (cm) stands for centre of mass, (i) is inertia, and (m), mass. x, y, and z are directions. q represents angle and v angular velocity.

For example wx_{cm} is the distance of center of mass associated with the waist in x direction.

The Equations

$$
\begin{aligned}
M(1,1) := -\Big(\big(m_4 \cdot \big(-\big(\cos(2\cdot q_2 + 2\cdot q_3)\cdot a_3^2\big) - 2\cdot\cos(2\cdot q_2 + q_3)\cdot a_2\cdot a_3 \\
- 4\cdot\cos(q_2+q_3)\cdot a_1\cdot a_3 - \cos(2\cdot q_2)\cdot a_2^2 \\
- 4\cdot\cos(q_2)\cdot a_1\cdot a_2 - 2\cdot\cos(q_3)\cdot a_2\cdot a_3 \\
- 2\cdot a_1^2 - a_2^2 - a_3^2 - 2\cdot d_2^2 - 4\cdot d_2\cdot d_3 - 2\cdot d_3^2\big) \\
+ 2\cdot\sin(2\cdot q_2 + 2\cdot q_3)\cdot m_3\cdot a_3\cdot ey_{cm} + 2\cdot\sin(2\cdot q_2 + 2\cdot q_3)\cdot
\end{aligned}
$$

$$m_3 \cdot ex_{cm} \cdot ey_{cm} + 2 \cdot \sin(2 \cdot q_2 + q_3) \cdot m_3 \cdot a_2 \cdot ey_{cm}$$
$$+ 4 \cdot \sin(q_2 + q_3) \cdot m_3 \cdot a_1 \cdot ey_{cm} + 2 \cdot \sin(2 \cdot q_2) \cdot m_2 \cdot a_2 \cdot sy_{cm}$$
$$+ 2 \cdot \sin(2 \cdot q_2) \cdot m_2 \cdot sx_{cm} \cdot sy_{cm} + 4 \cdot \sin(q_2) \cdot m_2 \cdot a_1 \cdot sy_{cm}$$
$$+ 2 \cdot \sin(q_3) \cdot m_3 \cdot a_2 \cdot ey_{cm} - \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot a_3^2 - 2 \cdot$$
$$\cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot a_3 \cdot ex_{cm} - \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot ex_{cm}^2$$
$$+ \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot ey_{cm}^2 + \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{xx}$$
$$- \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{yy} - 2 \cdot \cos(2 \cdot q_2 + q_3) \cdot m_3 \cdot a_2 \cdot a_3$$
$$- 2 \cdot \cos(2 \cdot q_2 + q_3) \cdot m_3 \cdot a_2 \cdot ex_{cm} - 4 \cdot \cos(q_2 + q_3) \cdot m_3 \cdot a_1 \cdot a_3$$
$$- 4 \cdot \cos(q_2 + q_3) \cdot m_3 \cdot a_1 \cdot ex_{cm} - \cos(2 \cdot q_2) \cdot m_2 \cdot a_2^2$$
$$- 2 \cdot \cos(2 \cdot q_2) \cdot m_2 \cdot a_2 \cdot sx_{cm} - \cos(2 \cdot q_2) \cdot m_2 \cdot sx_{cm}^2$$
$$+ \cos(2 \cdot q_2) \cdot m_2 \cdot sy_{cm}^2 - \cos(2 \cdot q_2) \cdot m_3 \cdot a_2^2$$
$$+ \cos(2 \cdot q_2) \cdot si_{xx} - \cos(2 \cdot q_2) \cdot si_{yy} - 4 \cdot \cos(q_2) \cdot m_2 \cdot a_1 \cdot a_2$$
$$- 4 \cdot \cos(q_2) \cdot m_2 \cdot a_1 \cdot sx_{cm} - 4 \cdot \cos(q_2) \cdot m_3 \cdot a_1 \cdot a_2$$
$$- 2 \cdot \cos(q_3) \cdot m_3 \cdot a_2 \cdot a_3 - 2 \cdot \cos(q_3) \cdot m_3 \cdot a_2 \cdot ex_{cm} - 2 \cdot m_1 \cdot$$
$$a_1^2 - 4 \cdot m_1 \cdot a_1 \cdot wx_{cm} - 2 \cdot m_1 \cdot wx_{cm}^2 - 2 \cdot m_1 \cdot wz_{cm}^2 - 2 \cdot m_2 \cdot a_1^2$$
$$- m_2 \cdot a_2^2 - 2 \cdot m_2 \cdot a_2 \cdot sx_{cm} - 2 \cdot m_2 \cdot d_2^2 - 4 \cdot m_2 \cdot d_2 \cdot sz_{cm}$$
$$- m_2 \cdot sx_{cm}^2 - m_2 \cdot sy_{cm}^2 - 2 \cdot m_2 \cdot sz_{cm}^2 - 2 \cdot m_3 \cdot a_1^2 - m_3 \cdot a_2^2$$
$$- m_3 \cdot a_3^2 - 2 \cdot m_3 \cdot a_3 \cdot ex_{cm} - 2 \cdot m_3 \cdot d_2^2 - 4 \cdot m_3 \cdot d_2 \cdot d_3$$
$$- 4 \cdot m_3 \cdot d_2 \cdot ez_{cm} - 2 \cdot m_3 \cdot d_3^2 - 4 \cdot m_3 \cdot d_3 \cdot ez_{cm} - m_3 \cdot ex_{cm}^2$$
$$- m_3 \cdot ey_{cm}^2 - 2 \cdot m_3 \cdot ez_{cm}^2 - 2 \cdot wi_{yy} - si_{xx} - si_{yy} - ei_{xx} - ei_{yy}) / 2)$$

$$M(1,2) := -(m_4 \cdot (\sin(q_2 + q_3) \cdot a_3 \cdot d_2 + \sin(q_2 + q_3) \cdot a_3 \cdot d_3$$
$$+ \sin(q_2) \cdot a_2 \cdot d_2 + \sin(q_2) \cdot a_2 \cdot d_3)$$
$$+ \sin(q_2 + q_3) \cdot m_3 \cdot a_3 \cdot d_2 + \sin(q_2 + q_3) \cdot m_3 \cdot a_3 \cdot d_3$$
$$+ \sin(q_2 + q_3) \cdot m_3 \cdot a_3 \cdot ez_{cm} + \sin(q_2 + q_3) \cdot m_3 \cdot d_2 \cdot ex_{cm}$$
$$+ \sin(q_2 + q_3) \cdot m_3 \cdot d_3 \cdot ex_{cm} + \sin(q_2 + q_3) \cdot m_3 \cdot ex_{cm} \cdot ez_{cm}$$

$$+\sin(q_2)\cdot m_2\cdot a_2\cdot d_2+\sin(q_2)\cdot m_2\cdot a_2\cdot sz_{cm}$$
$$+\sin(q_2)\cdot m_2\cdot d_2\cdot sx_{cm}+\sin(q_2)\cdot m_2\cdot sx_{cm}\cdot sz_{cm}$$
$$+\sin(q_2)\cdot m_3\cdot a_2\cdot d_2+\sin(q_2)\cdot m_3\cdot a_2\cdot d_3$$
$$+\sin(q_2)\cdot m_3\cdot a_2\cdot ez_{cm}+\cos(q_2+q_3)\cdot m_3\cdot d_2\cdot ey_{cm}$$
$$+\cos(q_2+q_3)\cdot m_3\cdot d_3\cdot ey_{cm}+\cos(q_2+q_3)\cdot m_3\cdot ey_{cm}\cdot ez_{cm}$$
$$+\cos(q_2)\cdot m_2\cdot d_2\cdot sy_{cm}+\cos(q_2)\cdot m_2\cdot sy_{cm}\cdot sz_{cm})$$

$$M(1,3):=-(m_4\cdot\sin(q_2+q_3)\cdot a_3\cdot(d_2+d_3)+m_3\cdot$$
$$(\sin(q_2+q_3)\cdot a_3\cdot d_2+\sin(q_2+q_3)\cdot a_3\cdot d_3+\sin(q_2+q_3)\cdot a_3\cdot ez_{cm}$$
$$+\sin(q_2+q_3)\cdot d_2\cdot ex_{cm}+\sin(q_2+q_3)\cdot d_3\cdot ex_{cm}$$
$$+\sin(q_2+q_3)\cdot ex_{cm}\cdot ez_{cm}+\cos(q_2+q_3)\cdot d_2\cdot ey_{cm}$$
$$+\cos(q_2+q_3)\cdot d_3\cdot ey_{cm}+\cos(q_2+q_3)\cdot ey_{cm}\cdot ez_{cm}))$$

$$M(2,1):=-(m_4\cdot(\sin(q_2+q_3)\cdot a_3\cdot d_2+\sin(q_2+q_3)\cdot a_3\cdot d_3$$
$$+\sin(q_2)\cdot a_2\cdot d_2+\sin(q_2)\cdot a_2\cdot d_3)$$
$$+\sin(q_2+q_3)\cdot m_3\cdot a_3\cdot d_2+\sin(q_2+q_3)\cdot m_3\cdot a_3\cdot d_3$$
$$+\sin(q_2+q_3)\cdot m_3\cdot a_3\cdot ez_{cm}+\sin(q_2+q_3)\cdot m_3\cdot d_2\cdot ex_{cm}$$
$$+\sin(q_2+q_3)\cdot m_3\cdot d_3\cdot ex_{cm}+\sin(q_2+q_3)\cdot m_3\cdot ex_{cm}\cdot ez_{cm}$$
$$+\sin(q_2)\cdot m_2\cdot a_2\cdot d_2+\sin(q_2)\cdot m_2\cdot a_2\cdot sz_{cm}$$
$$+\sin(q_2)\cdot m_2\cdot d_2\cdot sx_{cm}+\sin(q_2)\cdot m_2\cdot sx_{cm}\cdot sz_{cm}$$
$$+\sin(q_2)\cdot m_3\cdot a_2\cdot d_2+\sin(q_2)\cdot m_3\cdot a_2\cdot d_3$$
$$+\sin(q_2)\cdot m_3\cdot a_2\cdot ez_{cm}+\cos(q_2+q_3)\cdot m_3\cdot d_2\cdot ey_{cm}$$
$$+\cos(q_2+q_3)\cdot m_3\cdot d_3\cdot ey_{cm}+\cos(q_2+q_3)\cdot m_3\cdot ey_{cm}\cdot ez_{cm}$$
$$+\cos(q_2)\cdot m_2\cdot d_2\cdot sy_{cm}+\cos(q_2)\cdot m_2\cdot sy_{cm}\cdot sz_{cm})$$

$$M(2,2):=-\left(m_4\cdot\left(-(2\cdot\cos(q_3)\cdot a_2\cdot a_3)-a_2^2-a_3^2\right)\right.$$
$$+2\cdot\sin(q_3)\cdot m_3\cdot a_2\cdot ey_{cm}-2\cdot\cos(q_3)\cdot m_3\cdot a_2\cdot a_3$$

$$-2\cdot\cos(q_3)\cdot m_3\cdot a_2\cdot ex_{cm}-m_2\cdot a_2^2-2\cdot m_2\cdot a_2\cdot$$
$$sx_{cm}-m_2\cdot sx_{cm}^2-m_2\cdot sy_{cm}^2-m_3\cdot a_2^2-m_3\cdot a_3^2$$
$$-2\cdot m_3\cdot a_3\cdot ex_{cm}-m_3\cdot ex_{cm}^2-m_3\cdot ey_{cm}^2-si_{zz}-ei_{zz}\Big)$$

$$M(2,3):=-\left(-\left((m_4\cdot a_3)\cdot(\cos(q_3)\cdot a_2+a_3)\right)+\sin(q_3)\cdot m_3\cdot\right.$$
$$a_2\cdot ey_{cm}-\cos(q_3)\cdot m_3\cdot a_2\cdot a_3-\cos(q_3)\cdot m_3\cdot a_2\cdot ex_{cm}$$
$$\left.-m_3\cdot a_3^2-2\cdot m_3\cdot a_3\cdot ex_{cm}-m_3\cdot ex_{cm}^2-m_3\cdot ey_{cm}^2-ei_{zz}\right)$$

$$M(3,1):=-\left(m_4\cdot\sin(q_2+q_3)\cdot a_3\cdot(d_2+d_3)+m_3\cdot\right.$$
$$(\sin(q_2+q_3)\cdot a_3\cdot d_2+\sin(q_2+q_3)\cdot a_3\cdot d_3+\sin(q_2+q_3)\cdot a_3\cdot ez_{cm}$$
$$+\sin(q_2+q_3)\cdot d_2\cdot ex_{cm}+\sin(q_2+q_3)\cdot d_3\cdot ex_{cm}$$
$$+\sin(q_2+q_3)\cdot ex_{cm}\cdot ez_{cm}+\cos(q_2+q_3)\cdot d_2\cdot ey_{cm}$$
$$\left.+\cos(q_2+q_3)\cdot d_3\cdot ey_{cm}+\cos(q_2+q_3)\cdot ey_{cm}\cdot ez_{cm})\right)$$

$$M(3,2):=-\left(-\left((m_4\cdot a_3)\cdot(\cos(q_3)\cdot a_2+a_3)\right)+\sin(q_3)\cdot m_3\cdot\right.$$
$$a_2\cdot ey_{cm}-\cos(q_3)\cdot m_3\cdot a_2\cdot a_3-\cos(q_3)\cdot m_3\cdot a_2\cdot ex_{cm}$$
$$\left.-m_3\cdot a_3^2-2\cdot m_3\cdot a_3\cdot ex_{cm}-m_3\cdot ex_{cm}^2-m_3\cdot ey_{cm}^2-ei_{zz}\right)$$

$$M(3,3):=m_4\cdot a_3^2+m_3\cdot a_3^2+2\cdot m_3\cdot a_3\cdot ex_{cm}+m_3\cdot ex_{cm}^2+m_3\cdot ey_{cm}^2+ei_{zz}$$

$$Q(1,1):=-\Big(m_4\cdot\Big(\sin(2\cdot q_2+2\cdot q_3)\cdot v_1\cdot v_2\cdot a_3^2+\sin(2\cdot q_2+2\cdot q_3)\cdot v_1\cdot v_3\cdot a_3^2$$
$$+2\cdot\sin(2\cdot q_2+q_3)\cdot v_1\cdot v_2\cdot a_2\cdot a_3+\sin(2\cdot q_2+q_3)\cdot v_1\cdot v_3\cdot a_2\cdot a_3$$
$$+2\cdot\sin(q_2+q_3)\cdot v_1\cdot v_2\cdot a_1\cdot a_3+2\cdot\sin(q_2+q_3)\cdot v_1\cdot v_3\cdot a_1\cdot a_3$$
$$+\sin(2\cdot q_2)\cdot v_1\cdot v_2\cdot a_2^2+2\cdot\sin(q_2)\cdot v_1\cdot v_2\cdot a_1\cdot a_2$$
$$+\sin(q_3)\cdot v_1\cdot v_3\cdot a_2\cdot a_3+\cos(q_2+q_3)\cdot v_2^2\cdot a_3\cdot d_2$$
$$+\cos(q_2+q_3)\cdot v_2^2\cdot a_3\cdot d_3+2\cdot\cos(q_2+q_3)\cdot v_2\cdot v_3\cdot a_3\cdot d_2$$
$$+2\cdot\cos(q_2+q_3)\cdot v_2\cdot v_3\cdot a_3\cdot d_3+\cos(q_2+q_3)\cdot v_3^2\cdot a_3\cdot d_2$$

$$+\cos(q_2+q_3)\cdot v_3^2\cdot a_3\cdot d_3+\cos(q_2)\cdot v_2^2\cdot a_2\cdot d_2$$

$$+\cos(q_2)\cdot v_2^2\cdot a_2\cdot d_3\Big)+\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_3^2$$

$$+2\cdot\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_3\cdot ex_{cm}$$

$$+\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_2\cdot ex_{cm}^2-\sin(2\cdot q_2+2\cdot q_3)\cdot$$

$$m_3\cdot v_1\cdot v_2\cdot ey_{cm}^2+\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_3^2$$

$$+2\cdot\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_3\cdot ex_{cm}$$

$$+\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot ex_{cm}^2-\sin(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot$$

$$ey_{cm}^2-\sin(2\cdot q_2+2\cdot q_3)\cdot ei_{xx}\cdot v_1\cdot v_2-\sin(2\cdot q_2+2\cdot q_3)\cdot ei_{xx}\cdot v_1\cdot v_3$$

$$+\sin(2\cdot q_2+2\cdot q_3)\cdot ei_{yy}\cdot v_1\cdot v_2+\sin(2\cdot q_2+2\cdot q_3)\cdot ei_{yy}\cdot v_1\cdot v_3$$

$$+2\cdot\sin(2\cdot q_2+q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_2\cdot a_3+2\cdot\sin(2\cdot q_2+q_3)\cdot$$

$$m_3\cdot v_1\cdot v_2\cdot a_2\cdot ex_{cm}+\sin(2\cdot q_2+q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_2\cdot a_3$$

$$+\sin(2\cdot q_2+q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_2\cdot ex_{cm}+2\cdot\sin(q_2+q_3)\cdot$$

$$m_3\cdot v_1\cdot v_2\cdot a_1\cdot a_3+2\cdot\sin(q_2+q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_1\cdot ex_{cm}$$

$$+2\cdot\sin(q_2+q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_1\cdot a_3+2\cdot\sin(q_2+q_3)\cdot m_3\cdot v_1\cdot$$

$$v_3\cdot a_1\cdot ex_{cm}-\sin(q_2+q_3)\cdot m_3\cdot v_2^2\cdot d_2\cdot ey_{cm}-\sin(q_2+q_3)\cdot m_3\cdot$$

$$v_2^2\cdot d_3\cdot ey_{cm}-\sin(q_2+q_3)\cdot m_3\cdot v_2^2\cdot ey_{cm}\cdot ez_{cm}-2\cdot\sin(q_2+q_3)\cdot$$

$$m_3\cdot v_2\cdot v_3\cdot d_2\cdot ey_{cm}-2\cdot\sin(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot d_3\cdot ey_{cm}-2\cdot$$

$$\sin(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot ey_{cm}\cdot ez_{cm}-\sin(q_2+q_3)\cdot m_3\cdot v_3^2\cdot d_2\cdot ey_{cm}$$

$$-\sin(q_2+q_3)\cdot m_3\cdot v_3^2\cdot d_3\cdot ey_{cm}-\sin(q_2+q_3)\cdot m_3\cdot v_3^2\cdot ey_{cm}\cdot ez_{cm}$$

$$+\sin(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_2^2+2\cdot\sin(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_2\cdot sx_{cm}$$

$$+\sin(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot sx_{cm}^2-\sin(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot sy_{cm}^2$$

$$+\sin(2\cdot q_2)\cdot m_3\cdot v_1\cdot v_2\cdot a_2^2-\sin(2\cdot q_2)\cdot si_{xx}\cdot v_1\cdot v_2$$

$$+\sin(2\cdot q_2)\cdot si_{yy}\cdot v_1\cdot v_2+2\cdot\sin(q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_1\cdot a_2$$

$$+2\cdot\sin(q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_1\cdot sx_{cm}-\sin(q_2)\cdot m_2\cdot v_2^2\cdot d_2\cdot sy_{cm}$$

$$-\sin(q_2)\cdot m_2\cdot v_2^2\cdot sy_{cm}\cdot sz_{cm}+2\cdot\sin(q_2)\cdot m_3\cdot v_1\cdot v_2\cdot a_1\cdot a_2$$

$$+\sin(q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_2\cdot a_3+\sin(q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_2\cdot ex_{cm}$$

$$+2\cdot\cos(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_3\cdot ey_{cm}$$

$$+2\cdot\cos(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_2\cdot ex_{cm}\cdot ey_{cm}$$

$$+2\cdot\cos(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_3\cdot ey_{cm}$$

$$+2\cdot\cos(2\cdot q_2+2\cdot q_3)\cdot m_3\cdot v_1\cdot v_3\cdot ex_{cm}\cdot ey_{cm}$$

$$+2\cdot\cos(2\cdot q_2+q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_2\cdot ey_{cm}+\cos(2\cdot q_2+q_3)\cdot$$

$$m_3\cdot v_1\cdot v_3\cdot a_2\cdot ey_{cm}+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_1\cdot v_2\cdot a_1\cdot ey_{cm}$$

$$+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_1\cdot ey_{cm}+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot a_3\cdot d_2$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot a_3\cdot d_3+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot a_3\cdot ez_{cm}$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot d_2\cdot ex_{cm}+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot d_3\cdot ex_{cm}$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_2^2\cdot ex_{cm}\cdot ez_{cm}+2\cdot\cos(q_2+q_3)\cdot m_3\cdot$$

$$v_2\cdot v_3\cdot a_3\cdot d_2+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot a_3\cdot d_3$$

$$+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot a_3\cdot ez_{cm}+2\cdot\cos(q_2+q_3)\cdot$$

$$m_3\cdot v_2\cdot v_3\cdot d_2\cdot ex_{cm}+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot d_3\cdot ex_{cm}$$

$$+2\cdot\cos(q_2+q_3)\cdot m_3\cdot v_2\cdot v_3\cdot ex_{cm}\cdot ez_{cm}+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot a_3\cdot d_2$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot a_3\cdot d_3+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot a_3\cdot ez_{cm}$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot d_2\cdot ex_{cm}+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot d_3\cdot ex_{cm}$$

$$+\cos(q_2+q_3)\cdot m_3\cdot v_3^2\cdot ex_{cm}\cdot ez_{cm}+2\cdot\cos(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_2\cdot sy_{cm}$$

$$+2\cdot\cos(2\cdot q_2)\cdot m_2\cdot v_1\cdot v_2\cdot sx_{cm}\cdot sy_{cm}+2\cdot\cos(q_2)\cdot m_2\cdot v_1\cdot v_2\cdot a_1\cdot sy_{cm}$$

$$+\cos(q_2)\cdot m_2\cdot v_2^2\cdot a_2\cdot d_2+\cos(q_2)\cdot m_2\cdot v_2^2\cdot a_2\cdot sz_{cm}$$

$$+\cos(q_2)\cdot m_2\cdot v_2^2\cdot d_2\cdot sx_{cm}+\cos(q_2)\cdot m_2\cdot v_2^2\cdot sx_{cm}\cdot sz_{cm}$$

$$+\cos(q_2)\cdot m_3\cdot v_2^2\cdot a_2\cdot d_2+\cos(q_2)\cdot m_3\cdot v_2^2\cdot a_2\cdot d_3$$

$$+\cos(q_2)\cdot m_3\cdot v_2^2\cdot a_2\cdot ez_{cm}+\cos(q_3)\cdot m_3\cdot v_1\cdot v_3\cdot a_2\cdot ey_{cm}\Big)$$

$$Q(2,1):=\Big(m_4\cdot\Big(\sin(2\cdot q_2+2\cdot q_3)\cdot v_1^2\cdot a_3^2+2\cdot\sin(2\cdot q_2+q_3)\cdot v_1^2\cdot a_2\cdot a_3$$

$$+2\cdot\sin(q_2+q_3)\cdot v_1^2\cdot a_1\cdot a_3+\sin(2\cdot q_2)\cdot v_1^2\cdot a_2^2$$

$$+ 2 \cdot \sin(q_2) \cdot v_1^2 \cdot a_1 \cdot a_2 - 4 \cdot \sin(q_3) \cdot v_2 \cdot v_3 \cdot a_2 \cdot a_3$$

$$- 2 \cdot \sin(q_3) \cdot v_3^2 \cdot a_2 \cdot a_3 + 2 \cdot \cos(q_2 + q_3) \cdot g \cdot a_3$$

$$+ 2 \cdot \cos(q_2) \cdot g \cdot a_2) + \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3^2$$

$$+ 2 \cdot \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3 \cdot ex_{cm} + \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot$$

$$ex_{cm}^2 - \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot ey_{cm}^2 - \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{xx} \cdot v_1^2$$

$$+ \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{yy} \cdot v_1^2 + 2 \cdot \sin(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot a_3$$

$$+ 2 \cdot \sin(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ex_{cm} - 2 \cdot \sin(q_2 + q_3) \cdot g \cdot m_3 \cdot ey_{cm}$$

$$+ 2 \cdot \sin(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot a_3 + 2 \cdot \sin(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot ex_{cm}$$

$$+ \sin(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot a_2^2 + 2 \cdot \sin(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot a_2 \cdot sx_{cm}$$

$$+ \sin(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot sx_{cm}^2 - \sin(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot sy_{cm}^2$$

$$+ \sin(2 \cdot q_2) \cdot m_3 \cdot v_1^2 \cdot a_2^2 - \sin(2 \cdot q_2) \cdot si_{xx} \cdot v_1^2$$

$$+ \sin(2 \cdot q_2) \cdot si_{yy} \cdot v_1^2 - 2 \cdot \sin(q_2) \cdot g \cdot m_2 \cdot sy_{cm}$$

$$+ 2 \cdot \sin(q_2) \cdot m_2 \cdot v_1^2 \cdot a_1 \cdot a_2 + 2 \cdot \sin(q_2) \cdot m_2 \cdot v_1^2 \cdot a_1 \cdot sx_{cm}$$

$$+ 2 \cdot \sin(q_2) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot a_2 - 4 \cdot \sin(q_3) \cdot m_3 \cdot v_2 \cdot v_3 \cdot a_2 \cdot a_3$$

$$- 4 \cdot \sin(q_3) \cdot m_3 \cdot v_2 \cdot v_3 \cdot a_2 \cdot ex_{cm} - 2 \cdot \sin(q_3) \cdot m_3 \cdot v_3^2 \cdot a_2 \cdot a_3$$

$$- 2 \cdot \sin(q_3) \cdot m_3 \cdot v_3^2 \cdot a_2 \cdot ex_{cm} + 2 \cdot \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3 \cdot ey_{cm}$$

$$+ 2 \cdot \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot ex_{cm} \cdot ey_{cm} + 2 \cdot \cos(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ey_{cm}$$

$$+ 2 \cdot \cos(q_2 + q_3) \cdot g \cdot m_3 \cdot a_3 + 2 \cdot \cos(q_2 + q_3) \cdot g \cdot m_3 \cdot ex_{cm}$$

$$+ 2 \cdot \cos(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot ey_{cm} + 2 \cdot \cos(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot a_2 \cdot sy_{cm}$$

$$+ 2 \cdot \cos(2 \cdot q_2) \cdot m_2 \cdot v_1^2 \cdot sx_{cm} \cdot sy_{cm} + 2 \cdot \cos(q_2) \cdot g \cdot m_2 \cdot a_2$$

$$+ 2 \cdot \cos(q_2) \cdot g \cdot m_2 \cdot sx_{cm} + 2 \cdot \cos(q_2) \cdot g \cdot m_3 \cdot a_2$$

$$+ 2 \cdot \cos(q_2) \cdot m_2 \cdot v_1^2 \cdot a_1 \cdot sy_{cm} - 4 \cdot \cos(q_3) \cdot m_3 \cdot$$

$$v_2 \cdot v_3 \cdot a_2 \cdot ey_{cm} - 2 \cdot \cos(q_3) \cdot m_3 \cdot v_3^2 \cdot a_2 \cdot ey_{cm}) / 2$$

$$Q(3,1) := (m_4 \cdot a_3 \cdot (\sin(2 \cdot q_2 + 2 \cdot q_3) \cdot v_1^2 \cdot a_3 + \sin(2 \cdot q_2 + q_3) \cdot v_1^2 \cdot a_2$$

$$+ 2 \cdot \sin(q_2 + q_3) \cdot v_1^2 \cdot a_1 + \sin(q_3) \cdot v_1^2 \cdot a_2$$

$$
\begin{aligned}
&+ 2 \cdot \sin(q_3) \cdot v_2^2 \cdot a_2 + 2 \cdot \cos(q_2 + q_3) \cdot g\Big) \\
&+ \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3^2 + 2 \cdot \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3 \cdot ex_{cm} \\
&+ \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot ex_{cm}^2 - \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot \\
&m_3 \cdot v_1^2 \cdot ey_{cm}^2 - \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{xx} \cdot v_1^2 \\
&+ \sin(2 \cdot q_2 + 2 \cdot q_3) \cdot ei_{yy} \cdot v_1^2 + \sin(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot a_3 \\
&+ \sin(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ex_{cm} - 2 \cdot \sin(q_2 + q_3) \cdot g \cdot m_3 \cdot ey_{cm} \\
&+ 2 \cdot \sin(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot a_3 + 2 \cdot \sin(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot ex_{cm} \\
&+ \sin(q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot a_3 + \sin(q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ex_{cm} \\
&+ 2 \cdot \sin(q_3) \cdot m_3 \cdot v_2^2 \cdot a_2 \cdot a_3 + 2 \cdot \sin(q_3) \cdot m_3 \cdot v_2^2 \cdot a_2 \cdot ex_{cm} \\
&+ 2 \cdot \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot a_3 \cdot ey_{cm} \\
&+ 2 \cdot \cos(2 \cdot q_2 + 2 \cdot q_3) \cdot m_3 \cdot v_1^2 \cdot ex_{cm} \cdot ey_{cm} \\
&+ \cos(2 \cdot q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ey_{cm} + 2 \cdot \cos(q_2 + q_3) \cdot g \cdot m_3 \cdot a_3 \\
&+ 2 \cdot \cos(q_2 + q_3) \cdot g \cdot m_3 \cdot ex_{cm} + 2 \cdot \cos(q_2 + q_3) \cdot m_3 \cdot v_1^2 \cdot a_1 \cdot ey_{cm} \\
&+ \cos(q_3) \cdot m_3 \cdot v_1^2 \cdot a_2 \cdot ey_{cm} + 2 \cdot \cos(q_3) \cdot m_3 \cdot v_2^2 \cdot a_2 \cdot ey_{cm}\Big) / 2
\end{aligned}
$$

Bibliography

[1] A.P. Ambler. Rapt: An object level robot programming language. In *Colloquium on "Languages For Industrial Robots"*. IEE Computing and Control Division, Feb 1982.

[2] G.M. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. *Proc. AFIPS*, 30, 1967.

[3] C. H. An, C. G. Atkeson, and J. M. Hollerbach. Experimental determination of the effect of feedforward control on trajectory tracking errors. In *Proc. IEEE conf. on Robotics and Automation*, pages 55–60, San Francisco, 1986.

[4] C.H. An, C.G. Atkeson, and J.M. Hollerbach. *Model-Based Control of Robot Manipulators.* The MIT Press, 1988.

[5] R.P. Anex and M. Hubbard. Modelling and adaptive control of a mechanical manipulator. *Journal of Dynamic Systems, Measurement and Control.*, 106, September 1984.

[6] K. J. Astrom and B. Wittenmark. On self-tuning regulators. *Automatica*, 9, 1973.

[7] E. Bailey and A. Arapostathis. Simple sliding mode control scheme applied to robot manipulators. *Int. J. Control*, 45(4):1197–1209, 1987.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed computation-Numerical Methods.* Prentice-Hall Inc., 1989.

[9] C. Blume. Implicit robot programming based on a high-level explicit system. In Prof. Keith Rathmill, editor, *Int. Trends in Manufacturing Tech (Robotic Assembly).* IFS (Publications) Ltd, U.K., 1985.

[10] S. Bonner and K.G. Shin. A comparative study of robot languages. In *Computer*, pages 82–96, December 1982.

[11] Booze-Allen and Hamilton Inc. Review of the state-of-the-art of assembly technologies and programming languages for robotic applications. Technical report, For U.S. Air Force, Arlington, VA., 1982.

[12] J.W. Burdick. An algorithm for generation of efficient manipulator dynamic equations. In *Proceedings IEEE International Conference on Robotics and Automation.* IEEE Computer Society Press, 1986.

[13] C. Canudas, K.J. Astrom, and K. Braun. Adaptive friction compensation in dc-motor drives. *IEEE Journal of Robotics and Automation*, RA-3(6), December 1987.

[14] D. W. Clarke and P. J. Gawthrop. A self-tuning controller. *IEE proc.*, 122:929–934, 1975.

[15] K. Collins, A.J. Palmer, and K. Rathmill. Development of a european benchmark for the comparison of assembly robot programming systems. In Prof. Keith Rathmill, editor, *Int. Trends in Manufacturing Tech (Robotic Assembly).* IFS (Publications) Ltd, U.K., 1985.

[16] J. J. Craig, P. Hsu, and S. Sastry. Adaptive control of mechanical manipulators. In *IEEE Int. Conf. on Robotics and Automation*, San Francisco, California, 1986.

[17] C.W. de Silva and A.G.J. MacFarlane. *Knowledge-Based Control with Application to Robots.* Lecture Notes in Control and Info. Sciences. Springer-Verlag, 1989.

[18] H. Demircioglu and P. J. Gawthrop. Continuous-time relay self-tuning control. *Int. J. Control*, 47(4), 1988.

[19] S. Dubowsky and D.T. DesForges. The application of model referenced adaptive control to robot manipulators. *Trans. ASME*, 101, 1979.

[20] D.L. Eager, J. Zahorjan, and E.D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3), March 1989.

[21] R. Featherstone. *Robot Dynamics Algorithms.* Ph.d. thesis, Edinburgh, 1984.

[22] A. F. Filippov. *Am. math Soc. Transactions*, 62, 1960.

[23] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee. *Robotics Control, Sensing, Vision, and Intelligence.* McGraw-Hill, Inc., 1987.

[24] P. J. Gawthrop. *Continuous-Time Self-Tuning Control : Design*, volume 1 of *Engineering Control.* Research Studies Press Ltd, Letchworth, Herts., England, 1987.

[25] P. J. Gawthrop. Robust stability of a continuous time self-tuning controller. *Int. J. Adaptive Control Signal Processing*, pages 31–84, 1987.

[26] P. J. Gawthrop. *Continuous-Time Self-Tuning Control*, volume 2. Research Studies Press, 1990.

[27] P.J. Gawthrop. Parametric identification of transient signals. *IMA Journal of Mathematical Control & Information*, 1984.

[28] W.A. Gruver, B.I. Soroka, J.J. Craig, and T.L. Turner. Industrial robot programming languages: A comparative evaluation. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(4):565–570, July-August 1984.

[29] J.Y. Han, H. Hemami, and S. Yurkovich. Nonlinear adaptive control of an n-link robot with unknown load. *The International Journal of Robotic Research*, 6(3), Fall 1987.

[30] K. Hashimoto and H. Kimura. A new parallel algorithm for inverse dynamics. *The International Journal of Robotics Research*, 8(1), February 1989.

[31] HDE/GP/PLG. *The GBUS96 User Manual.* Sension Ltd, Denton Drive, Northwich, Cheshire, U.K., November 1988.

[32] J.M. Hollerbach. A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamic formulation complexity. *IEEE Trans. on Systems, Man And Cybernetics*, 10, 1980.

[33] W. G. Holzbock. *Robotic Technology, Principles and Practice.* Van Nostrand Reinhold Co., 1986.

[34] R. Horowitz and M. Tomizuka. An adaptive control scheme for mechanical manipulators — compensation of nonlinearity and decoupling control. *Journal of Dynamic Systems, Measurement and Control*, 108, June 1986.

[35] Inmos Ltd. *Transputer Architecture.*

[36] Inmos Ltd. *Occam Programming Manual*, 1984.

[37] A. Izaguirre and R. Paul. Automatic generation of the dynamic equations of the robot manipulators using a lisp program. In *Proceedings IEEE International Conference on Robotics and Automation.* IEEE Computer Society Press, 1986.

[38] D.I. Jones and P.M. Entwistle. Parallel computation of an algorithm in robotic control. In *International Conference on Control 88*, 13-15 April 1988.

[39] A.M. Karnik and N.K. Sinha. Adaptive control of an industrial robot. *Robotica*, 4, 1986.

[40] H. Kasahara and S. Narita. Parallel processing of robot-arm control computation on a multimicroprocessor system. *IEEE Journal of Robotics and Automation*, RA-1(2), June 1985.

[41] S. Kawamura, F. Miyazaki, and S. Arimoto. Bettering operations of robots by learning. *Journal of Robotic Systems*, 1984.

[42] Haruhisha Kawasaki and Kunitoshi Nishimura. Terminal-link parameter estimation of robotic manipulators. *IEEE Journal of Robotics and Automation*, 4(5), 1988.

[43] P. K. Khosla and T. Kanade. Real-time implementation and evaluation of computed-torque scheme. *IEEE Transactions on Robotics and Automation*, 5(2), April 1989.

[44] M. Kinnaert and R. Hanus. Adaptive pole-placement with predictive action for robotic manipulator control. In *Proc. 16th International Symposium on Industrial Robots*, Brussels, Belgium, October 1986.

[45] D. E. Koditscheck. Natural motion for robot arms. In *IEEE procs of the 23th conf. on Decision and Control*, pages 733–735, LasVegas, 1984.

[46] D. Koditschek. Robot control systems. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence.* John Wiley and Sons, 1987.

[47] A.J. Koivo. Self-tuning manipulator control in cartesian base coordinate system. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 107, December 1985.

[48] A.J. Koivo and T. Guo. Adaptive linear controller for robotic manipulators. *IEEE Transactions on Automatic Control*, AC-28(2), February 1983.

[49] Antti J. Koivo. *Fundamentals For Control of Robotic Manipulators.* John Wiley and Sons, 1989.

[50] Y. D. Landau. Adaptive control-robustness and performance enhancement. In *Adaptive Systems in Control and Signal Processing*, Glasgow, U.K., April 1989. IFAC.

[51] R.H. Lathrop. Parallelism in manipulator dynamics. *Int. Journal of Robotics Research*, 4(2), 1985.

[52] C. S. G. Lee and M. J. Chung. An adaptive control strategy for computer based manipulators. In *Proc. 21st Conf. on Decision and Control*, pages 95–100, 1982.

[53] C.S.G. Lee, B.H. Lee, and R. Nigam. Development of the generalized d'alembert equations of motion for mechanical manipulators. In *Proceedings of the 22nd Conference on Decision and Control*, December 1983.

[54] G. G. Leininger. Adaptive control of manipulators using self-tuning methods. In *MIT Robotics Research 1st Int. Symp.*, NH, Sept. 1983.

[55] G. G. Leininger. Self-tuning adaptive control of manipulators. In *Adavanced Software in Robotics Int. Symp.*, 1983.

[56] W. Li and J. J. E. Slotine. Parameter estimation strategies for robotic applications. In *A.S.M.E. Winter Annual Meeting*, Boston, Massachusetts, 1987.

[57] K.Y. Lim and M. Eslami. New controller designs for robot manipulator systems. In *Proc. ACC*, Boston USA, 1985.

[58] J.P. Linde, H.H. Ven, and F.H.R. Lucassen. Adaptive robot control with an inverse model. In *Proc. 16th International Symposium on Industrial Robots*, Brussels, Belgium, October 1986.

[59] Mai-Hua Liu, L. Wei, and Y. Huang. Pole-assignment self-tuning control of robotic manipulators. In *Proc. 16th International Symposium on Industrial Robots*, Brussels, Belgium, October 1986.

[60] Mei-Hua Liu. An adaptive control strategy for robotic manipulators. In *Proc. 15th ISIR*, 1985.

[61] J.Y.S. Luh and C.S. Lin. Scheduling of parallel computation for a computer-controlled mechanical manipulator. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12(2), March/April 1982.

[62] G. Luo and G. N. Saridis. Robust compensation for a robotic manipulator. *IEEE Transactions on Automatic Control*, AC-29(6), June 1984.

[63] A. K. Mahalanabis and S. Galou. On the state and parameter estimation of a robotic system. In *Proceedings of IFAC 88*, 1988.

[64] David May. *Occam 2 Language Definition.* Inmos Ltd.

[65] H. Mayeda, K. Osuka, and A. Kangawa. A new identification method for serial manipulator arms. In *The 9th IFAC Congr.*, 1984.

[66] Meiko Ltd, 650 Aztec West, Bristol, U.K. *Hardware Reference Manual*, December 1988.

[67] R. H. Middleton and G. C. Goodwin. Adaptive computed torque control for rigid link manipulators. In *Proceedings of 25th Conference on Decision and Control*, Athens, Greece, December 1986.

[68] R. G. Morgan and U. O. Ozguner. A decentralised variable structure control algorithm for robotic manipulators. *IEEE J. Robotics Automat.*, RA-1:57–65, 1985.

[69] J.J. Murray and C.P. Newman. Organizing customized robot dynamics algorithms for efficient numerical evaluation. *IEEE Trans. on Systems, Man, and Cybernetics*, 18, 1988.

[70] R. Nigam and C.S.G. Lee. A multiprocessor-based controller for the control of mechanical manipulators. *IEEE Journal of Robotics and Automation*, RA-1(4), December 1985.

[71] R. Ortega and M. W. Spong. Adaptive motion control of rigid robots: a tutorial. *Automatica*, 25(26):877–888, 1989.

[72] R. Ortega and Y. Tang. Robustness of adaptive controllers-a survey. *Automatica*, 25(5):651–677, 1989.

[73] P. V. Osburn, H. P. Whitaker, and A. Kezer. New developments in the design of adaptive control systems. Technical Report 61-39, Inst. Aeronautical Sciences, 1961.

[74] K. H. Park and L. W. Dowdy. Dynamic partitioning of multiprocessor systems. *International J. of Parallel Programming*, 18(2), April 1989.

[75] R.P. Paul. *Robor Manipulators: Mathematics, Programming, and Control.* The MIT Press series in artificial intelligence, Cambridge, MA., 1981.

[76] B. Raucent, G. Campton, G. Bastin, and J. C. Samin. Identification of barycentric parameters of robotic manipulators from external measurements. In *Proceedings of IFAC 88*, 1988.

[77] N. Sadegh and R. Horowitz. Stability analysis of an adaptive controller for robotic manipulators. In *IEEE Int. Conf. on Robotics and Automationb*, Raligh, North Carolina, 1987.

[78] H. Seraji. Direct adaptive control of manipulators in cartesian space. *Journal of Robotic Systems*, pages 157–178, 1987.

[79] P.N. Sheth. *A Digital Computer Based Simulation Procedure For Multiple Degree Of Freedom Mechanical Systems With Geometric Constraints.* Ph.d. thesis, The University of Wisconsin, 1972.

[80] S.N. Singh and A.A. Schy. Robust trajectory following control of robotic systems. *Trans. of the ASME Journal of Dynamic Systems, Measurement and Control*, 107, December 1985.

[81] J. E. Slotine and S. S. Sastry. Tracking control of nonlinear systems using sliding surfaces, with application to robot manipulators. *Int. J. Control*, 1983.

[82] J. J. E. Slotine. Sliding controller design for nonlinear systems. *Int. J. Control*, 40:421, 1984.

[83] J. J. E. Slotine and W. Li. On the adaptive control of robot manipulators. In F. W. Paul and K. Youcef-Toumi, editors, *Robotics: Theory and Applications.* California, 1986. ASME winter annual meeting.

[84] J. J. E. Slotine and W. Li. Adaptive robot control-a new perspective. In *IEEE Conf. on Decision and Control*, LA, California, 1987.

[85] J. J. E. Slotine and W. Li. Composite adaptive control of robot manipulators. *Automatica*, 25(4):509–519, 1989.

[86] Jean-Jacques E. Slotine and Weiping Li. On the adaptive control of robot manipulators. *The Int. Journal of Robotics Research*, 6(3):49–59, Fall 1987.

[87] Mark W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons, 1989.

[88] SYNTEL Microsystems, Queens Mill Road Huddersfield U.K. *SYN ADC4 A/D Converter Module, User Manual*, 1985.

[89] SYNTEL Microsystems, Queens Mill Road Huddersfield U.K. *SYN DAC8 Digital to Analog Converter, User Manual*, 1985.

[90] M. Takeyaki and S. Arimoto. A new feedback method for dynamic control of manipulators. *Trans. ASME J. Dyn. Syst. Meas. Control*, 102, June 1981.

[91] V. D. Tourassis. In *Proc. Conf. on Applied Moition Control*, page 239, 1986.

[92] V. D. Tourassis and C. P. Neuman. Robust nonlinear feedback control for robotic manipulators. *IEE Proceedings*, 132(4), July 1985.

[93] H. Unbehauen and G.P. Rao. *Identification of Continuous Systems*, volume 10. North Holland Systems and Control Series, 1988.

[94] V. Utkin. Variable structure systems with sliding modes. *IEEE Transactions on Automatic Control*, AC-22(2), April 1977.

[95] VDI. Industrial robot data (irdata), general structure record types and transmission. Technical Report 2863, VEREIN DEUTSCHER INGENIEURE (German Engineers Association), July 1986.

[96] M. Vukobratovic and N. Kircanski. Decoupled control of robots via asymptotic regulators. *IEEE Transactions on Automatic Control*, AC-28(10), October 1983.

[97] Micheal W. Walker. Estimating manipulator load mass properties. In *Preceedings IEEE International Symposium on Intelligent Control*, Philadelphia, Pennsylvania, January 1987.

[98] M.W. Walker and D.E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Transactions of ASME, Journal of Dynamic Systems, Measurement, and Control*, 104, September 1982.

[99] J.P. Wander and D. Tesar. Pipelined computation of manipulator modeling matrices. *IEEE Journal of Robotics and Automation*, RA-3(6), December 1987.

[100] J. Wieslander and B. Wittenmark. An approach to adaptive control using real time identification. *Automatica*, 7, 1971.

[101] C. Wu and P. Paul. Manipulator compliance based on joint torque control. In *Proc. 9th IEEE Conf. Decision and Control*, 1980.

[102] J. Xu, H. Harhimoto, J. E. Slotine, Y. Arai, and F. Harashima. Implementation of vss control to robotic manipulators- smoothing modification. *IEEE Transactions on Industrial Electronics*, 1989.

[103] K. S. Yeung and Y. P. Chen. A new controller design for manipulators using the theory of variable structure systems. *IEEE Trans. on Automatic Control*, 33(2), Feb 1988.

[104] S. Yin and J. Yuh. An efficient algorithm for automatic generation of manipulator dynamic equations. In *Proceedings IEEE International Conference on Robotics and Automation.* IEEE Computer Society Press, May 1989.

[105] Y.F. Yong, J.A. Gleave, J.L. Green, and M.C. Bonney. Off-line programming of robots. In Y. Shimon, editor, *Handbook of Industrial Robotics.* John Wiley and Sons, 1985.

[106] Bu Yonghong and Wang Yi. An identification method for geometric parameter estimation of robot manipulators. In *Proceedings of IFAC 88*, 1988.

[107] K.D. Young. Controller design for a manipulator using theory of variable structure systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8(2), February 1978.