HARDWARE/SOFTWARE SYSTEM DESIGN FOR A ROBOT ARM

by

Raymond Hang Yick Lam, B.E.(Hons.)

Submitted in fulfilment of the requirements
for the degree of

Master of Engineering Science

UNIVERSITY OF TASMANIA
HOBART

1987

I declare that this thesis does not contain anything that has been accepted for the award of a degree or diploma in any other university, and also that, to the best of my knowledge and belief, the thesis does not contain a copy or a paraphrase of material written and/or published by anyone else except where due reference to it is made in the text of the thesis.

Raymond Hang Yick Lam

CONTENTS

ABSTRACT

This project is the design, implementation and evaluation of a control system for a robot manipulator. It is initiated as a foundation research in robotics in the Electrical Department, University of Tasmania. The Tasrobot0 is the first robot arm built in the Department. The design includes all the necessary electronic hardwares as well as softwares for the control of the manipulator.

A complete robot system is a multi-variable, interacting and non-linear system with time-varying parameters. As Hewit has pointed out: "no applicable corpus of control theory exists to deal with systems possessing such a combination of problematic features", the design of suitable controllers is impossible without making assumptions to simplify the system.

As it is still in the developing stage, the size and weight of the Tasrobot0 manipulator is far less than commonly encountered working robots. The interactions between the manipulator links are small. The joint systems are thus assumed to be mutually independent systems, with time-varying parameters resulting from changes in the arm configuration. With a suitable controller, this time-varying effect was shown to be insignificant in the closed-loop dynamic response of each joint system.

A trajectory planning technique was developed to generate cubic spline segment functions which interpolate between specified joint coordinates. This technique offers

optimality in the sense that it defines the shortest curve passing through the specified points while at the same time satisfying the velocity and acceleration contraints. In the operation mode, command signals are generated in real time from segment functions derived for each joint to control its motion. This helps smoothen jerky motions and reduce the deviations of the executed path from the planned path.

The design and developed techniques have been tested and are in use today controlling Tasrobot0.

ACKNOWLEDGEMENTS

I would like to express my thanks to the following people:

My supervisor Mr.Gregory The for his enthusiasm and constant assistance in my process throughout the thesis.

My parents, brothers and sisters, who have provided encouragement and support during my study overseas.

The workshop staff of the Electrical Engineering Departments, especially Mr.Bob Barclay and Mr.John Grace, for the equipment they have made for me.

Finally, my thanks to Ms.May Wong, my best friend, who helped prepare the drawings and edit the final draft of the thesis.

CHAPTER ONE

INTRODUCTION

1.1 CONTRIBUTIONS

The contributions of this thesis are:

- ♦Development of a technique for modelling and controlling small and light weight robot arms.
- ♦Development of a trajectory planning technique which guarantees continuity in joint velocities and accelerations of a robot manipulator. The technique also ensures the joint velocities and accelerations are within the mechanical limits of the manipulator.
- ♦Development of a system identification software package for identifying third order systems. The package is also applicable to higher order systems upon minor modifications.
- ♦Development of a technique to convert transfer functions from discrete-time model to continuous-time model with zero-order-hold data extrapolator.
- ♦A paper[R38] on the identification and control system design of a robot manipulator was accepted by the IASTED for presentation at the 15th IASTED International Conference on Applied Simulation and Modelling, held at Santa Barbara, California USA on May 26-29, 1987.
- ♦Two papers, one on the identification and control of a robot manipulator, one on the technique of trajectory planning of a robot manipulator, were accepted for presentation as poster papers at the IREECON '87. Regretfully, these papers have to be withdrawn due to lack of Department travel funds.

1.2 THESIS OUTLINE

This thesis is an account of the design and implementation of hardware and software control system for a robot manipulator.

Chapter 1 starts with the contributions and thesis outline, continues with a brief description of a robot system and ends with a review of the programming methods commonly encountered in robot systems.

Chapter 2 describes the mechanical structure and the control schemes of the Tasrobot0 manipulator.

Chapter 3 summarizes the hardware design implementing the control schemes discussed in chapter 2. Hardware interface to the IBM/PC microcomputer, the host computer of the Tasrobot0 system, is also included.

Chapter 4 describes the model building technique for the joint systems. Also detailed are the theories, techniques and results of the identification of the model parameters.

Chapter 5 contains an analysis of the results in chapter 4. A complete model of each joint system is established to facilitate controller design. Non-linearities and time-varying parameters of the joint systems are also addressed.

Chapter 6 describes techniques for trajectory planning and path execution.

Chapter 7 reviews the software design of the control system of Tasrobot0. It contains detailed descriptions of developed softwares for robot programming, trajectory planning and path execution.

Chapter 8 summarizes the contributions and gives possible extensions to the work as motivation for further work.

1.3 DEFINITION OF ROBOTS

In order to distinguish a robot from many single-purpose machines, which have some features making them look like robots, a clear definition is important. The definition developed by the Robotics International Division of the Society of Manufacturing Engineers (RI/SME) is as follows:-

> "A robot is a reprogrammable multifunctional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks."

Reprogrammable in the definition means that the machine can be programmed repeatedly to perform a new or different task. The definition also emphasizes multifunctional which means that the machine must be able to perform many different functions, depending on the program and tooling used. By this definition, most single-purpose machines are not classified as robots since they are usually not programmable and can only perform single function. Even though tele-operators look like robots, they cannot be classified as robots according to the robot definition because they require human operator to perform a task at a distance and are not programmable.

1.4 THE ROBOT SYSTEM

At present, industrial robots are actually mechanical handling devices that can be manipulated under computer control. A basic robot system is illustrated in Figure (1.4-

1). The system includes a manipulator, a computer and a power source.

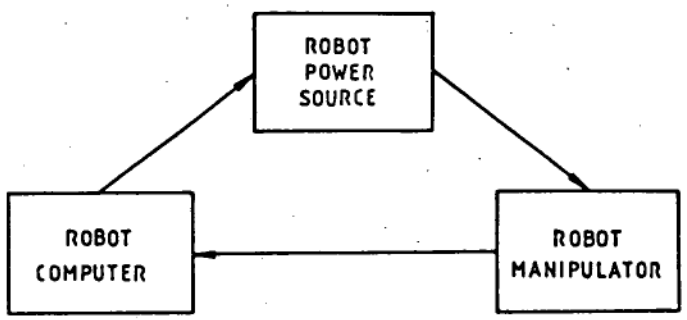


Figure (1.4-1): A Basic Robot System

(a) Robot Manipulator

A robot manipulator, which does the physical work of a robot system, is a mechanical device driven by electric motors, pneumatic devices or hydraulic actuators. In general, the structure of a robot manipulator consists of an arm, a wrist and an end-effector. An end-effector is located at the end of a manipulator where a working tool is attached. The motion of the arm of the manipulator controls the position of the end-effector while the motion of the wrist controls the orientation of the end-effector with respect to a work piece. Typically, an arm as well as a wrist consists of a sequence of mechanical links connected by joints. Each joint is driven to provide linear or rotational motion by a driving element which is either a prismatic or rotatory actuator. A working tool attached to the end-effector can be a welding head, a spray gun, a machining tool or a gripper with open-close jaws, depending on the applications of the robot.

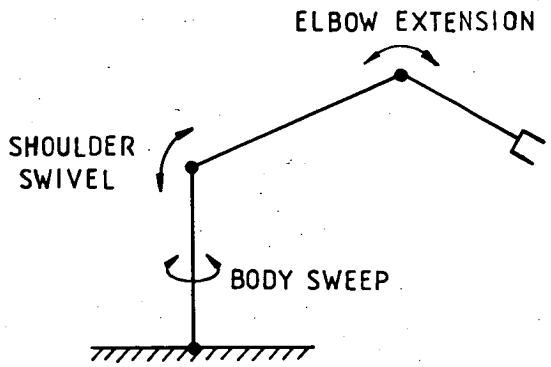


Figure (1.4a-1): A Typical Structure of An Arm of A Manipulator

The positioning of an end-effector in space requires motion of at least three degree-of-freedom and is controlled by the motion of the manipulator arm. A typical structure of an arm consists of three links as shown in Figure (1.4a-1).

Robots can be classified into four basic groups according to the characteristics of arm motion or their geometric principles. The basic geometries include rectangular, cylindrical, spherical and revolute as shown in Figure (1.4a-2). Regardless of the type of robot, an arm of

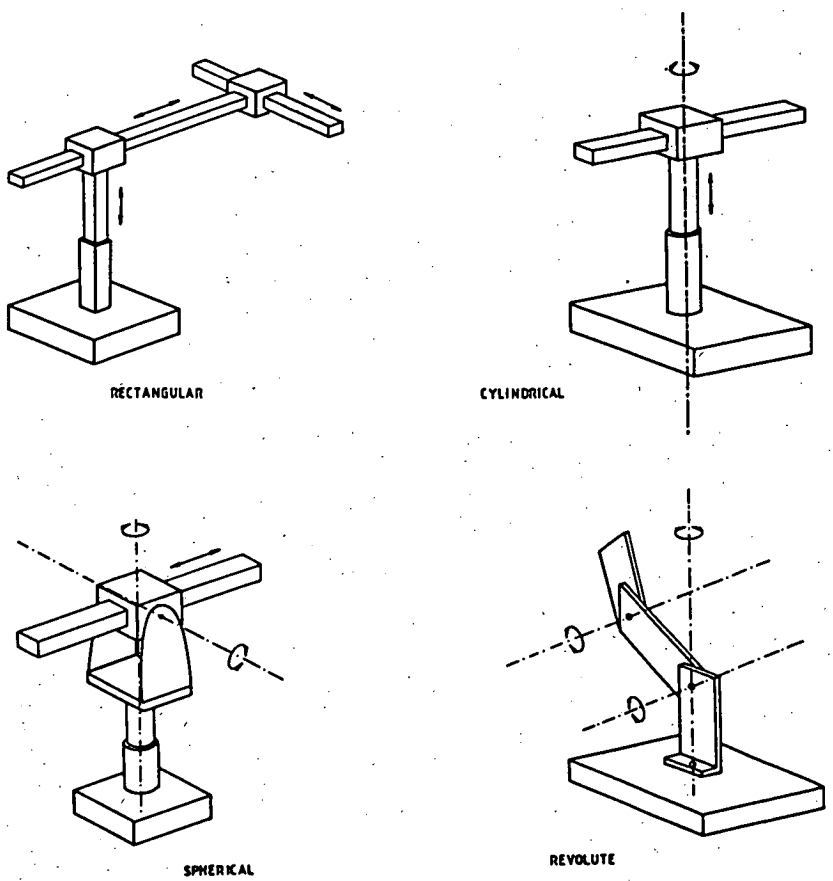


Figure (1.4a-2): Four Basic Geometries of A Robot

a robot typically consists of three movable joints. The combined motion of these three joints enables the manipulator to move to required position within its work space.

The orientation of the end-effector is controlled by the motion of the manipulator wrist. The wrist motion is often a sequence of axial rotations which provide three-degree-freedom motion for orientation of the end-effector. The two types of angles most frequently used to describe orientation of the wrist motions are the Euler angles and the Roll-Pitch-Yaw angles as shown in Figure (1.4a-3). A combination of these axial rotations enables the end-effector to take any arbitrary orientation in space. Nevertheless, there are robots whose wrists are capable of only two or even one degree-of-freedom motion. These types of robots, however, have limitations in their applications.

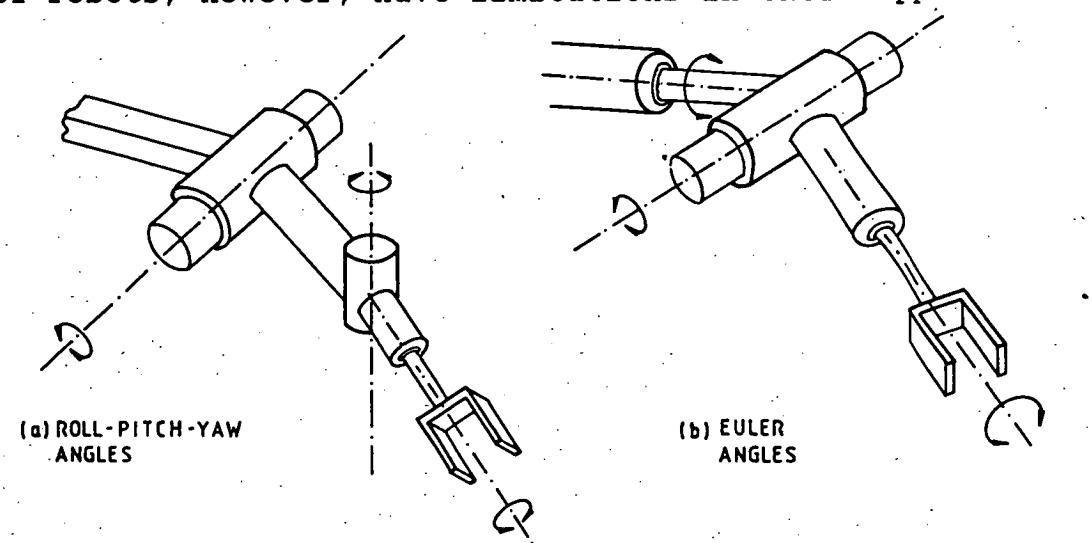


Figure (1.4a-3): Two Typical Types of Angles describing Wrist Motions

(b) Robot Computer

A robot computer is the source of intelligence of a robot system. Its presence distinguishes a robot from a teleoperator. The function of a robot computer is to control

the motion of a manipulator. It senses signals from sensors on the manipulator and generates appropriate command signals to drive the manipulator.

A robot computer block can be divided into two units, the intelligent unit and the controller unit. The intelligent unit is often called the host computer, whose functions are to communicate with robot users, to store necessary information of a task, to plan trajectories for a task, to process signals from sensors and, in some advanced robots, to make decisions according to stimuli from the environment. The controller unit is essentially a path controlling unit which receives command signals from the host computer and, in turn, generates appropriate signals to drive the manipulator.

The controllers used in the controller unit may be servo or non-servo. A servo-controller unit may be as simple as a position feedback controller. However, a position feedback controller can only result in a basically point-to-point robot. A more sophisticated servo controller unit can receive command signals containing information such as required Cartesian position and orientation as well as desirable velocity of the end-effector of the manipulator. Suitable signals to control the manipulator joints are then generated such that the end-effector will move to the required position and orientation with the desired velocity. A controller of this type is required for continuous-path robot.

(c) Robot Power Source

There are three primary power sources most commonly found in robot systems in driving a robot manipulator. They are, namely, hydraulic, pneumatic and electric power systems.

A basic hydraulic power system is illustrated in Figure (1.4c-1). The principle of a hydraulic system is to force high-pressure incompressible oil into a hydraulic actuator which converts hydraulic energy into mechanical energy. The motor-driven pump provides oil at high pressure for the system from the oil tank while the motor-controlled-four-way valve switches the direction of flow of the high pressure oil into or out of the actuator thus controlling the direction of motion of the driven actuator.

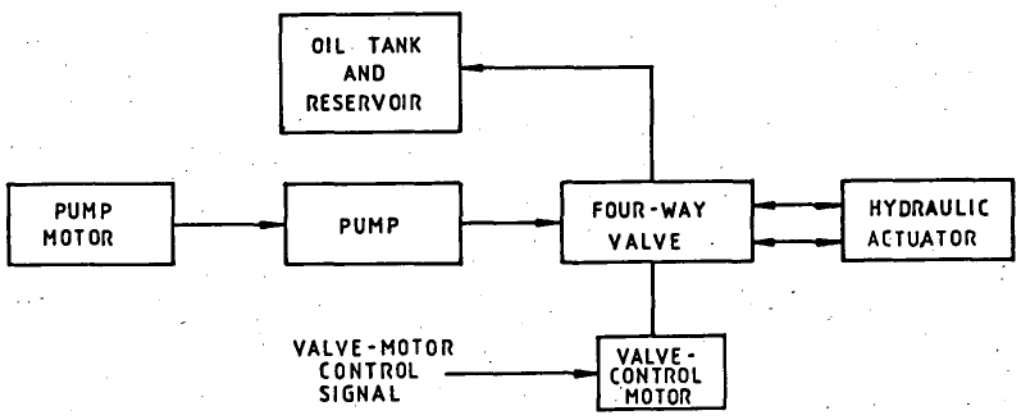


Figure (1.4c-1): A Basic Structure of A Hydraulic Power System

An advantage in using hydraulic actuator is that it has a very large power-to-size ratio and thus is capable of handling large loads. However, the high cost in equipment, the oil leakage problem and the high maintenance cost tend to outweight its advantage.

The basic components in a pneumatic drive system is same as those in a hydraulic system. A major difference is

that power is being transferred by gas, usually air, under pressure rather than by oil. An obvious advantage of this system over a hydraulic system is that system leakage does not cause contamination problem to the work area. Also, the total cost for a pneumatic system is less than that for a hydraulic system. However, using pneumatic drive system presents difficulties in achieving feedback control to provide proportional operation and multiple stops due to the properties of the compressed gas. Therefore, in most pneumatic robots, the acutators are driven against fixed stops at extremes of travel.

The electric system includes a source of electrical power and an electric motor. In most applications, the motors used are servo motors, but stepper motors are also used in some robots where the payload is small. A major disadvantage of stepper motors in robot applications concerns the load torque to allowable speed characteristics of a stepper motor. The driving speed of a stepper motor, at large loads, cannot be too high; otherwise, loss of steps will result. This loss of steps cannot be detected and will result in permanent position errors. To allow variations of load torque, the driving speed of a stepper motor is usually kept low. Consequently, the motion of the driven link is slow.

Servo motors are usually d.c. motors although a.c. motors are also used in some robot applications. An electric motor provides an excellent source of rotational torque either directly, or indirectly through gearings, and is most commonly used in driving revolute joints. Linear joint

motion can also be achieved by using ballscrew drive which is analogous to bolt-and-nut operation. The advantages of servo motors, particularly d.c. motors, are their excellent speed regulation, high torque and high efficiency and are therefore ideally suited for control applications.

1.5 TEACHING METHODS

In general, a robot computer in a robot system involves a control program and a task program. The control program is provided by the robot designer to control each joint of the manipulator. The task program is provided by the robot operator to specify the required manipulator motions for a particular job.

The way of generating a task program depends on the method of teaching or programming employed in a robot system. There are two methods of teaching a robot: off-line programming and on-line programming.

(a) Off-line Programming

In off-line programming, the robot operator makes use of commands set by the robot designer to specify conditions for a job, such as required positions and orientations of the end-effector, its velocity and acceleration for each specified point in space. The method is characterized by commands which inform the robot what to do, but the robot itself is not used during the programming stage.

This method provides a quick way in programming when the required Cartesian points in space are relatively easy to access and measure. In practice, however, specifying orientations of the end-effector with respect to a work

piece can be tedious. Also, it is not easy to visualize the necessary rotation for each wrist joint nor the work limit of the manipulator. User-specified points may not be accessible by the robot and thus have to be checked before a task path can be planned or executed. Moreover, in cases where the location of the robot manipulator is frequently changed, off-line method becomes undesirable.

(b) On-line Programming

Teach-by-showing is referred to as on-line programming and the robot itself is used during the programming stage.

During the programming stage, the operator moves the robot arm through a set of required points or a desired path in space by means of some teaching aids such as a teaching pendant, a control handle or a joystick. When a required point is reached, the operator presses a memory button and the system will `remember' the joint coordinates at that instant. These joint coordinates are normally recorded so that the robot will aware of its physical working environment and avoids collision with obstacles during the execution of a task.

The additional teaching aid required for robot systems using on-line programming may seem to be a disadvantage but the software required during the programming stage is far simpler than that required by off-line programming method . Besides, critical points are easily realized by operator during teaching stage. Depending on the teaching aid employed, on-line programming can be subdivided into individual-joint teaching or lead-through teaching.

(i) Individual-joint Teaching

In individual-joint teaching, a teaching pendant (teaching box) is used. The pendant consists of push buttons and teaching is done by pressing appropriate buttons to rotate each joint of the robot until the combination of all joint positions and orientations yields the desired position and orientation of the end-effector in space. Then the operator stores the joints coordinates of the robot at that instant by pressing a memory button. The process is repeated for each required point in space until the task program is completed. This teaching method demands patience in adjusting all joint axes, one by one, every time in giving a required point setting of the end-effector in space. A large amount of time is thus normally required in teaching the robot.

There are also teaching pendant which, instead of controlling a number of joint positions separately, can directly manipulate the position and orientation of the end-effector in space. During the teaching stage, the tool tip of the robot can be moved in a straight line and rotated about fixed axis in space. Although this solves the problem of adjusting one joint axis at a time, the software required for this type of teaching pendant is more complicated and involves lengthy mathematics for required transformations from Cartesian to joint coordinates and vice versa. The resulting motion is, therefore, usually very slow.

(ii) Lead-through Teaching

A better way of teaching a robot is perhaps by grasping the robot's end-effector, leading it through a desired path,

and simultaneously recording the joint positions at desired points. Although teaching aid is not required in this method while the same robot arm is used for teaching, the fact that large forces are required to move the arm against its driving and transmission elements prevents this method from being ideally used in practice.

A better approach is to use a teach arm which is similar to the actual manipulator but with far simpler structure. A teach arm may be equipped with position transducers on each joint, but with no driving elements nor transmission elements so that it can easily be moved to take any configurations. During teaching stage, the actual arm follows the configuration of the teach arm; desired positions and orientations of the actual arm can be input by push buttons on the teach arm.

CHAPTER TWO

THE TASROBOT0 SYSTEM

The Tasrobot0 system is the first robot arm system built in University of Tasmania. Similar to most robot systems, it has three basic components: the robot manipulator, the robot computer and the robot power source.

2.1 THE MANIPULATOR

The Tasrobot0 manipulator belongs to the class of revolute coordinate robot and has five axes of rotation. The end-effector of the manipulator is a permanently attached open/close gripper with two movable jaws. The arm has three-degree-of-freedom for positioning and two-degree-of-freedom for orientating the gripper with respect to a work piece.

(a) The Arm

Positioning of the gripper is controlled by motion of three revolute joints which are denoted as body joint, shoulder joint and elbow joint. These three joints connect four links of the manipulator to form the major physical structure of the manipulator. The four links are denoted as base, body, shoulder and elbow as shown in Figure (2.1-1).

The motion of each joint is provided through direct gearings by an actuator which is a permanent magnet dc motor. The position of each of the three joints is indicated by a position transducer mounted on respective rotating

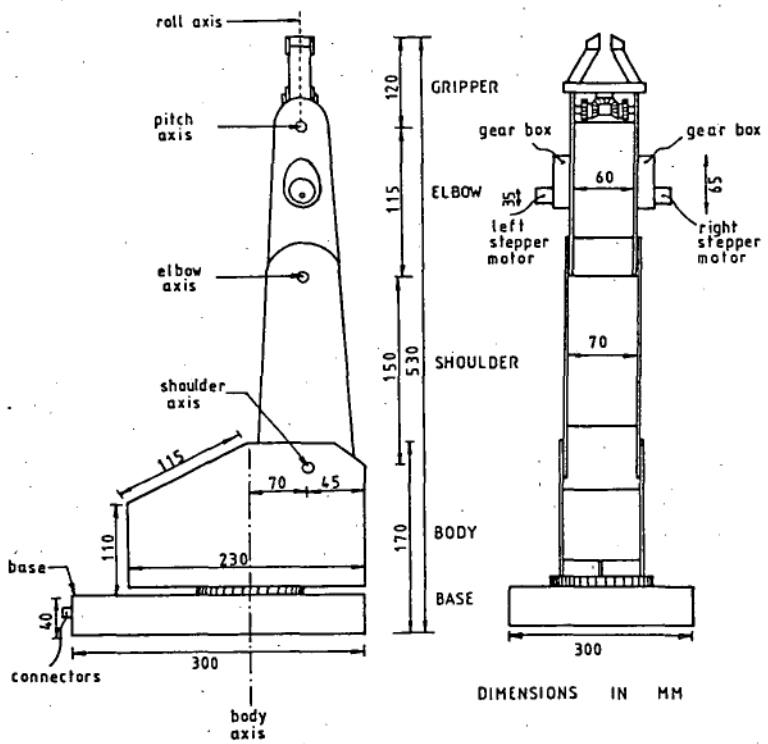


Figure (2.1-1): The Tasrobot0 Manipulator

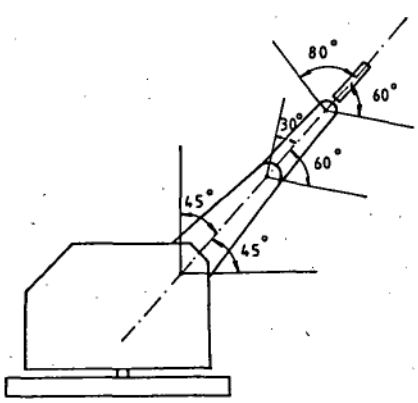
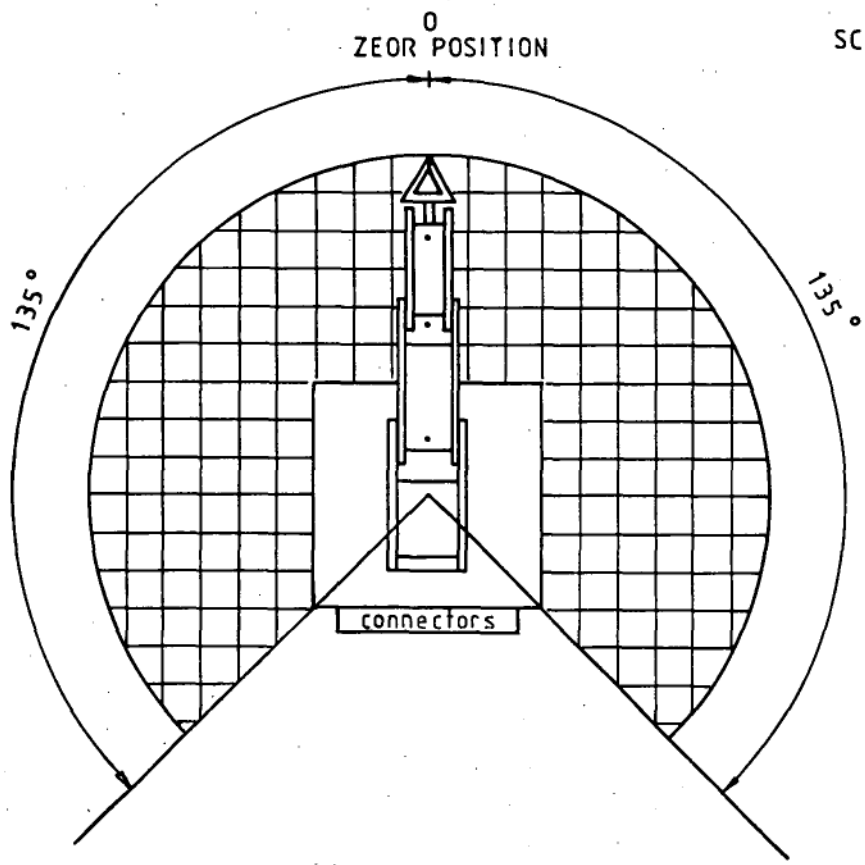


Figure (2.1a-1): Range of Motion of The Tasrobot0 Manipulator

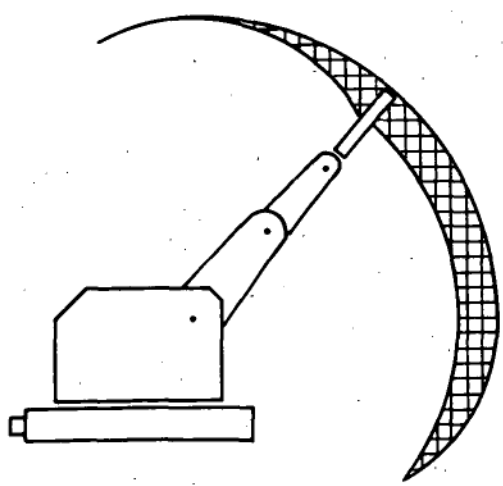
shaft of the joint. The body joint has a vertical axis of rotation and its motion range is 270°. Each of the shoulder joint and elbow joint has a horizontal axis of rotation and a motion range of 90° as shown in Figure (2.1a-1).

The work envelope of the system, which is defined as the reach of the robot, is illustrated in Figure (2.1a-2). A relatively small range of motion of the elbow joint compared with other robot systems accounts for the small side-view envelope of the Tasrobot0 manipulator.



(a) TOP VIEW

SCALE : 1 mm = 10mm



(b) SIDE VIEW

Figure (2.1a-2): Work Envelope of The Tasrobot0 Manipulator

(b) The Wrist

The wrist of the Tasrobot0 manipulator consists of two axes of rotation for controlling the orientation of the gripper with respect to a work piece. Two stepper motors are used to give pitch and roll rotations as shown in Figure (2.1b-1). The lack of yaw rotation does not allow the gripper to reach objects not aligned with the arm.

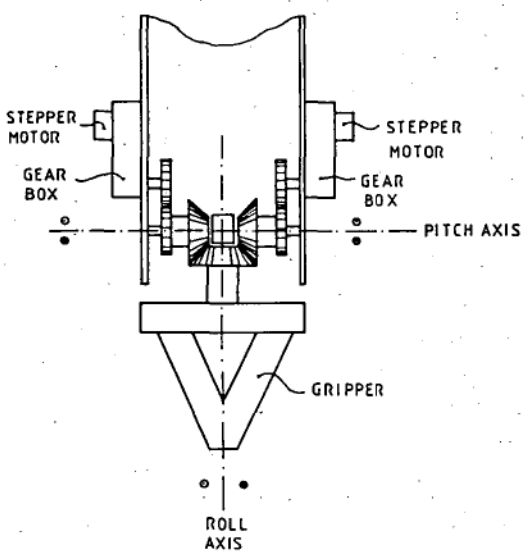


Figure (2.1b-1): Gearings For The Wrist Rotation

Pitch rotation is achieved by rotating the two stepper motors in same direction. Roll rotation results when the two stepper motors are stepping in different directions. The range of motion for roll rotation is physically unlimited, while the range of motion for pitch rotation is limited to 140° as shown in Figure (2.1a-1).

(c) The Gripper

The end-effector of the Tasrobot0 manipulator is a gripper. It consists of two jaws, each of which is a two-bar-linkage structure. Normally, the jaws are closed by the action of a compression spring and can be opened by pulling against the spring. The pulling force is provided by a dc motor with gearings. The whole conceptual structure of the gripping system is illustrated in Figure (2.1c-1).

To reduce the effect of weight of the gripping system on the joints of the manipulator, the gripper driving unit is installed in the body link. But then the steel string has

to pass through all links of the manipulator and the effective length of the steel string has to depend on the configuration of the arm. As a result, the amount of rotation required for the gripper motor to rotate to open or close the gripper depends on the configuration of the arm.

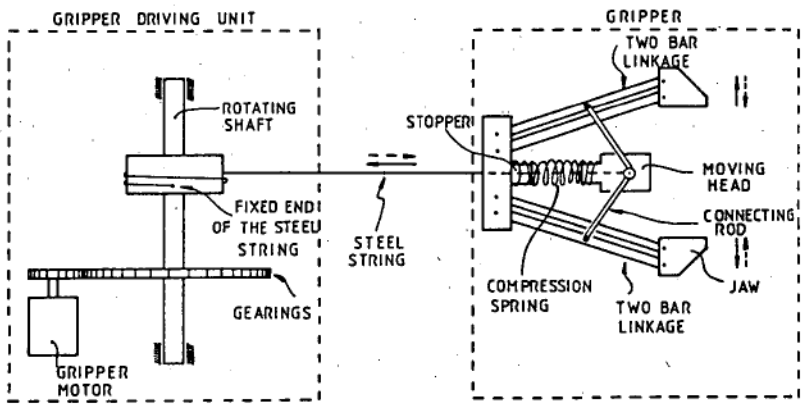


Figure (2.1c-1): Conceptual Structure of The Gripping System

A position transducer is used to record the amount of rotation required for opening the gripper in a specific arm configuration awared during the teaching stage. The shaft position for the gripper to be fully closed is assigned in such a way that the effective length of the steel string is long enough for the gripper to stay closed in any arm configuration. However, an attempt to drive the arm with a fully open gripper may result in closing or further opening of the gripper. Although the former causes no serious problem, further opening of the gripper will cause damage to the gripper or even the arm as the force resulted from the tension of the steel string acts on the joints of the manipulator.

In normal operation, the gripper will remain closed unless instructed to open when comes to a target object.

Therefore, the change of arm configuration is usually small during the period when the gripper is open. Moving the manipulator with an open gripper has been dealt with in designing software control and the possibility of occurrence of previously mentioned cases are eliminated.

2.2 THE ROBOT COMPUTER

(a) The Host Computer

The host computer for the Tasrobot0 system is an IBM/PC microcomputer. The computer uses an intel 8088 16-bit microprocessor and a 4 MHz clock. The computation speed of the host computer is greatly improved by installing the intel 8087 arithematic co-processor. It is also equipped with a commercially available interfacing board called Lab-Master Board provided by the Scientific Solution Inc.. In addition to the conventional analog-to-digital converter (ADC) and digital-to-analog converter (DAC), the Board provides timing signals through the AM9513 System Timing Controller and digital interface through the 8255 Programmable Parallel Port Interface (PPI).

The board can be programmed by using the accompanied software package - the Labpac Subroutines - from the Tecmar Inc., which are specially written to handle the hardwares in the Lab-Master Board. However, the Lab-Master Board provides only two channels for D/A conversion, which are obviously not enough to control the five axes motion of the manipulator; hence, part of the hardware interface were designed and built.

The Labpac software provides only 16-bit input and 8-bit output in the digital-to-digital interface subroutines and cannot be used for controlling the additional hardwares. Software interfacing subroutines in 8088 assembly language were developed to control the addition hardwares and to provide special functions in manipulator control. These subroutines will be discussed in Chapter 7.

(b) The Controller Unit

In the controller unit, two types of controllers, servo and non-servo, are used. Here, the positioning of the joints and the gripping action of the gripper are monitored by the command signals recieved from the host computer.

Three position feedback controllers are used in the controller unit for controlling the positions of the three joints. Trading off with its simpler circuitry and much lower price, position feedback controller has disadvantage of no control in the motion velocity of the joint and will result in a basically point-to-point robot.

The gripper controller is classified as non-servo. The feedback signal from the gripper motor does not go directly into the controller itself, instead, it is sampled by the host computer which then generates suitable commands to control the gripper motor as shown in Figure (2.2b-1). The overall control of the gripping action is however closed-loop, although it appears to be open-loop in the controller unit.

Command signals from the host computer, controlling the gripping action, consists of two bits. The controller will

react to these two bits to turn the motor on in one or the other direction or to turn the motor off.

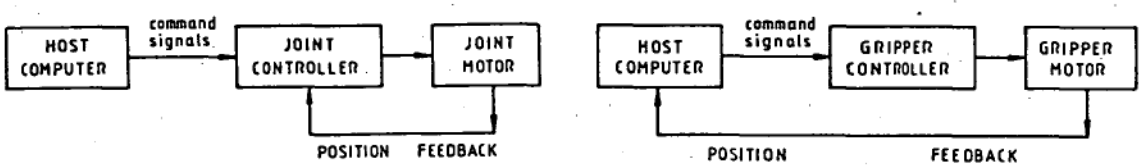


Figure (2.2b-1): Control System of (a) Joint (b) Gripper

The pitch and the roll rotations are provided by motion of two stepper motors as shown in Figure (2.1b-1). Positions of the two wrist rotation axes are absolute provided that the robot arm is properly set to the reference position before commencing. Command signals for each axial motion are position codes of 12 bits. Although there is no feedback signal in the wrist controller, its principle of operation is similar to servo type controllers in which the axial motion depends on the resulting error signal.

The feedback signal for each wrist axis is provided by a 12-bit counter which acts as a dynamic memory device and stores the previous position of that wrist axis in code. The magnitude of the error signal generates the number of clocking pulses required to move the current position of the wrist joint to a desired position while the sign of the error signal controls the direction of rotation of the motion. Clocking signals for driving the stepper motors are also used to renew the counters simultaneously.

However, when the controller unit is first energized, the contents of the counters are arbitrary and so are the positions of the two wrist axes. In order that the contents

of the counters can represent the positions of the wrist joints, the wrist joints must be moved to pre-defined positions, called reference positions, and the contents of the counters must be assigned pre-defined values, called reference codes, corresponding to the reference positions. Since there are no sensors mounted on the actual rotating shafts of the wrist joints, the computer cannot identify the reference positions. The reference positions of the wrist joints will have to be set manually before setting the contents of the counters. When the reference positions and reference codes are set, the counters will function like absolute shaft encoders of the two rotating wrist axes provided no loss of steps of the stepper motors occurs.

Since the two stepper motors must be energized to perform one axis of rotation, the controller unit is designed so that only one axis of rotation is implemented at a time.

2.3 POWER SOURCE

The power source of the Tasrobot0 arm are electric actuators. There are four permanent magnet dc motors and two stepper motors.

Three of the dc motors are used to drive the positioning joints and the other is to provide gripping action. The high speed but low torque characteristics of dc motors does not make them prevalent for driving loads directly. In practice, higher driving torque is achieved by trading off the high speed of the motor through speed reduction gearings. Thus, the torque used for driving a load is N

times that developed from motor, but the driving speed is reduced to 1/N times the motor speed.

The stepper motors used are 7.5° stepper motors each having four 12V dc windings and permanent magnet rotor construction. Each motor is equipped with a gear box which reduces the output step angle to 0.6° and decreases the maximum step rate to about 20 steps/sec (or 12°/sec) to achieve a maximum output torque of the motor.

2.4 PROGRAMMING THE TASROBOT0 MANIPULATOR

The programming of the Tasrobot0 manipulator is done through the use of a teach arm as shown in Figure (2.4-1).

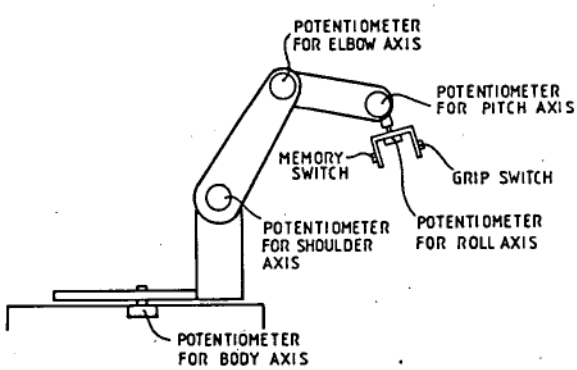


Figure (2.4-1): The Structure of The Teach Arm of The Tasrobot0 System

In teach mode, the configuration of the actual arm will follow that of the teach arm. Therefore, grasping the end-effector of the teach arm will be similar to grasping the end-effector of the actual arm, better still, the teach arm can easily be moved to take any configuration. Desired configuration of the actual arm can be recorded by pressing a memory switch located at the end-effector of the teach arm.

There are five potentiometers and two microswitches in the teach arm. One potentiometer is installed in each rotating joint to indicate the position of that joint. The GRIP-switch is to signal the robot computer to open or close the gripper. The MEM-switch is to signal the robot computer to record the current joint coordinates.

During teaching stage, the robot computer samples signals from sensors on the teach arm and drives the actual manipulator accordingly at the same time. The GRIP-switch has two functions. Not only does it tell the computer to open or close the gripper of the actual arm, it also allows the user to indicate how much the gripper motor must be turned to open the gripper in a specific arm configuration with respect to how long the GRIP-switch is held pressed.

Programming a robot for a task using a teach arm is simple and convenient since the operator does not need to measure the Cartesian coordinates in space as required by off-line programming, nor to worry about the position of each individual joint of the robot as required in manual teaching using teach pendant. In addition, the control program for the teaching process is much simpler as no mathematics for coordinate transformations are required.

CHAPTER THREE

HARDWARE DESIGN OF THE TASROBOT0 CONTROLLER UNIT

The hardwares for the controller unit consist of three printed circuit boards (PCBs) and a power rectifier unit. The three PCBs are identified as the analog control board, the digital control board and the interface circuit board.

The analog control board mainly controls the analog devices, i.e. the four dc motors of the manipulator. The digital control board controls the two stepper motors. And the interface circuit board provides communication between the control boards and the host computer. The dc power required by each board is provided by the rectifier unit which converts 240V ac mains power to unregulated dc voltages of ±20V and +11V. Regulated dc power required by active components used on each board is provided by its onboard regulators. Figure (3-1) illustrates a schematic structure of the Tasrobot0 controller unit.

The complete circuit diagrams for the controller unit with pin-to-pin configurations are shown in Appendix A. The designed ciruits were built and used as the control hardwares of the Tasrobot0 system.

3.1 THE ANALOG CONTROL BOARD

The analog control circuit was designed to control the four dc motors driving the Tasrobot0 manipulator, three of which control the motion of the three positioning joints and

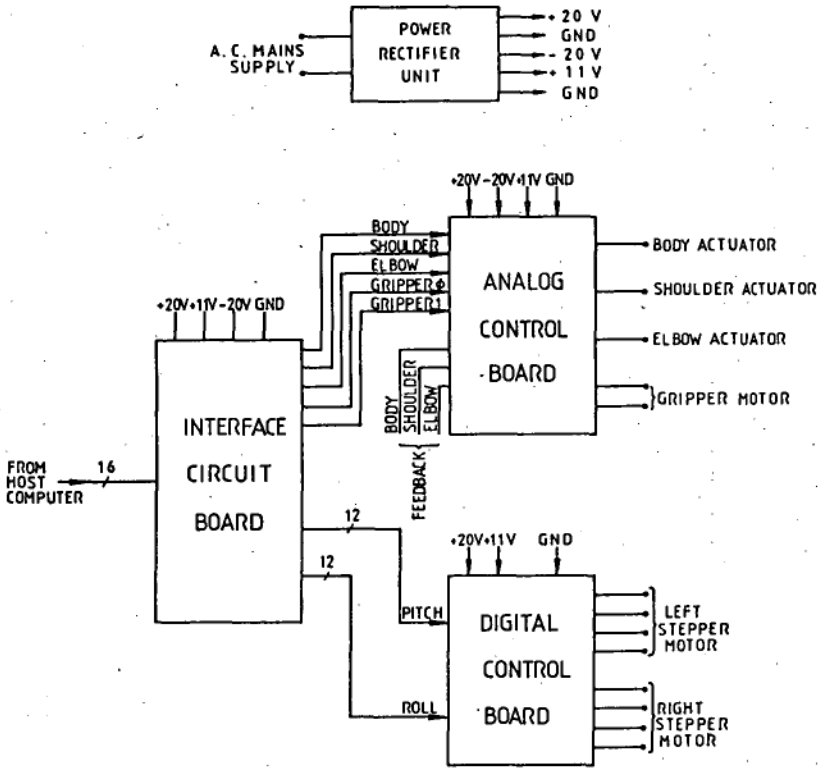


Figure (3-1) : Schematic Diagram of The Tasrobot0 Controller Unit

one controls the gripping action of the two-jaw gripper. The former are closed-loop control while the latter is closed-loop control via the host computer. Figure (3.1-1) illustrates a schematic layout of the board.

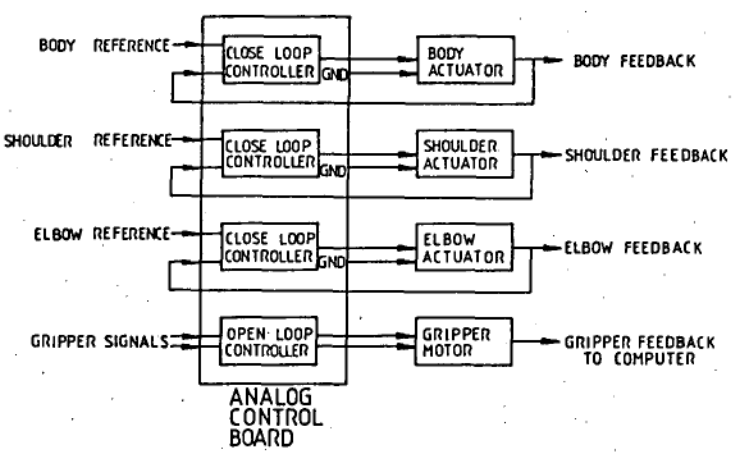


Figure (3.1-1): Schematic Layout of The Analog Control Board

(a) The Closed-loop Controller

The three closed-loop controllers closely resemble each other and their typical structure is illustrated in Figure (3.1a-1).

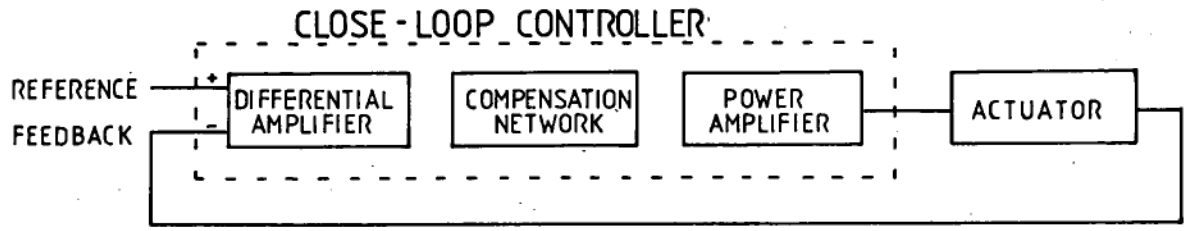


Figure (3.1a-1): A Typical Structure of A Closed-loop Controller of The Tasrobot0 System

The differential amplifier block was implemented by an instrumentation amplifier as shown in Figure (3.1a-2). By choosing $R_1=R_2=R_3=R_4$; $R_5=R_7=R$ and $R_6=AR$, the output of the amplifier becomes:

$$V_o = -(V_{REF1}-V_{REF2})(1+2/A) \qquad (3.1a-1)$$

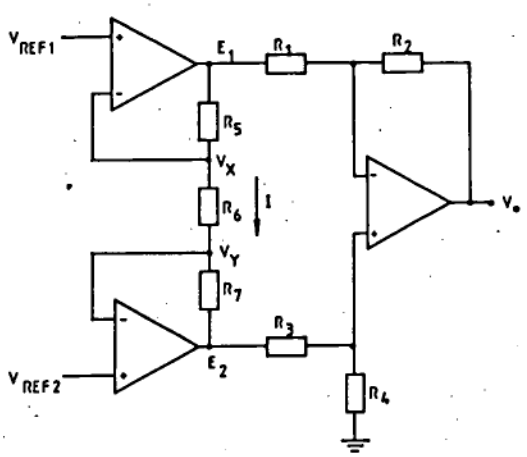


Figure (3.1a-2): An Instrumentation Amplifier

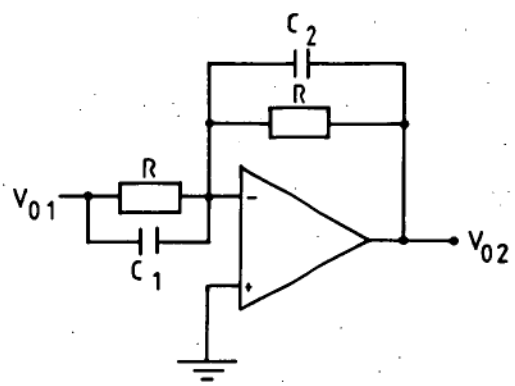


Figure (3.1a-3): The Compensator Circuit

The compensation network block is a simple one-pole-one-zero compensator and was implemented as shown in Figure (3.1a-3). The input/output relationship is:

$$\frac{V_{o2}}{V_{o1}} = \frac{-(sRC_1+1)}{(sRC_2+1)} \qquad (3.1a-2)$$

The power amplifier block was implemented by a type 165 Power Op-amp and the circuit is shown in Figure (3.1a-4).

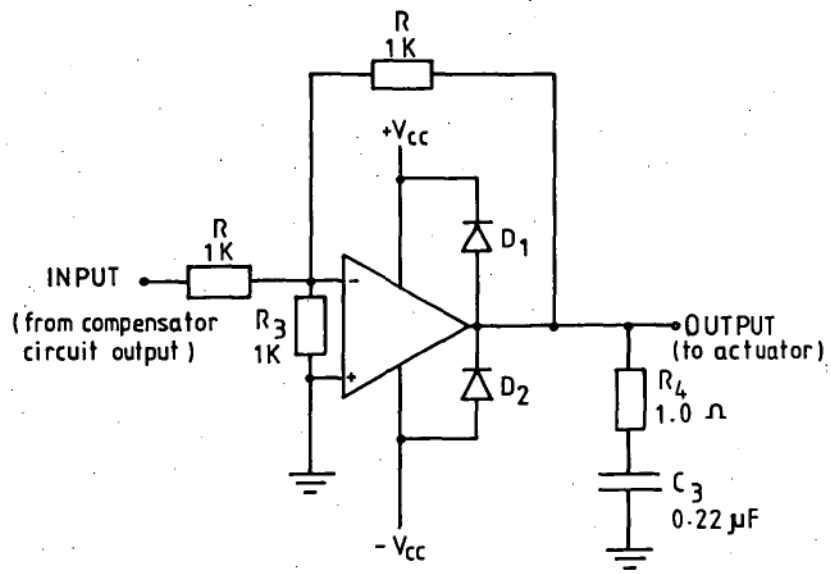


Figure (3.1a-4): A Power Op-amp Circuit Using Type 165 Op-amp

These three circuits form the closed-loop controller for the positioning joints of the Tasrobot0 manipulator.

(b) The Open-loop Controller

The gripper motor is controlled in closed-loop via the host computer. Command signals from the host computer directly control the motion of the gripper motor. A 2-bit signal is used for controlling the three possible states: rotating forward, backward or stop. The implementation of the control scheme merely requires a suitable power amplifier to drive the motor correspond to the 2-bit command signal.

The control circuit for the gripper motor is shown in Figure (3.1b-1). A general purpose light emitting diode (LED), with a voltage drop of about 2V across, was used to provide reference to the 2-bit TTL command signal from the host computer. A gain of 10 in each amplifier allows the

output to be pulled up to its maximum or minimum saturation voltage, thus enabling higher initial torque and faster operation of the motor.

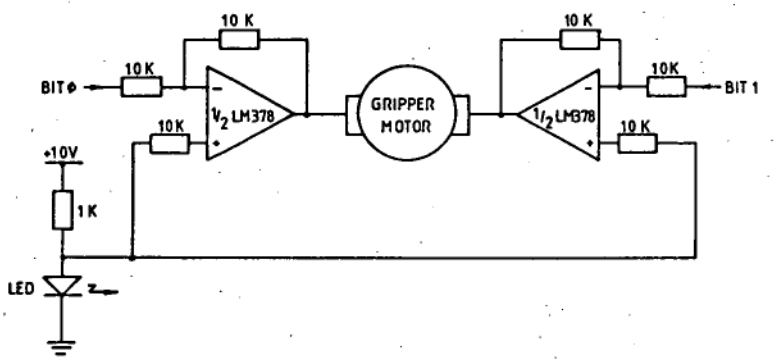


Figure (3.1b-1): The Gripper Control Circuit

Table (3.1b-1) summarizes the action of the gripper in relation to the status of the two command signals (BIT0 and BIT1).

BIT 0	BIT1	GRIPPER MOTOR	GRIPPER JAWS
0	0	UNENERGIZED	NO ACTION
0	1	CLOCKWISE ROTATION	OPENING JAWS
1	0	ANTICLOCKWISE ROTATION	CLOSING JAWS
1	1	UNENERGIZED	NO ACTION

Table (3.1b-1): Actions of Gripper Motor And Gripper Jaws With Respect To The States of The 2-bit Command Signal

3.2 THE DIGITAL CONTROL BOARD

The digital control circuit was designed to control the two stepper motors which drive the two wrist rotations of the Tasrobot0 manipulator. The control schemes for the two wrist rotations are similar in structure and is illustrated in Figure (3.2-1).

Unlike the dc servo motor controller, the wrist controller does not have feedback directly from correspond-

ing moving shaft. The feedback signal is from a position register which simulates a shaft encoder.

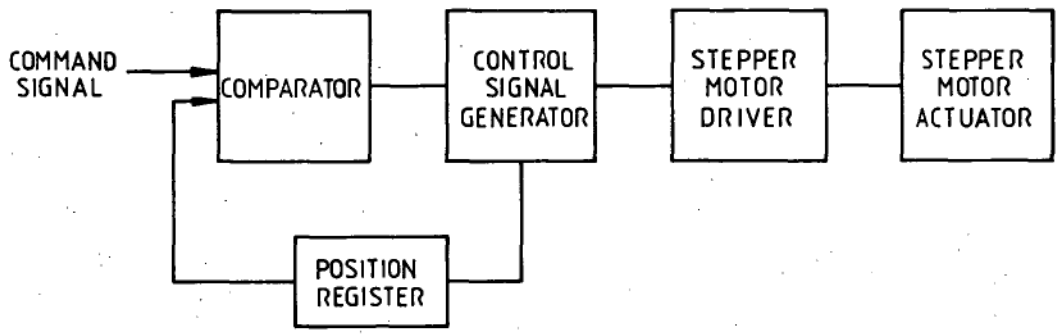


Figure (3.2-1): Block Diagram of The Control Scheme For A Stepper Motor

Initially, this position register is assigned a value corresponding to a certain position of the rotating axis. When a step is sent to rotate the corresponding shaft clockwisely, the position register will be incremented by one. Similarly, if a step is sent to rotate the shaft anticlockwisely, the position register will be decremented by one.

While pitch and roll rotation cannot occur simultaneously, the controller logic unit will only allow one rotation at a time with priority given to pitch rotation.

The control scheme can be implemented by software technique through computer programming or by hardware technique through digital electronic circuits. The former technique was not used because its resulted wrist rotations are comparatively slow and hence substantial amount of computer time will be required to generate wrist rotations. Also, during such wrist rotations, the computer would not be able to generate commands for motion of the other joints; and very slow overall motion of the manipulator will be expected. Although software technique was not used, the flow-chart of the control algorithm shown in Figure (3.2-2)

helps illustrating the hardware design of the control scheme.

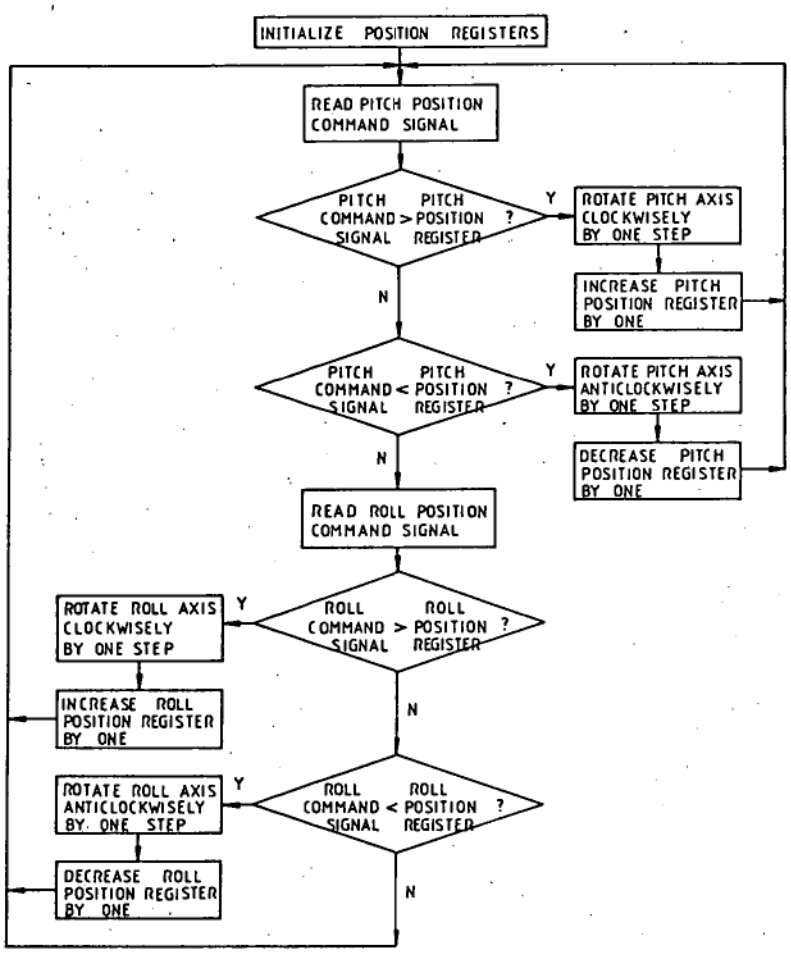


Figure (3.2-2): Flow-chart of The Wrist Control Algorithm

In the implementation of the stepper motor control scheme, the control blocks shown in Figure (3.2-1) were replaced by hardware functional blocks shown in Figure (3.2-3).

Since a step angle for the stepper motor is 0.6°, a minimum of 600 discrete levels or a 10-bit decoder is required to unambiguously represent the position of the wrist joint shaft . In this design, a 12-bit decoder was used to provide greater adaptability of the circuit. For example, if the step angle is now changed to 0.09° to

provide larger output torque, about 6 times that of 0.6°, the same circuit can be used without modification.

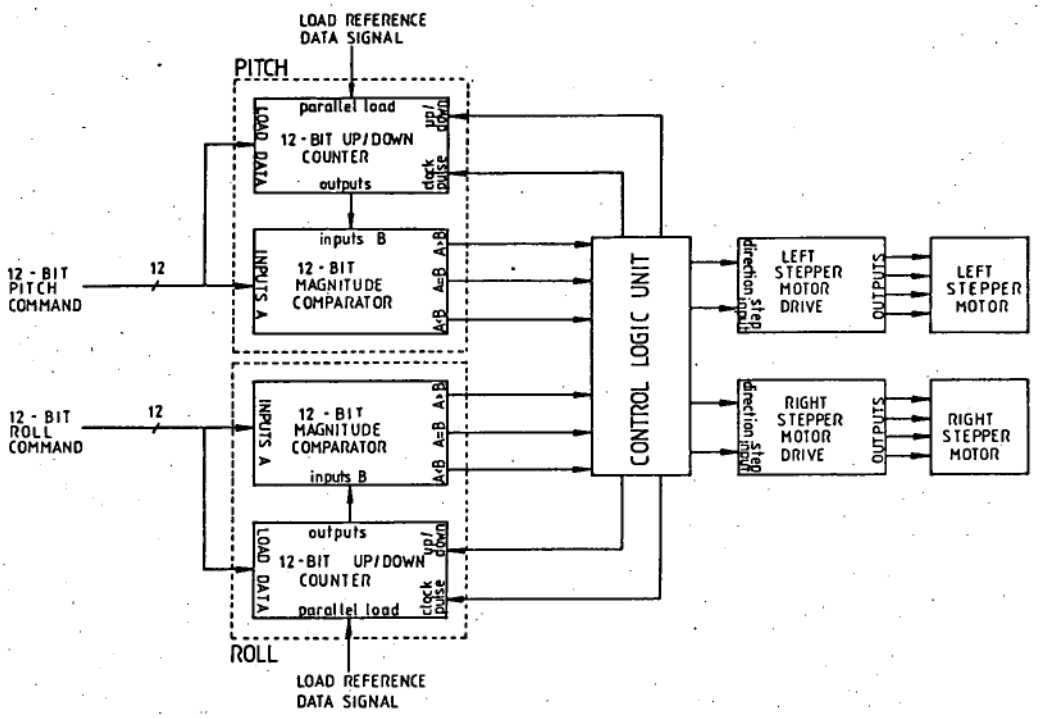


Figure (3.2-3): Hardware Block Diagram of The Digital Control Scheme

The position register block in the control scheme was implemented by a 12-bit Up/Down counter. Command data from the host computer is compared with the counter output, the latter represents the shaft position of the corresponding wrist joint. Compared results were used to signal the control logic unit to generate appropriate signals to drive the stepper motors and to update the counters. The control logic unit also distinguishes pitch or roll rotation and gives priority to the former.

(a) <u>The 12-Bit Comparator</u>

Two 12-bit comparators are required, one for each wrist rotation. Each comparator was implemented by cascading three 74C85 4-bit magnitude comparators and is illustrated in Figure (3.2a-1).

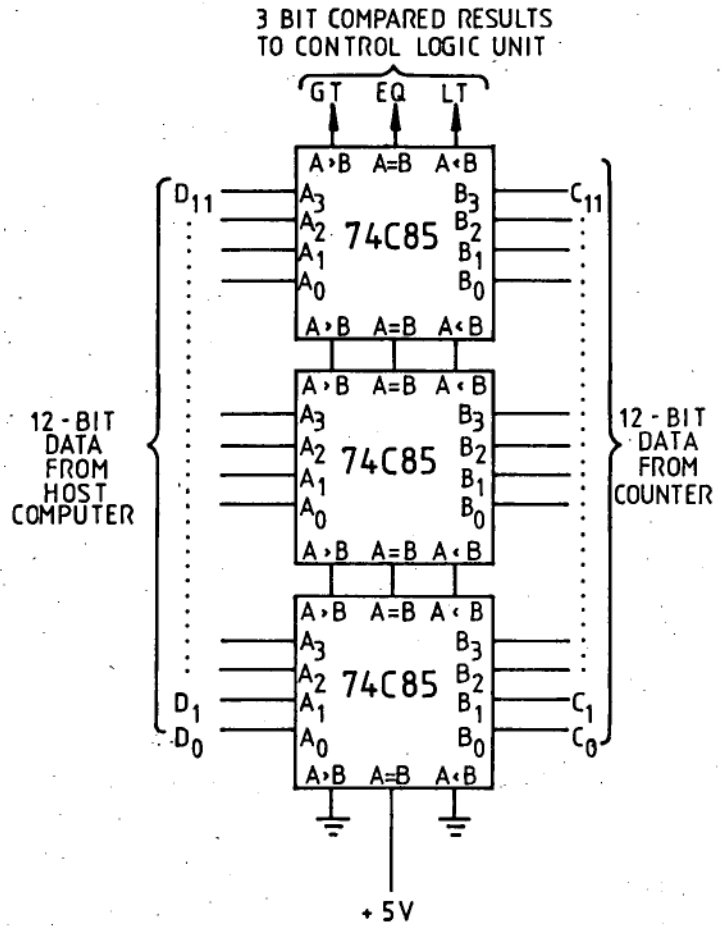


Figure (3.2a-1): A 12-bit Magnitude Comparator

(b) The 12-bit Up/Down Counter

The control scheme requires two 12-bit up/down counters, one for each wrist rotation. The counters act like position indicators of the current shaft positions. Pulses generated to move the stepper motors also update the counters in such a way that the counter outputs will be counted up or down towards the value of the command position codes. To enable the position reference data to be set, the counters must be presettable and therefore, 74LS191 Presettable 4-Bit Binary Up/Down Counters were used. Three 74LS191 counters were cascaded to give a 12-bit counter and is shown in Figure (3.2b-1).

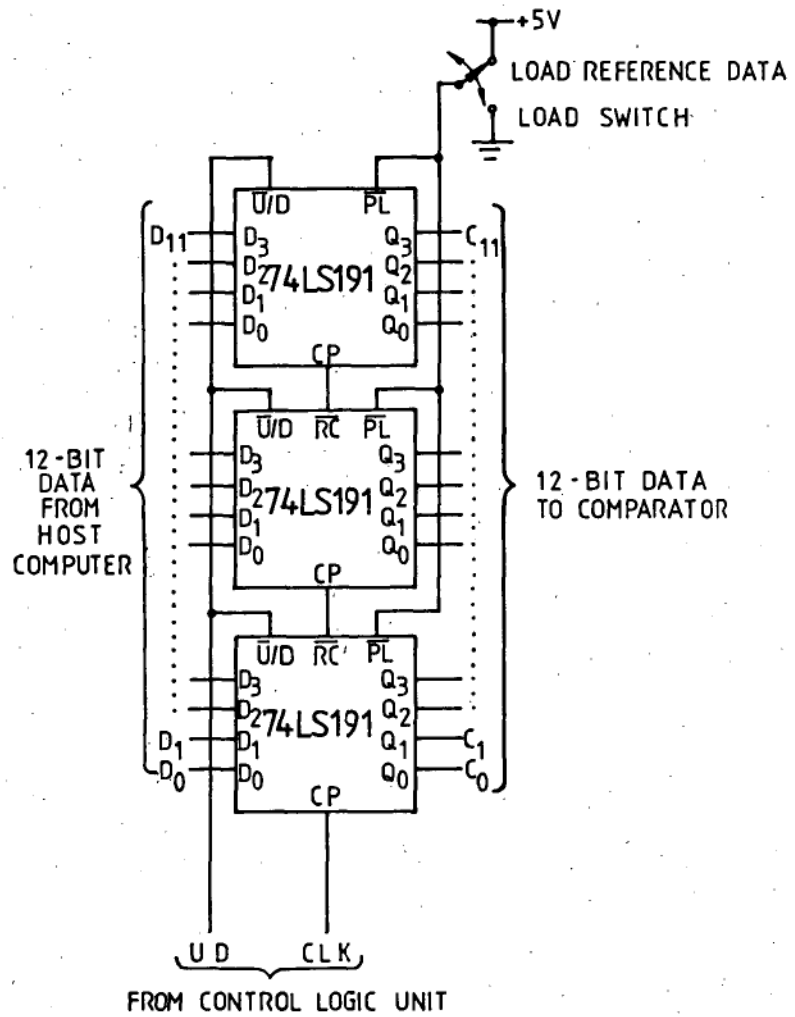


Figure (3.2b-1): A 12-bit Up/Down Counter

(c) The Stepper Motor Driver

A suggested stepper motor driver IC for the stepper motor is the SAA1027. A typical circuit for the driver is shown in Figure (3.2c-1). A Low-to-High transition on the STEP pin turns the motor by one step. The sequence of output signals and the direction of rotation of the motor depend on the signal level of the Direction pin (DIR).

It is important to note that logic high level of the SAA1027 is represented by a voltage level between 7.5V to 12V and low is represented by a voltage level between 0V to 4.5V.

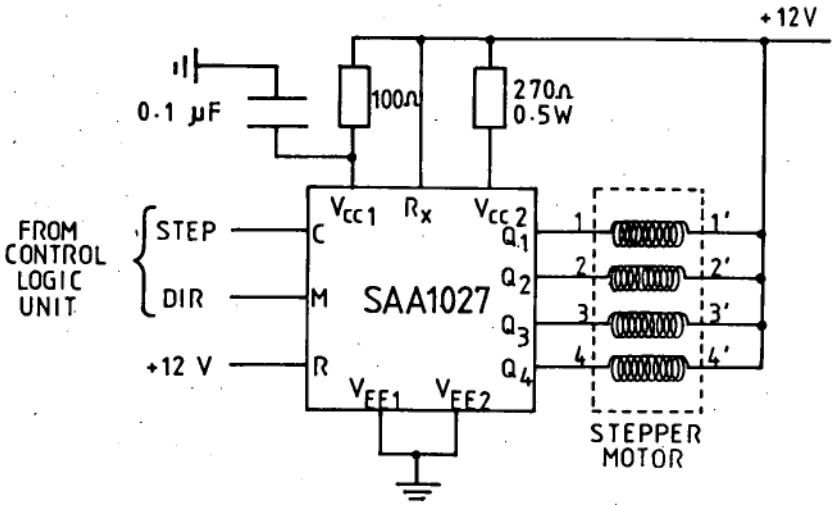


Figure (3.2c-1): Circuit Diagram For The Stepper Motor Driver SAA1027

(d) The Control Logic Unit

The control logic unit is the heart of the control scheme. It generates TTL logic level voltages to update the counters. It generates appropriate logic level voltages to control the stepper motor drivers in driving the stepper motors, with respect to the six resulting signals from the two comparators. The six input signals are arranged in two groups of three, one group from one comparator.

There are four pairs of output signals in the control logic unit. Each pair of output signals consists of a clocking signal and a direction control signal. Two pairs of output signals are TTL logic level voltages for controlling the two 12-bit counters. The other two pairs of output signals are special logic level voltages for controlling the two stepper motor drivers. Figure (3.2d-1) shows the sources of the input signals and the destinations of the output signals. The symbols used in the figure will carry their meanings throughout the rest of this section.

Figure (3.2d-1) also illustrates the internal structure of the control logic unit. Functionally, the unit can be

divided into three parts, namely, the clock generator, the pulse generator and the direction signal generator.

The clock generator outputs a continuous TTL clock signal with a constant frequency. This clock signal is fed into the pulse generator to provide clocking signal to the two counters and the stepper motor drivers. The clocking frequency is about 20Hz providing maximum working torque for the motors. The direction signal generator generates appropriate signals to up-count or down-count the counters and to rotate the stepper motor in either directions.

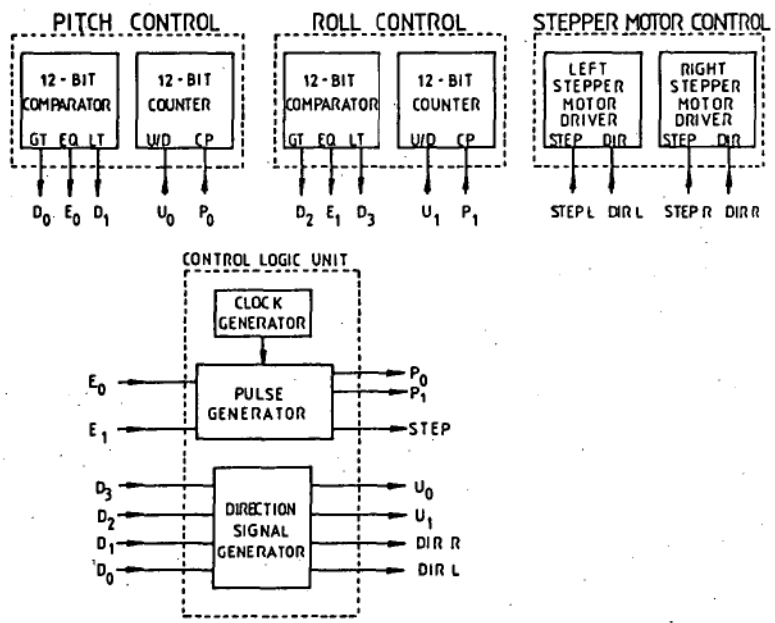


Figure (3.2d-1): Block Diagram of The Input Sources And Output Destinations of The Control Logic Unit

(i) The clock generator

The clock generator of the control logic unit is a free-running multivibrator as illustrated in Figure (3.2d(i)-1).

With $R_1 = R_2 = R_4 = R$; $R_3 >> R_5$ and $R_5 << R$, the frequency of oscillation f_{osc} can be shown as:

$$f_{osc} = 1/1.386R_3C \qquad (3.2d(i)-1)$$

Resistor R_3 was selected as 22k and C as 1.5μF, and the resultant frequency of oscillation is about 22Hz.

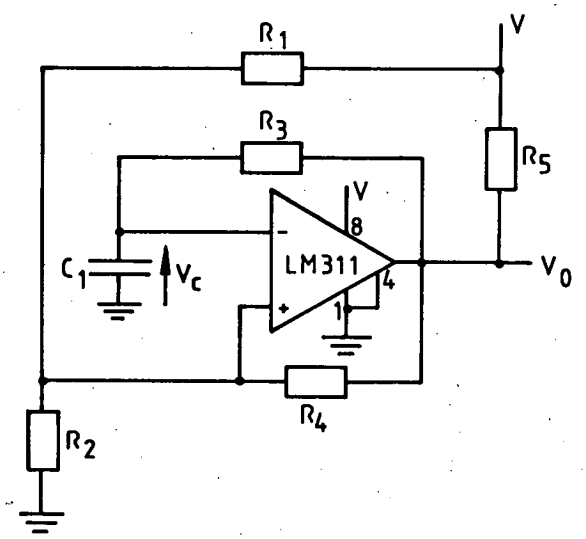


Figure (3.2d(i)-1): The Clock Generator Circuit

(ii) The pulse generator

The main functions of the pulse generator are to provide non-TTL signal, STEP, to the stepper motor drivers to drive the stepper motors; and to provide TTL signals, P_0 and P_1, to activate the counting function of the two counters.

The non-TTL signal, STEP, will be activated unless both the EQ signals from the two comparators are high. Therefore, only the input signals, E_0 and E_1, of the control logic unit are required to implement clocking control of the stepper motors.

The clocking source of the pulse generator is the clock generator. By enabling or disabling the clocking source, the outputs of the pulse generator, P_0 and P_1, will activate or deactivate counting function of the counters. Since only one type of rotation is allowed at a time and higher priority is given to pitch rotation, clocking signal for roll rotation

counter, P_1, will only be enabled when the specific pitch rotation has completed.

INPUTS			OUTPUTS		
E_0	E_1	CLK**	P_0	P_1	STEP*
0	0	0	0	1	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

NOTE :
0 = LOW LOGIC LEVEL
1 = HIGH LOGIC LEVEL
* HIGH LOGIC LEVEL IS +12 V AND LOW LOGIC LEVEL IS 0 V
** CLK IS THE SIGNAL FROM THE OUTPUT OF THE CLOCK GENERATOR

Table (3.2d(ii)-1): Truth Table For The Outputs of The Pulse Generator

CLK \ E_0E_1	00	01	11	10
0	010	010	111	100
1	111	111	111	111

Figure (3.2d(ii)-1): Karnaugh Map For The Output Signals P_0 P_1 STEP, of The Pulse Generator

The required truth table for the signals P_0, P_1 and STEP is shown in Table (3.2d(ii)-1). The karnaugh map for the output signals is illustrated in Figure (3.2d(ii)-1). The logical expression for each output is:

$$P_0 = E_0 + CLK \qquad (3.2d(ii)-1)$$

$$P_1 = \overline{E_0} + E_1 + CLK \qquad (3.2d(ii)-2)$$

$$STEP = E_0 E_1 + CLK \qquad (3.2d(ii)-3)$$

The circuit for the pulse generator is shown in Figure (3.2d(ii)-2).

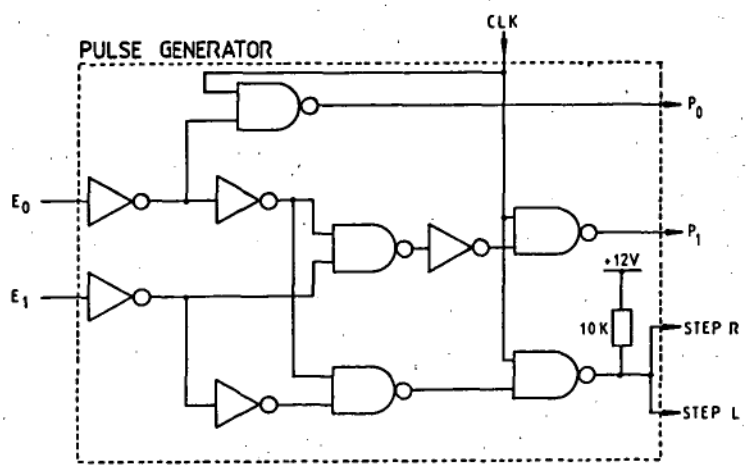


Figure (3.2d(ii)-2): Circuit Diagram of The Pulse Generator

(iii) The direction signal generator

The direction signal generator provides two types of signals: non-TTL type signals, DIRL and DIRR, to control the type of rotation, pitch or roll, as well as its direction; and TTL type signals, U_0 and U_1, to control the counting direction of the counters.

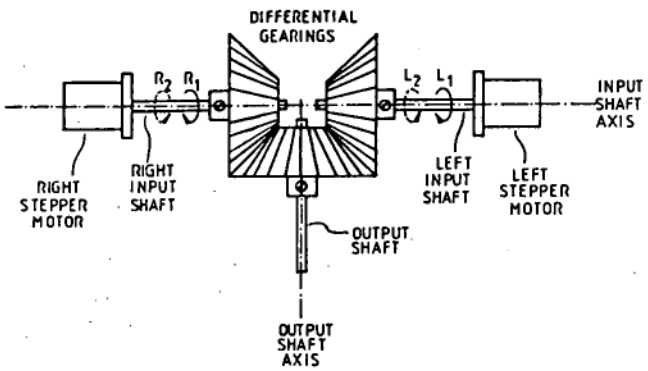


Figure (3.2d(iii)-1): Motions of The Input Shafts And The Output Shaft of The Wrist

LEFT INPUT SHAFT MOTION	RIGHT INPUT SHAFT MOTION	OUTPUT SHAFT MOTION
L_1	R_1	PITCH CLOCKWISE
L_1	R_2	ROLL CLOCKWISE
L_2	R_1	ROLL ANTICLOCKWISE
L_2	R_2	PITCH ANTICLOCKWISE

Table (3.2d(iii)-1): Types And Directions of Motion Referring to Figrue(3.2d(iii)-1)

The axes of rotation of the two stepper motors are aligned in one axis with differential gearings as shown in Figure (3.2d(iii)-1). Two non-TTL signals, DIRL and DIRR, from the direction signal generator, control the two stepper motor drivers. With two possible directions of motion for each motor, the output shaft has four different types of resulting motion. Taking L_1, L_2, R_1 and R_2 to be the directions of motion of the two input shafts as shown in Figure (3.2d(iii)-1), the resulting motion of the output shaft is shown in Table (3.2d(iii)-1). The definitions of directions of pitch and roll rotations are shown in Figure (3.2d(iii)-2). The required DIRL and DIRR signals to achieve various wrist motions are shown in Table (3.2d(iii)-2).

The additional requirement to give higher priority to pitch rotation is achieved by generating the two direction control signals, DIRL and DIRR, for pitch rotation first.

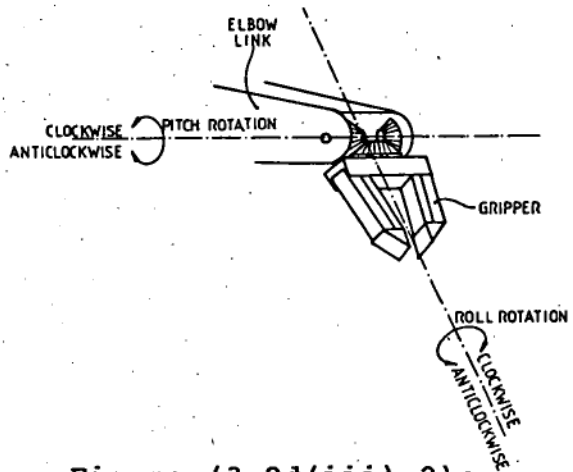


Figure (3.2d(iii)-2): Definitions of The Wrist Rotations

DIR L	DIR R	WRIST MOTION
0	0	PITCH CLOCKWISE
0	1	ROLL CLOCKWISE
1	0	ROLL ANTICLOCKWISE
1	1	PITCH ANTICLOCKWISE

Table (3.2d(iii)-2): Wrist Rotations With Respect To The Direction Control Signals

The truth table for the signals DIRL and DIRR is shown in Table (3.2d(iii)-3). The karnaugh maps for the two signals DIRL and DIRR are shown in Figure (3.2d(iii)-3); and the logical expressions for the two signals are:

$$DIRL = D_0 + \overline{D_1} D_2 \qquad (3.2d(iii)-1)$$

$$DIRR = D_0 + D_3 \overline{D_1} \qquad (3.2d(iii)-2)$$

Two TTL type signals, U_0 and U_1, are generated by the direction signal generator for controlling counting direction of the two counters. Unlike the non-TTL signals, U_0 and U_1 are not governed by the priority requirement. This allows the signals U_0 and U_1 to be generated in a simpler scheme as:

$$U_0 = \overline{D_1} \qquad (3.2d(iii)-3)$$

$$U_1 = \overline{D_3} \qquad (3.2d(iii)-4)$$

The overall circuit diagram for the direction signal generator is shown in Figure (3.2d(iii)-4). Open-collector

NAND gates are used to provide non-TTL signals, DIRL and DIRR, with a high level of +12V and a low level of 0V.

D_3	D_2	D_1	D_0	DIRL	DIRR	REMARKS
0	0	0	0	X	X	BOTH ROTATIONS COMPLETED
0	0	0	1	1	1	PITCH ANTICLOCKWISE ROTATION
0	0	1	1	0	0	PITCH CLOCKWISE ROTATION
0	0	1	1	X	X	DO NOT OCCUR
0	1	0	0	1	0	ROLL CLOCKWISE ROTATION
0	1	0	1	1	1	(PRIORITY) PITCH ANTICLOCKWISE ROTATION
0	1	1	0	0	0	(PRIORITY) PITCH CLOCKWISE ROTATION
0	1	1	1	X	X	DO NOT OCCUR
1	0	0	0	0	1	ROLL ANTICLOCKWISE ROTATION
1	0	0	1	1	1	(PRIORITY) PITCH ANTICLOCKWISE ROTATION
1	0	1	0	0	0	(PRIORITY) PITCH CLOCKWISE ROTATION
1	0	1	1	X	X	DO NOT OCCUR
1	1	0	0	X	X	
1	1	0	1	X	X	
1	1	1	0	X	X	
1	1	1	1	X	X	

NOTE: "1" = LOGIC HIGH LEVEL
"0" = LOGIC LOW LEVEL
X = DON'T CARE

Table (3.2d(iii)-3): Truth Table For The Control of The Wrist Rotations

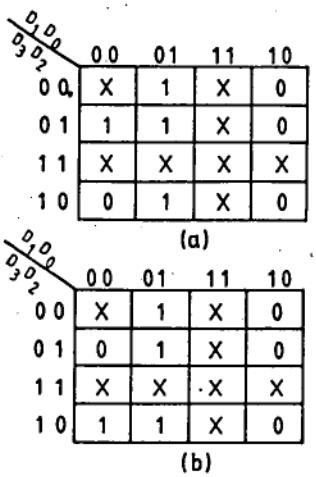


Figure (3.2d(iii)-3): Karnaugh Maps For (a) DIRL Signal And (b) DIRR Signal

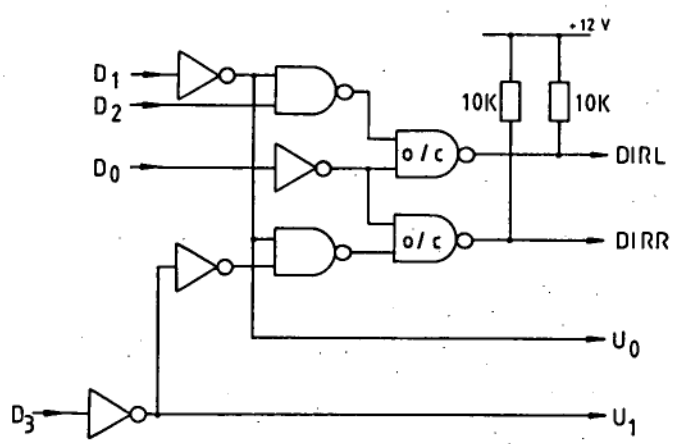


Figure (3.2d(iii)-4): Circuit Diagram of The Direction Signal Generator

3.3 THE INTERFACE CIRCUIT BOARD

In the Tasrobot0 system, the major hardware interfacing was done by a commercially available interfacing board, the Lab Master Board, from the Tecmar Inc.. The board, located at addresses 0710H to 071FH, provides three major types of interface, namely, analog-to-digital, digital-to-analog and digital-to-digital interfaces. Each type of interface can be initiated by writing appropriate signals to its corresponding address shown in the manual[R32]. Software subroutines written for the board were provided by a commercially available package, the Lab-pac Subroutines, from the Scientific Solutions Inc.. Each of these subroutines was written for a specific function as described in the manual[R33].

The analog-to-digital interface of the Lab Master Board consists of an 8-bit multiplexer, giving 256 possible analog input channels. The DAC installed has a resolution of 12 bits. With the aid of the Lab-pac Subroutines, the input channels can be sampled at regular intervals with maximum sampling frequency as high as 1kHz. Regular sampling can be achieved by hardware interrupts generated by the System Timing Controller AM9513. All the analog input interfacing for the Tasrobot0 system was done through the Lab Master Board using Lab-pac Subroutines.

However, the Lab Master Board provides only two digital-to-analog channels which are not enough for controlling the three dc servo motors of the Tasrobot0 manipulator. Three digital-to-analog channels were designed. Each channel

is accessed through the data bus using the 8255 Programmable Parallel Port Interface (PPI) in the Lab Master Board.

The digital-to-digital interface of the Lab Master Board is provided by the 8255 PPI. The PPI occupies four address locations, one for command register and three for data registers. There are 24 programmable input/output channels in the PPI. The command register, located at 071FH, allows each of the 24 channels, which are arranged in three 8-bit ports: portA, portB and portC, to be programmed as either input or output channels. The input data or output data can be read or written through the data registers located at 071CH to 071EH for portA, portB and portC respectively.

PortA, the upper 8-bits of the 24 bits, was programmed as input port for manipulating digital input signals. At present, there are three one-bit digital input signals, one from the MEM-switch and one from the GRIP-switch, both mounted on the teach arm. The third signal is from the +5V power supply rail so that the power supply for the electronics in the Tasrobot0 controller unit can be examined.

PortB and portC which occupy the lower 16 bits of the 24 bits were programmed as output ports. Since there are six output devices, four dc motors and two stepper motors, to be controlled, six registers are required to latch data to appropriate devices. The upper 4 bits of the 16 bits output from the host computer were used to initialize one of the six registers using a 3-to-8 decoder, and the lower 12 bits were used as data bits. The interfacing circuit of the

Tasrobot0 system thus consists of six 12-bit data registers, one 3-to-8 decoder and three DACs as shown in Figure (3.3-1).

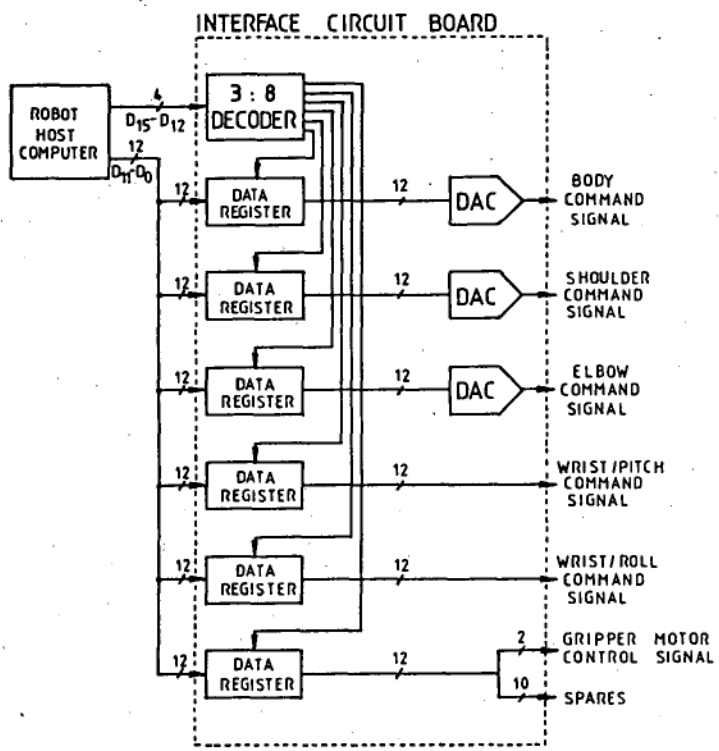


Figure (3.3-1): Layout of The Interface Circuit Board

(a) The Decoder and The Data Register Circuit

The 3-to-8 decoder and the six 12-bit data registers form the addresses of the output devices. Three of the data registers latch data for the three DACs to control the positioning motion of the manipulator while the remaining three provide digital control on the stepper motors and the gripper motor. The data signals for the five joint motors are 12-bit. The gripper motor only requires 2-bit data signals and 10 bits are spare in its data register.

Of the 16 bits from the computer data bus, four bits are used to generate the chip-select-signals: three of which decode the address of the six output devices and one enables the decoder.

The 74LS138 3-to-8 Decoder was used in the design. It decodes one-of-eight lines based on the conditions of the three binary select inputs: A, B and C; and the three enable inputs: G_1, G_{2A} and G_{2B}. When the decoder is enabled, one of the eight outputs is pulled down to low level according to the levels on the select inputs as shown in Table (3.3a-1).

Two 74LS174 Hex D Flip-Flops were used to form a data register circuit. Six registers are driven by the decoder, all having similar circuits as shown in Figure (3.3a-1) but driven by different output of the decoder as indicated in Table (3.3a-2).

INPUTS				OUTPUTS							
ENABLE	SELECT										
G_1	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
L	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	H	H	H	H	H	H	H
H	L	L	H	H	L	H	H	H	H	H	H
H	L	H	L	H	H	L	H	H	H	H	H
H	L	H	H	H	H	H	L	H	H	H	H
H	H	L	L	H	H	H	H	L	H	H	H
H	H	L	H	H	H	H	H	H	L	H	H
H	H	H	L	H	H	H	H	H	H	L	H
H	H	H	H	H	H	H	H	H	H	H	L

NOTE: "H" = LOGIC HIGH
"L" = LOGIC LOW
"X" = DONT CARE
G_{2A} AND G_{2B} ARE HELD LOW

Table (3.3a-1): Truth Table of The Decoder 74LS138

SELECT INPUTS OF DECODER			ENABLED OUTPUT OF DECODER	ENABLED DATA REGISTER
C	B	A		
L	L	L	Y_0	BODY
L	L	H	Y_1	WRIST / PITCH
L	H	L	Y_2	SHOULDER
L	H	H	Y_3	WRIST / ROLL
H	L	L	Y_4	ELBOW
H	L	H	Y_5	GRIPPER MOTOR
H	H	L	Y_6	NOT USED
H	H	H	Y_7	

Table (3.3a-2): Data Register Enabled With Respect To Select Inputs of The Decoder 74LS138

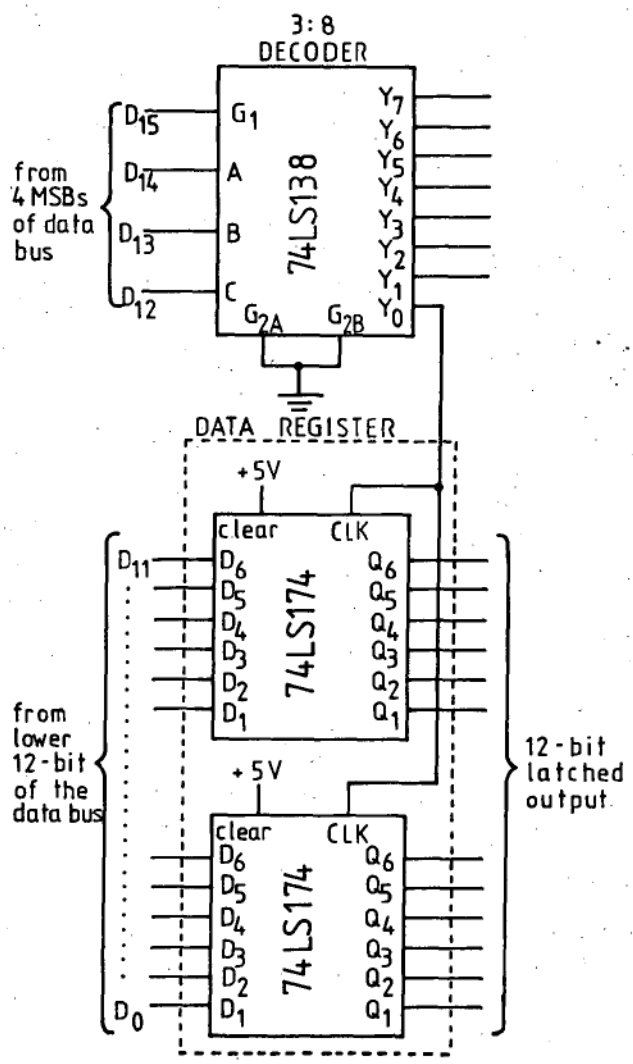


Figure (3.3a-1): Decoder And A Data Register Circuit

In order to avoid erronous decoding due to transient states of the signals on the select inputs, decoding is done in a sequence of three steps. Firstly, the 3-bit chip-select signals are sent to the decoder while the outputs of the decoder are disabled by holding G_1 low. Secondly, G_1 is changed to high with the select signals unaltered. This causes the corresponding output to go low. Thirdly, G_1 is switched back to low and a positive-going triggering signal for an appropriate data register is generated. The enable signal, G_1, and the 3-bit chip-select signals were generated in this sequence by softwares which programmes the 8255 with 8088 assembly language and will be discussed in Chapter 7.

(b) The DAC Circuit

Three DACs are required to provide analog signal commands to the three positioning joints. The three DAC circuits are identical, each consisting of an AD7541 12-Bit Monolithic Multiplying DAC plus external circuit to convert the binary weighted output currents to the equivalent binary weighted voltages.

For full four-quadrant multiplying D/A conversion, the AD7541 can be connected with two op-amps as shown in Figure (3.3b-1). The output voltage can be expressed as:

$$V_{OUTA} = (1-2D)V_{ref} \qquad (3.3b\text{-}1)$$

where

$$D = \frac{B_{11}}{2} + \frac{B_{10}}{2^2} + \cdots\cdots + \frac{B_0}{2^{12}} \qquad (3.3b\text{-}2)$$

The code and voltage output relationship is shown in Table (3.3b-1) which is an offset binary coded conversion.

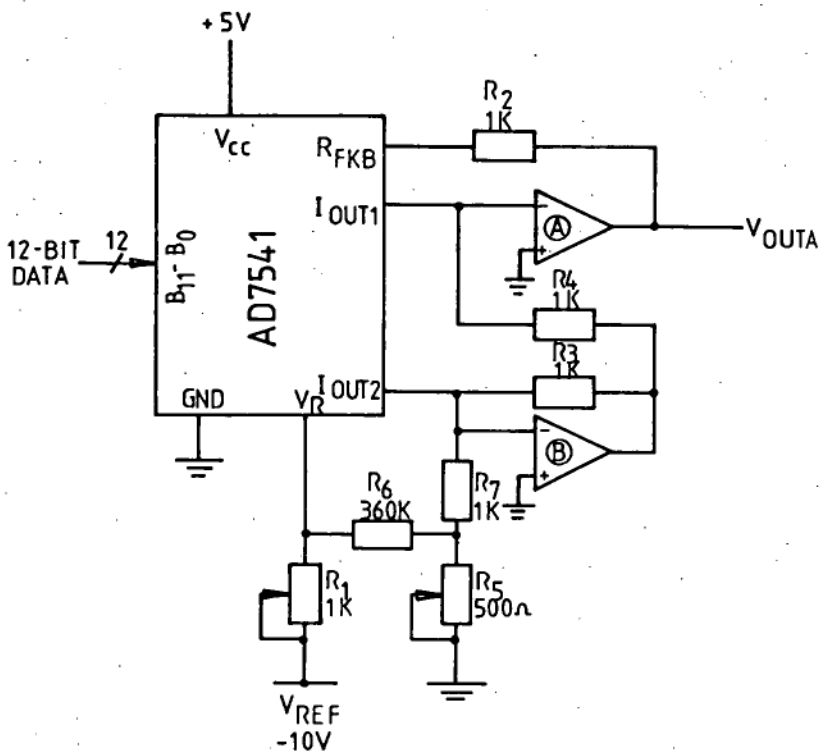


Figure (3.3b-1): Circuit Diagram For Bipolar Conversion of Offset Binary Codes

DIGITAL INPUTS	NORMINAL ANALOG OUTPUT
1 1 1 1 1 1 1 1 1 1 1 1	- 0.9 9 9 5 1 2 V_{REF}
1 0 0 0 0 0 0 0 0 0 0 1	- 0.0 0 0 4 8 8 V_{REF}
1 0 0 0 0 0 0 0 0 0 0 0	0
0 1 0 0 0 0 0 0 0 0 0 0	0.5 0 0 0 0 0 V_{REF}
0 0 0 0 0 0 0 0 0 0 0 0	1.0 0 0 0 0 0 V_{REF}

Table (3.3b-1): Offset Binary Code Conversion

DIGITAL INPUTS	NORMINAL ANALOG OUTPUTS
0 1 1 1 1 1 1 1 1 1 1 1	9.9 9 5 1 2 V
0 0 0 0 0 0 0 0 0 0 0 1	4.8 8 3 mV
0 0 0 0 0 0 0 0 0 0 0 0	0 V
1 1 0 0 0 0 0 0 0 0 0 0	- 5.0 V
1 0 0 0 0 0 0 0 0 0 0 0	- 10.0 V

Table (3.3b-2): Two's Complement Code Conversion With V_{REF}=-10V

The offset binary code is a natural consequence of the structure of the DAC. However, the two's complement code is preferred for binary mathematics. The DAC can easily be structured to convert the two's complement code to its equivalent voltage by inverting the MSB of the digital signal before it is connected to the MSB pin (B_{11}) of the DAC. The output voltage expression for the two's complement conversion can be written as:

$$V_{OUT} = (1-2F)V_{ref} \qquad (3.3b-3)$$

where,

$$F = \frac{B_{11}}{2} + \frac{B_{10}}{2^2} + \cdots + \frac{B_0}{2^{12}} \qquad (3.3b-4)$$

The code and voltage output relationship is shown in Table (3.3b-2).

3.4 THE POWER SUPPLY CIRCUIT DESIGN

The design circuit boards discussed so far have their own voltage regulators on board. Unregulated power are supplied to each board so that each piece of circuitry has an individual regulated supply. This arrangement has an advantage of avoiding disturbation to regulated supply via inevitably noise-voltage affected leads which increases the ripple factor of the regulated supply.

The power required by each circuit board and the regulators used are shown in Table (3.4-1). Circuit diagrams for connecting these regulators are standardized. The pin-to-pin configurations and the full board circuit diagrams are shown in Appendix A.

CIRCUIT BOARD	REGULATED VOLTAGE	TOTAL MAX.POSSIBLE CURRENT	SELECTED REGULATOR	UNREGULATED BUS
ANALOG CONTROL BOARD	+15 V	1.5 A	LM 340T-15	BUS2
	-15 V	-1.5 A	LM 320T-15	BUS3
	+10 V	10 mA	LM 78L-10A	BUS2
	-10 V	-10 mA	LM 39L- 5A	BUS3
DIGITAL CONTROL BOARD	+12 V	1.0 A	LM 340T-12	BUS2
	+ 5 V	280 mA	LM 340T-5A	BUS1
INTERFACE CIRCUIT BOARD	+15 V	12 A	LM 340T-15	BUS2
	-10 V	6 mA	LM 79L-5A	BUS3
	+ 5 V	240 mA	LM 340T-5A	BUS1

Table (3.4-1): Power Requirement For Each Circuit Board And The Regulators Used

The power supply circuit for the three circuit boards is illustrated in Figure (3.4-1). Three unregulated voltage buses are provided, they are BUS1, BUS2 and BUS3.

BUS1 has nominal voltage of +12V and provides power for +5V regulated supply; BUS2 and BUS3 have nominal voltages of +20V and -20V respectively and provide power to ±15V, +12V and ±10V regulated supply in the circuit boards. The current supply requirement for each regulator is also shown in Table (3.4-1).

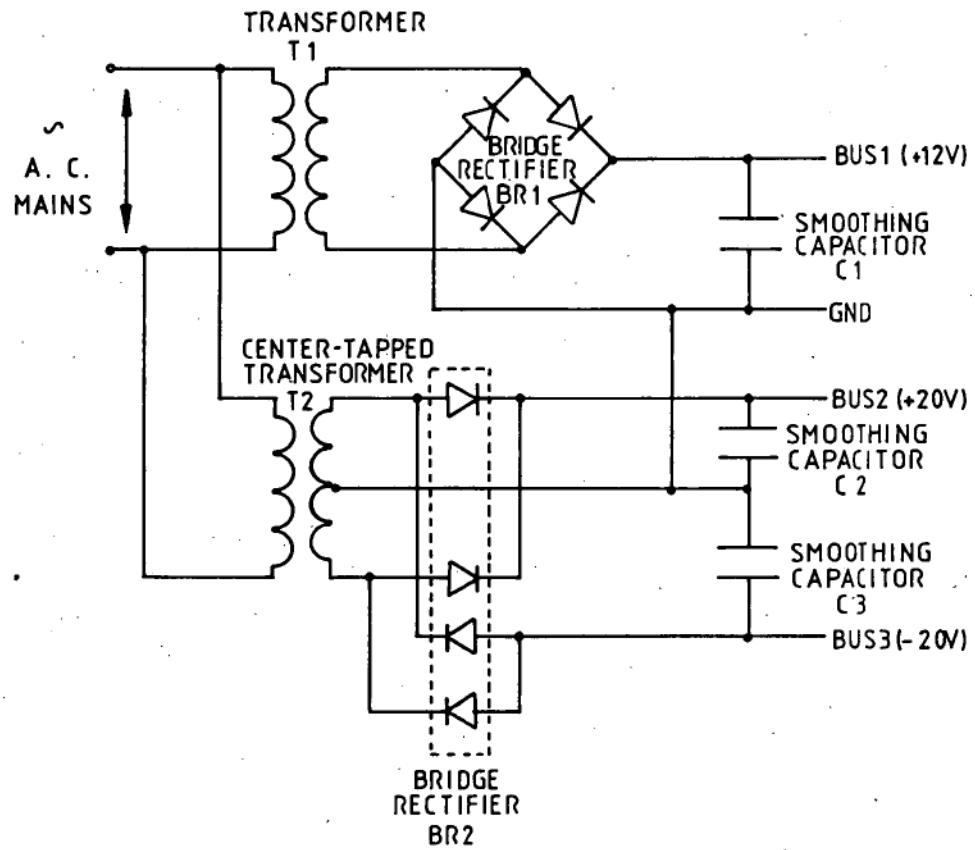


Figure (3.4-1): Power Supply Circuit For The Hardware Control Boards

The minimum requirements of the passive components used in the power supply circuit are shown in Table (3.4-2).

MINIMUM REQUIREMENTS ON SMOOTHING CAPACITORS	SELECTED C (allowing 50 % variation on C)	8952 μF	1432 μF
	RIPPLE FACTOR	4 %	8 %
	V_{RIPPLE} (PK)	1.13 V	1.13 V
	I_{RIPPLE} (allowing 100 % variation on C)	4.5 A	0.72 A
MINIMUM REQUIREMENTS ON BRIDGE RECTIFIERS (PER DIODE)	I_F(AV) / DIODE	1.25 A	0.25 A
	I_{FM} / DIODE	7.5 A	1.5 A
	I_F (RMS) / DIODE	2.5 A	0.5 A
	I_{SURGE} / DIODE	60 A	12.5 A
MINIMUM REQUIREMENTS ON TRANSFORMERS	V_{SEC} (RMS)	18.74 V	10.6 V
	R_S	0.5 Ω	1.2 Ω
	VA RATING	66 VA	7.5 VA

Table (3.4-2): Summary of The Design Process of The Unregulated Voltage Supply Buses And The Minimum Requirements of The Elements Used In The Power Supply Circuit

CHAPTER FOUR

IDENTIFICATION OF THE MANIPULATOR SYSTEM

A complete robot manipulator system is a non-linear multi-variable system. Upon analysing the dynamics of a robot manipulator using Lagrangian mechanics[R1], the torque acting on a manipulator joint is due to a combination of effective inertia and viscous friction of the joint and the effect of motion of other joints. The latter includes coupling inertia, centripetal acceleration and coriolis acceleration. However, the Tasrobot0 arm is only a test robot arm. Its size and weight are far less than commonly encountered working robots. The interactions between links are thus small. The effects due to coupling inertia, centripetal acceleration and coriolis acceleration can be neglected. The torque acting on a Tasrobot0 manipulator joint can be approximated as due to effective inertia and viscous friction of the joint only. For a horizontal joint system, the extra torque due to the weight of the links and the carried load can be included as a disturbance torque.

The Tasrobot0 is a robot manipulator system with five degrees of freedom. The roll and pitch rotations of the wrist joints are controlled by stepper motors. The rotations of the body-, shoulder- and elbow-joints are controlled by dc motors with constant field excitation. To control the latter three joints and provide servo operation, analogue controllers were designed. To facilitate the design, models

for the three joint systems and their unknown parameters were built and identified.

4.1 MODEL BUILDING

Although the models of respective joint systems have different parametric values, their structures are identical as shown in Figure (4.1-1).

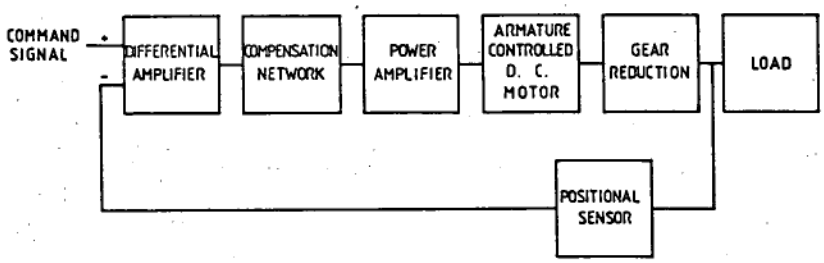


Figure (4.1-1): Block Diagram of A Joint System of The Tasrobot0 Manipulator

In the model building stage, the differential amplifier and the power amplifier in each joint system are grouped together and modelled as an amplifier with controllable gain A. Since the compensator used is a simple one-pole-one-zero compensator, the compensation network is modelled by a transfer function $G_{comp}(s)$ with pole at $s=-1/T_2$ and zero at $s=-1/T_1$:

$$G_{comp}(s) = \frac{sT_1+1}{sT_2+1} \qquad (4.1\text{-}1)$$

The dc servo motor, the reduction gear box, the load and the position sensor together are considered as a subsystem. The physical model of this dc motor subsystem is illustrated in Figure (4.1-2). The symbols listed below are used in Figure (4.1-2) and will be used in the derivation of the transfer function of the dc motor subsystem model:

v_a = voltage applied to the motor armature

i_a = armature current

R_a = motor armature resistance

L_a = motor armature inductance

N = gear reduction ratio of the gear train

J_m = effective inertia of the motor

B_m = coeficient of viscous friction of the motor

J_L = effective inertia of the load

B_L = coefficient of viscous friction of the load

$\Theta_m(t)$ = motor shaft angular displacement

$\Theta_L(t)$ = load shaft angular displacement

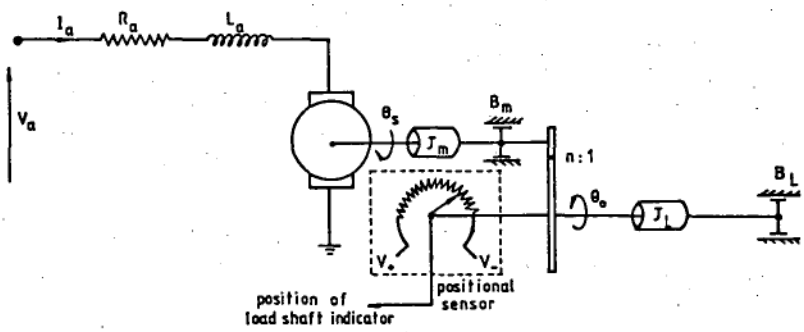


Figure (4.1-2): A Physical Model of A Joint System of The Tasrobot0 Manipulator

Considering the motor side of the dc motor subsystem, the electrical terminal equations of the constant-field armature-controlled dc servo may be obtained by equating the voltage drop around the armature loop as:

$$v_a(t) - K_v \Theta_m(t) = R_a i_a(t) + L_a \frac{d}{dt} i_a(t) \qquad (4.1\text{-}2)$$

where K_v is the back e.m.f. constant of the motor.

The torque, $T_m(t)$, generated by an armature controlled dc servo is proportional to the armature current, i_a, and is given by:

$$T_m(t) = K_T i_a(t) \qquad (4.1\text{-}3)$$

where K_T is the torque constant of the dc motor.

On the load side, the motor is geared down to drive the load with gear reduction ratio N. Assuming an ideal gear box, the torque on the load is N times the driving torque and the driving shaft velocity is N times the load velocity. If J is the total inertia and B is the resulting coefficient of viscous friction acting on the motor shaft, then:

$$J = J_m + J_L/N^2 \qquad (4.1\text{-}4a)$$

$$B = B_m + B_L/N^2 \qquad (4.1\text{-}4b)$$

$$\Theta_m(t) = N\Theta(t) \qquad (4.1\text{-}5)$$

The last terms of equations (4.1-4a) and (4.1-4b) represent respectively the reflected inertia and the reflected coefficient of viscous friction of the load on the motor shaft. If T_L is the resulting load torque acting on the motor shaft due to these inertia and viscous friction, then T_L can be expressed as:

$$T_L(t) = B\dot{\Theta}_m(t) + J\ddot{\Theta}_m(t) \qquad (4.1\text{-}6)$$

In some cases where the axis of rotation of the joint is not vertical, there is an additional torque acting on the joint due to the weight of its driven links. This additional torque, caused by the gravity loading effect, will be modelled as a disturbance torque and designated by T_g. If T is the total load torque acting on the motor shaft, then

$$T(t) = B\dot{\Theta}_m(t) + J\ddot{\Theta}_m(t) + T_g(t) \qquad (4.1\text{-}7)$$

This total load torque required is provided by the motor generated torque, $T_m(t)$. Therefore, equating equations (4.1-3) and (4.1-7) and rearranging, one gets:

$$K_T i_a(t) - T_g(t) = B\dot{\Theta}_m(t) + J\ddot{\Theta}_m(t) \qquad (4.1\text{-}8)$$

If V_θ is the voltage from the position sensor having a proportional constant K_b, then

$$V_\theta(t) = K_b \Theta(t) \qquad (4.1\text{-}9)$$

A model for the dc motor system can now be established according to equations (4.1-2) to (4.1-9) and is illustrated in Figure (4.1-3).

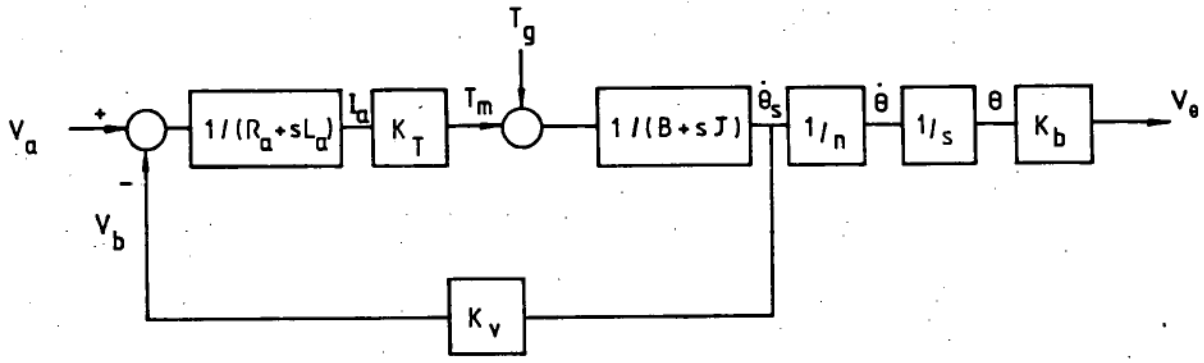


Figure (4.1-3): Block Diagram of The DC Motor Subsystem

In practice, there are non-linearities that cannot be ignored in the system. The most commonly encountered non-linearities are the saturation non-linearity due to saturation of amplifiers and the dead-space non-linearity due to static friction between sliding surfaces of the joint. The parameters for the saturation non-linearity model can be determined by direct measurements. But the parameters of the dead-space non-linearity model are usually time-varying as the non-linearity is caused by the coefficient of static friction which changes with joint position.

If V_1 and V_2 represent the positive and negative saturations of the amplifier used, and D_1 and D_2 represent the characteristics of the dead-space non-linearity, then the model structure of a joint system can be illustrated in Figure (4.1-4).

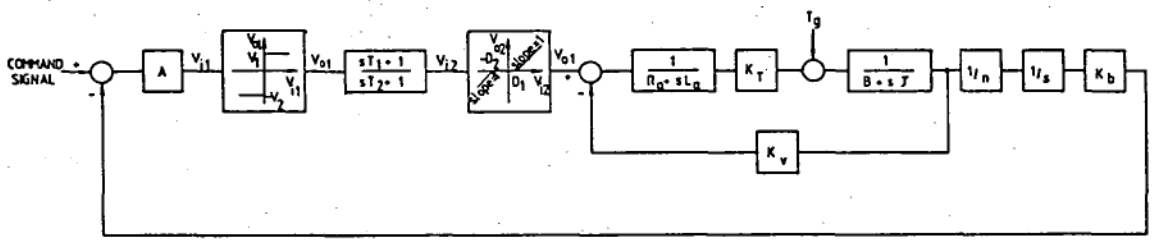


Figure (4.1-4): Block Diagram of A Model Structure of A Joint System

4.2 PARAMETERS TO BE IDENTIFIED

Since the dc motor subsystem contains most of the parameters of the model, the identification is mainly a construction of an optimum transfer function for the dc motor subsystem.

The postulated model shown in Figure (4.1-4) is closed-loop. There are two ways to identify the parameters of the subsystem. Either, the closed loop transfer function of the system is identified and then the model of the subsystem is derived from the identified parameters. Or, the open-loop transfer function of the system is identified directly by opening the feedback path. The latter method was used in order to eliminate the effect of amplifier saturation on the model parameters.

If the parameters of the dead space non-linearity model are small and can be linearized, the appromixated transfer function for the dc motor subsystem followed from Figure (4.1-4) can be written as:

$$G(s) = \frac{K}{s(s^2+as+b)} \qquad (4.2\text{-}1)$$

where $$K = \frac{AK_b NK_t}{JL_a}$$

$$a = \frac{R_a}{L_a} + \frac{B}{J}$$

$$b = \frac{R_a B + K_v K_t}{JL_a}$$

4.3 IDENTIFICATION TECHNIQUE

The model to be identified is a third order continuous-time transfer function denoted by G(s). The type of technique used for solving the parameter estimation problem is called generalized model-adjustment technique as illustrated in Figure (4.3-1).

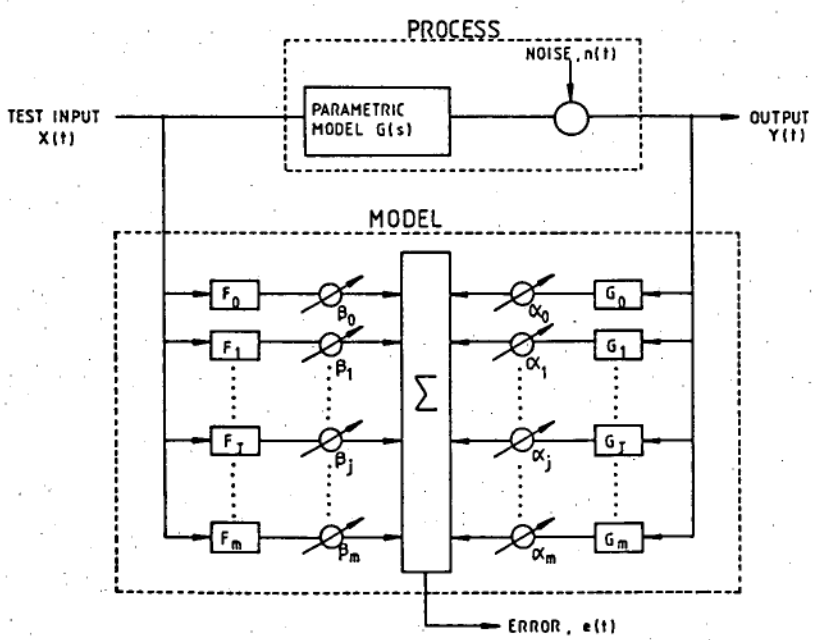


Figure (4.3-1): Block Diagram of The Generalized Model-Adjustment Technique

The Blocks $F_0, F_1, \ldots F_m$ and $G_0, G_1, \ldots G_n$ are operators such as linear transfer function, non-linear operators etc.. These blocks, together with the 'potentiometers' $\alpha_0, \alpha_2, \ldots \alpha_n$ and $\beta_0, \beta_1, \ldots \beta_m$ and the summer, form a

'generalized model' of the process. The noise, n(t), represents the measurement error of the output, y(t); and e(t) represents the equation error. An estimation scheme minimizing the sum squares of the equation error with respect to values $\alpha_0, \alpha_1, \ldots \alpha_n$ and $\beta_0, \beta_1, \ldots \beta_m$ was used. For implementation of the technique in a digital computer, a discrete model rather than an analog model was adopted. The parameters in the identified model are thus discrete-time.

The interface between the analog model and the computer is through a digital latch and a DAC which can be modelled as a zero-order hold data-extrapolator having a transfer function:

$$G_{zoh}(s) = \frac{1-e^{-sT}}{s} \qquad (4.3\text{-}1)$$

where T is the sampling interval

The resulting transfer function of the process including the zero-order hold data-extrapolator, $G^*(s)$, can be written as:

$$G^*(s) = G_{zoh}(s) \cdot G(s)$$

or,
$$G^*(s) = \frac{1-e^{-sT}}{s}\left[\frac{K}{s(s^2+as+b)}\right] \qquad (4.3\text{-}2)$$

Applying the Z-transformation, one obtains:

$$G^*(z) = \frac{B^*_0+B^*_1 z^{-1}+B^*_2 z^{-2}+B^*_3 z^{-3}}{1-A^*_1 z^{-1}-A^*_2 z^{-2}-A^*_3 z^{-3}} = \frac{Y(z)}{X(z)} \qquad (4.3\text{-}3)$$

where $G^*(z)$ represents an ideal discrete-time transfer function of the dc motor subsystem; Y(z) and X(z) are the Z-transforms of the corresponding sampled output signal, y(t) and input signal, x(t); and A^*_1, A^*_2, A^*_3, B^*_0, B^*_1, B^*_2 and

B^*_3 are the ideal discrete-time parameters of the process and can be found as:

$$B^*_0 = 0$$

$$B^*_1 = -Ka(1-\alpha)/b^2 + KT/b + D\varepsilon$$

$$B^*_2 = -Ka(\beta-1)/b^2 - 2\alpha KT/b - 2D\varepsilon$$

$$B^*_3 = -Ka(\alpha-\beta)/b^2 + \beta KT/b + D\varepsilon \qquad (4.3\text{-}4)$$

$$A^*_1 = (2\alpha+1)$$

$$A^*_2 = -(2\alpha+\beta)$$

$$A^*_3 = \beta$$

and,

$$\alpha = e^{-\sigma T}\cos wT$$

$$\beta = e^{-2\sigma T}$$

$$\varepsilon = e^{-\sigma T}\sin wT \qquad (4.3\text{-}5)$$

$$\sigma = a/2$$

$$w = \sqrt{b-\sigma^2}$$

$$D = K(a-b-\sigma)/b^2 w$$

This process model can be identified using discrete model-adjustment technique as shown in Figure (4.3-2).

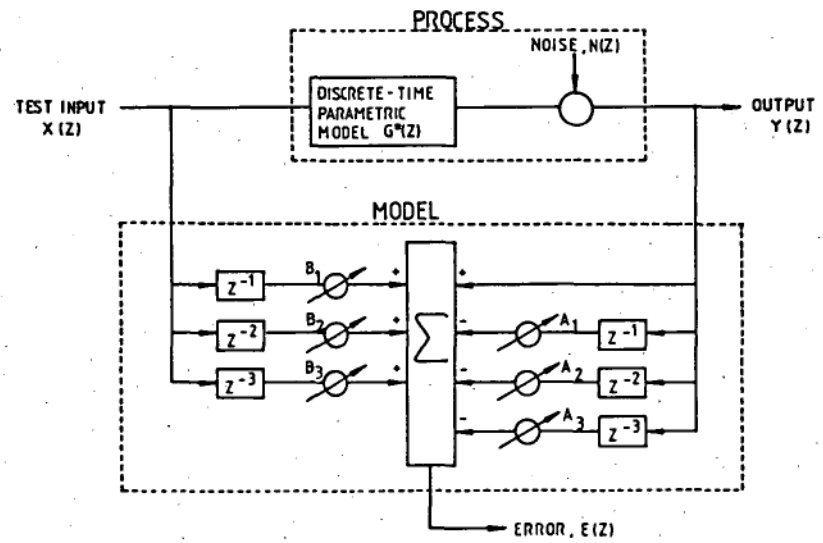


Figure (4.3-2): Block Diagram of The Discrete Model-Adjustment Technique For Identifying Joints of The Tasrobot0 Arm

From Figure (4.3-2), the discrete equation error E(z) can be expressed as:

$$E(z) = X(z)B_1 z^{-1} + X(z)B_2 z^{-2} + X(z)B_3 z^{-3} - Y(z) - Y(z)A_1 z^{-1} - Y(z)A_2 z^{-2} - Y(z)A_3 z^{-3} \qquad (4.3\text{-}6)$$

Denoting the k^{th} sampled value of the output as y_k, it follows from the Z-transform theory that:

$$Y(z)z^{-n} = y_{k-n} \qquad (4.3\text{-}7a)$$

Similarly,

$$X(z)z^{-n} = x_{k-n} \qquad (4.3\text{-}7b)$$

Therefore equation (4.3-6) can be expressed in matrix form as:

$$y_k = [x_{k-1} \;\; x_{k-2} \;\; x_{k-3} \;\; y_{k-1} \;\; y_{k-2} \;\; y_{k-3}] \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} + e_k \qquad (4.3\text{-}8)$$

For a large number of input and output measurements, k=1,2,.....n, equation (4.3-6) can be written as:

$$\begin{bmatrix} y_4 \\ y_5 \\ \vdots \\ \vdots \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_3 & x_2 & x_1 & y_3 & y_2 & y_1 \\ x_4 & x_3 & x_2 & y_4 & y_3 & y_2 \\ \vdots & & & \vdots & & \vdots \\ \vdots & & & \vdots & & \vdots \\ \vdots & & & \vdots & & \vdots \\ x_{n-1} & x_{n-2} & x_{n-3} & y_{n-1} & y_{n-2} & y_{n-3} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} + \begin{bmatrix} e_4 \\ e_5 \\ \vdots \\ \vdots \\ \vdots \\ e_n \end{bmatrix} \qquad (4.3\text{-}9)$$

Or, $$\underline{Y} = H \cdot \underline{\Theta} + \underline{E} \qquad (4.3\text{-}10)$$

where $\underline{Y}$ and $\underline{E}$ are the [nx1] output- and error-vectors; H is the [nx6] input-output matrix; and $\underline{\Theta}$ is the [6x1] unknown parameter-vector. Equation (4.3-10) can also be written as:

$$\underline{E} = \underline{Y} - H \cdot \underline{\Theta} \qquad (4.3\text{-}11)$$

If $\hat{\underline{\Theta}}_n$ is the best estimator such that the sum of squares of the error vector components is minimized, then the cost function J can be expressed as:

$$J = \tfrac{1}{2}(\underline{Y} - H \cdot \underline{\Theta})^T (\underline{Y} - H \cdot \underline{\Theta}) \qquad (4.3\text{-}12)$$

For a minimum J with respect to $\underline{\Theta}$, it requires that:

$$\frac{\partial J}{\partial \underline{\Theta}} = -H^T (\underline{Y} - H \cdot \underline{\Theta}) = 0 \quad (\text{at } \underline{\Theta} = \hat{\underline{\Theta}}_n) \qquad (4.3\text{-}13)$$

Solving equation (4.3-13) gives the least squares estimate of the parameter-vector, $\hat{\underline{\Theta}}_n$, as:

$$\hat{\underline{\Theta}}_n = P_n H^T Y \qquad (4.3\text{-}14)$$

where $P_n = (H^T H)^{-1}$.

If there is an additional set of measurement: x_{k+1} and y_{k+1}, the refined best estimator can be written as[R26]:

$$\hat{\underline{\Theta}}_{n+1} = \hat{\underline{\Theta}}_n + K_{n+1}(Y_{n+1} - H_{n+1}\hat{\underline{\Theta}}_n) \qquad (4.3\text{-}15)$$

where $K_{n+1} = P_n H_n{}^T (1 + H_{n+1} P_n H_{n-1}{}^T)^{-1}$ (4.3-16)

$$P_{n+1} = P_n - K_{n+1} H_{n+1} P_n \qquad (4.3\text{-}17)$$

$$H_{n+1} = [x_n \; x_{n-1} \; x_{n-2} \; y_n \; y_{n-1} \; y_{n-2}] \qquad (4.3\text{-}18)$$

System identification using recursive least squares method can be initiated with:

either (i) a data block of say, k=10, and computing $\hat{\Theta}_{10}$ and P_{10}

or (ii) an initial estimate for $\hat{\Theta}_0$ and a P matrix of large diagonal values.

4.4 TEST INPUT

The test signal used to energize a joint system for identification process is a 12-bit maximal length pseudo-random binary sequence (PRBS). It can be easily generated using a digital computer and is reproducible. Its bandwidth can be adjusted so that all frequency modes of the system under test can be excited.

The principle of generation of the test signal has been described elsewhere[R16, R18]. In the case of a 12-bit PRBS, the generation scheme is illustrated in Figure (4.4-1). The input to the shift register is from a clock generator with frequency f_c. The PRBS is obtained from the 12th stage

output of the shift register. The actual test signal is generated by converting the two logical levels of the PRBS into corresponding voltage levels using a comparator as shown in Figure (4.4-1).

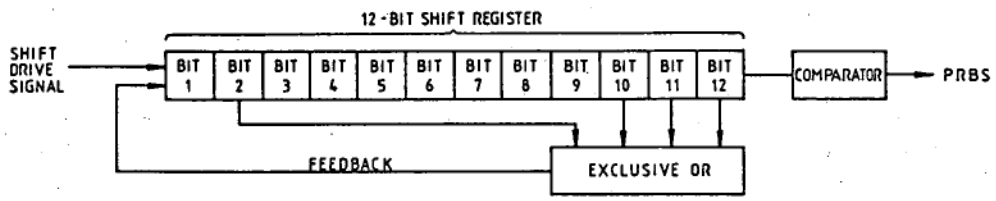


Figure (4.4-1): The Principle of Generating A 12-Bit Maximal Length PRBS Test Signal

A discussion of the nature of the test signal will be followed. For the sake of convenience, the two states of the binary sequence are assigned values +1 and -1 . A typical section of the PRBS test signal is shown in Figure (4.4-2).

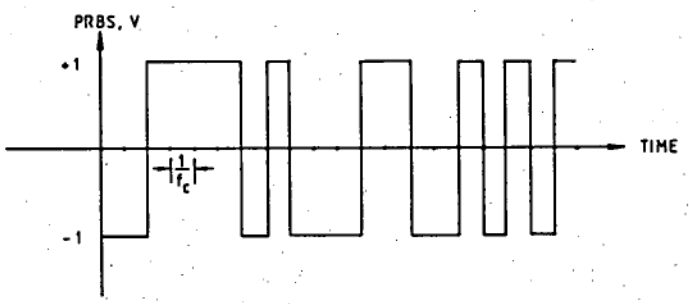


Figure (4.4-2): A Typical Section of A PRBS

The corresponding autocorrelation function, $\phi_x(\tau)$, of the PRBS signal is shown in Figure (4.4-3) and is given by[R15]:

$$\phi_x(\tau) = \frac{-1}{L} + \frac{L+1}{LT_c} \sum_{k=-\infty}^{\infty} g(\tau - kLT_c) \qquad (4.4\text{-}1)$$

where $L = 2^N - 1$.

N = number of bit in the shift register (=12)

$T_c = 1/f_c$ = period of clock driving the shift register

$$g(\tau) = p_r(\tau+T_c) - 2p_r(\tau) + p_r(\tau-T_c)$$

where $p_r(\tau)$ is a unit ramp function.

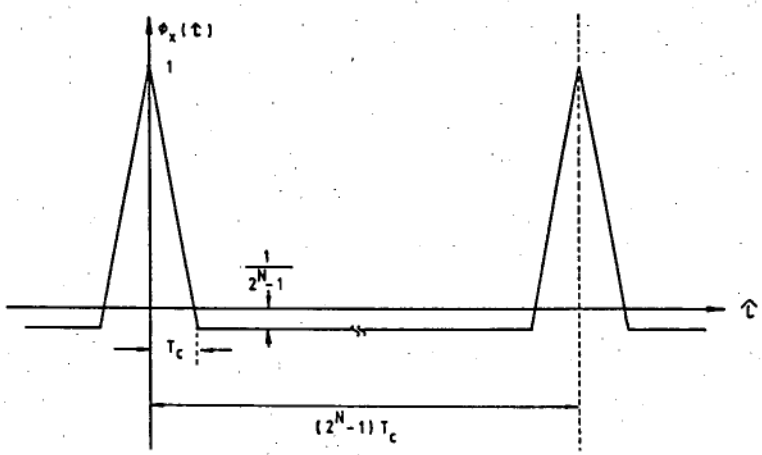


Figure (4.4-3): Auto-Correlation Function of A Maximal Length N-Bit PRBS

The function consists of an infinite series of triangular spikes centered at $\tau=kLT_c$ for $k=-\infty,..,-1,0,1,..\infty$.

The power-density spectrum, $\Phi_x(f)$, of the PRBS can be found from the Fourier Transform of the periodic auto-correlation function, $\phi_x(\tau)$, in equation (4.4-1). The transform relation for the discrete spectra, with fundamental frequency $1/LT_c$ Hz, is given by[R15]:

$$\Phi_x(f) = \frac{1}{L^2}U_0(f) + \sum_{\substack{r=-\infty \\ r\neq 0}}^{\infty} \frac{L+1}{L^2}\left[\frac{\sin(r\Pi/L)}{(r\Pi/L)}\right]^2 U_0\left(f-\frac{r}{LT_c}\right) \quad (4.4\text{-}2)$$

where $U_0(f-r/LT_c)$ is a unit impulse function centered at $f=r/LT_c$.

The shape of the power-density spectrum $\Phi_x(f)$ is a discrete sinc^2 function and is shown in Figure (4.4-4). It should be noted that because of the very small dc component in the PRBS signal, the power density spectrum at f=0 is very small and has value $1/L^2$. The bandwidth of the signal is equal to the driving clock frequency, $1/T_c$ or f_c. It can be varied by simply varying the driving clock frequency.

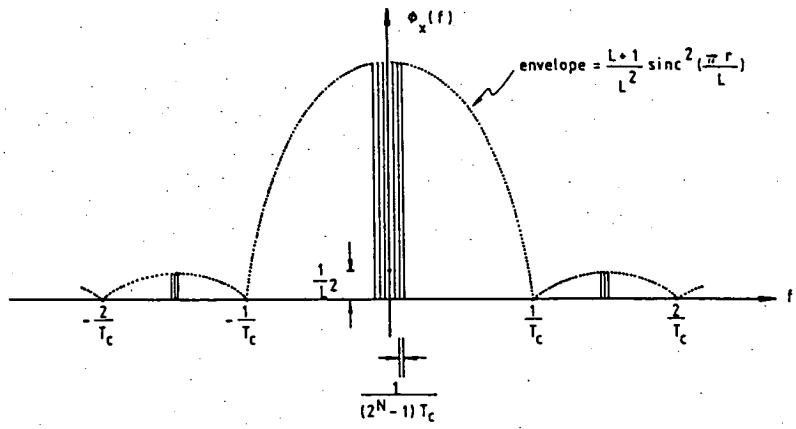


Figure (4.4-4): Power-Density Spectrum of A Maximal Length N-Bit PRBS

However, from equation (4.4-2), the number of line spectra within the bandwidth of the test signal is constant for a fixed number of bits of the PRBS and is equal to (2^N-1). If the bandwidth of the signal is increased, the fundamental frequency of the discrete spectra will also be increased.

4.5 IMPLEMENTATION OF THE IDENTIFICATION PROCESS

The identification process was implemented off-line by a series of programs written in Fortran-77 language. Details of the programs can be found in Appendix D. The programs can be arranged in a series of three main stages. The first stage is the generation of a 12-bit maximal length PRBS whose two states are represented by user-specified voltage levels. In the second stage, the generated PRBS is used as a test signal to excite a joint system and the resulting output is sampled and stored. The clock frequency generating the PRBS is user-specified. In the third stage, the PRBS data generated and the corresponding output data sampled are used to estimate the parameters of the discrete model using the technique discussed in section 4.3. The recursive least squares method is applied starting with an estimate for the

unknown parameter-vector and a P matrix with large diagonal values.

During the identification of a joint system, the equivalent voltage levels and the driving clock frequency of the PRBS must be carefully selected. When the voltage level is too small, the dead-space non-linearity cannot be overcome; where if it is too large, the amplifier becomes saturated. Also, the clocking frequency for the PRBS must be high enough to excite all modes of a joint system. But the higher the clocking frequency is, the larger the fundamental frequency of the PRBS discrete-spectra will be. When the signal passes through a low pass system, the power output decreases as the fundamental frequency of the signal discrete spectra increases. While most mechanical systems have low pass characteristics, the output power of a joint system excited by the PRBS input will be degraded as the clocking frequency of the PRBS increases. This reduces the output signal-to-noise ratio and thus lowers the accuracy of the identified parameters.

Owing to the integrating nature of a dc motor subsystem, an excited joint may move out of its limits during identification and cause damage to the gearings. The identification process is programmed to terminate itself whenever the joint limits are likly to be exceeded by the excited joint.

The shoulder joint and the elbow joint of the Tasrobot0 system have horizontal axes of rotation. To eliminate the gravitation effect on the parameters, the joint axis was aligned with the vertical in the respective identification

process. The gravitational effect will be discussed in next chapter.

The three joint systems are assumed to be mutually independent. Only one joint will be excited during each identification process. However, the effective inertia of a joint depends on the configuration of the arm. It is well-known in feedback control theory that the variation of open-loop parameters will have little effect on the closed-loop dynamic response of a system if the loop gain is sufficiently large. In Figure (4.1-4), provided adequate loop gain is allowed for, the two feedback loops will reduce the effect of the varying effective inertia of a joint caused by changes in arm configuration. It is, therefore, assumed that variation of joint effective inertia is negligible for any joint system of the Tasrobot0. This assumption will be checked in next chatper.

Under this assumption, the arm configuration becomes insignificant in the controller design for the three dc motor joints. In the identification process, the links are configured so that the moment of inertia of a joint system is set approximately to the average value of any likely values.

4.6 CONVERSION FROM DISCRETE-TIME MODEL TO CONTINUOUS-TIME MODEL

Since a discrete time model has been adopted, its transfer function must be translated to a corresponding continuous-time transfer function to enhance the design of an analog compensator. The approximated discrete-time transfer function is in the form:

$$G(z) = \frac{B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3}}{1 - A_1 z^{-1} - A_2 z^{-2} - A_3 z^{-3}} \qquad (4.6\text{-}1)$$

In order to convert this form into a corresponding continuous-time transfer function, equation (4.6-1) must first be converted from z-domain to s-domain by taking inverse Z-transformation. The resulting form in s-domain is then factorized into two continuous-time transfer functions - one for the zero-order-hold data extrapolator representing the DAC and the other for the dc motor subsystem.

For a third order system with transfer function shown in equation (4.6-1), there are four possible cases of conversion depending on the nature of the roots of the denominator polynomial. These four cases are analysed and shown in Appendix C.

4.7 RESULTS OF IDENTIFICATION ON THE THREE JOINT SYSTEMS

In applying the recursive least squares method, the initial estimates for the model parameters are zero and the diagonal elements of matrix P are set to 100.

For the body-joint system, the equivalent voltage level and the clock period of the PRBS input test signal were set to 0.5V and 0.05 seconds respectively. 4095 input/output measurements were obtained. From these data, the unknown parameters of the body-joint discrete-time model were deduced. The transfer function of the model, $G_B(z)$, can be expressed as:

$$G_B(z) = \frac{0.02507851z^2 + 0.03633119z + 0.02122747}{(z-0.9999464)(z-0.0972323)(z-0.3817543)} \qquad (4.7\text{-}1)$$

The variations of parameters for the last 500 estimates is shown in Figure (4.7-1). Little variations can be seen in the parameters in their estimation process. The final values of the diagonal elements of the P matrix are in the order of 10^{-9} to 10^{-5}. Hence, the parameter-vector converges. From Figure (4.7-1), the percentage variations of the numerator coefficients are larger than that of the denomiator coefficients in the discrete-time function. These variations account for the presence of zeros in the translated continuous-time transfer function and will be discussed in next chapter.

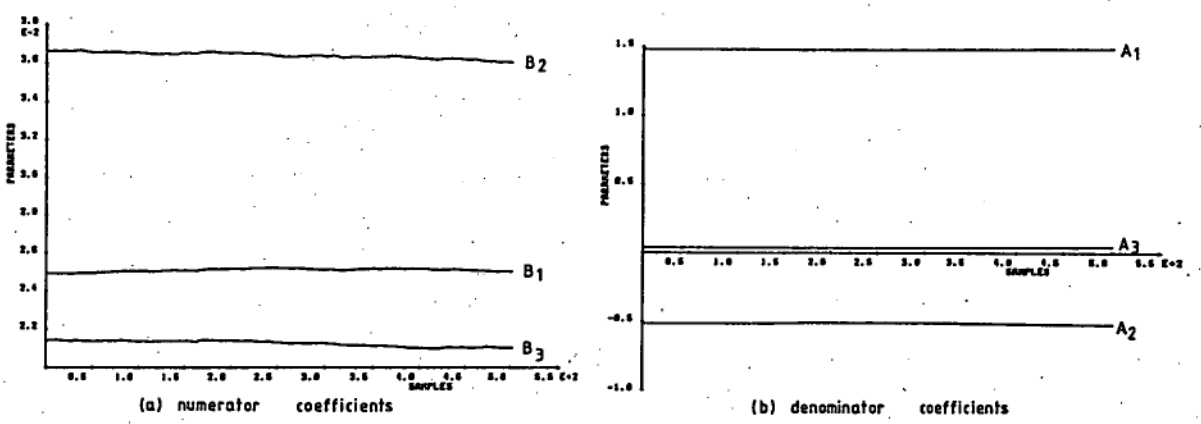


Figure (4.7-1): Variations of Parameters For The Last 500 Estimates of The Body-joint System

For the shoulder-joint system, the same PRBS test signal was used as input and 3133 input/output measurements were obtained. The transfer function, $G_S(z)$, can be expressed as:

$$G_S(z) = \frac{0.08505298z^2 + 0.15495700z + 0.08702245}{(z-0.9998622)(z-0.2744630)(z-0.1776742)} \qquad (4.7\text{-}2)$$

For the elbow-joint system, a slightly modified PRBS input test signal was used. The equivalent voltage level and the clock period of the PRBS test signal were set to 0.5V

and 0.02 seconds respectively. The higher frequency PRBS input enhances a larger collection of data within a shorter time before the elbow joint is terminated for tendancy to exceed its limits in the open loop identification process. 1426 input/output measurements were obtained and the transfer function, $G_E(z)$, can be expressed as:

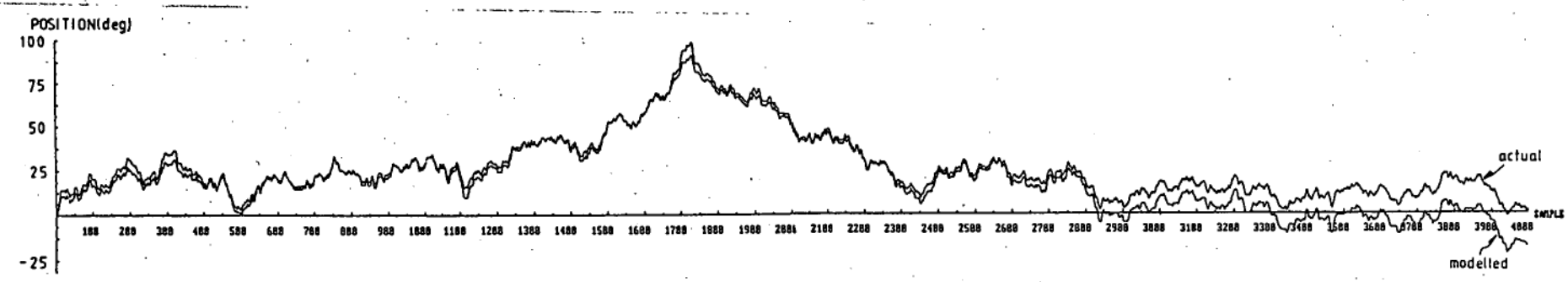
$$G_E(z) = \frac{0.12504890z^2+0.17420570z+0.05432743}{(z-0.9990688)(z-0.5215821)(z-0.0080810)} \qquad (4.7\text{-}3)$$

The variations of the parameters for the shoulder- and the elbow-joint systems are as minimal as that for the body-joint system. The final values of the diagonal elements of the P matrix for each case are also in the same order of magnitudes.

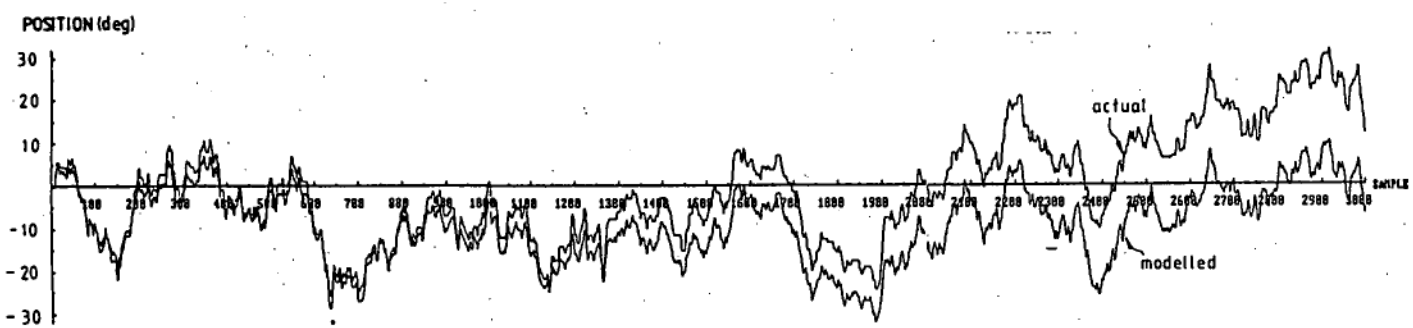
The accuracy of the discrete-time model is checked by comparing the actual output of the joint system with the model output as shown in Figures (4.7-2). From these figures, the variations of the model outputs are same as the actual outputs but there are significant drifts in the body- and shoulder-joint cases. Since the employed least squares method minimizes only the equation error, the output error has not been emphasised. The relationship between the equation error, $E_{eq}(z)$, and the output error, $E_{out}(z)$, can be shown as:

$$\frac{E_{out}(z)}{E_{eq}(z)} = \frac{1}{1-A_1 z^{-1}-A_2 z^{-2}-A_3 z^{-3}} \qquad (4.7\text{-}4)$$

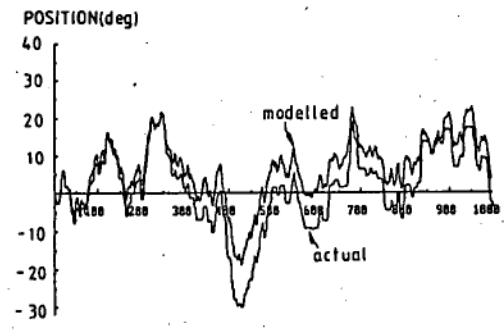
where $1-A_1 z^{-1}-A_2 z^{-2}-A_3 z^{-3}$ is an approximated denominator polynomial of an identified system. Owing to the existence of an integrating element in each identified system, its equation error is also integrated. If the equation error has



(a) BODY



(b) SHOULDER



(c) ELBOW

Figure (4.7-2): The Modelled Outputs And The Actual Outputs Variations of The Three Joint Systems

a non-zero mean, the output error will accummulate. This accounts for the drifts in the body- and the shoulder-joint systems.

A steepest-descent method using the least squares estimates as the starting point was also used to minimize the sum of the squares of the output error. However, the optimum parameter values obtained differ only by about 2% from the least squares estimates and there was no significant improvement in the degree of match.

Each of the discrete-time transfer functions contains a pole very close to z=1, which is as expected since each joint system contains an integrating element. Formula (C-7) and (C-9) shown in Appendix C were used and the corresponding continuous-time transfer functions for the dc motor subsystem can be found as:

$$G_B(s) = \frac{0.141(0.000423s^2+0.01055s+1)}{s(0.02145s+1)(0.05192s+1)} \qquad (4.7\text{-}5a)$$

$$G_S(s) = \frac{0.493(0.000394s^2-0.01240s+1)}{s(0.02893s+1)(0.03867s+1)} \qquad (4.7\text{-}5b)$$

$$G_E(s) = \frac{0.692(0.000304s^2-0.00772s+1)}{s(0.01038s+1)(0.07682s+1)} \qquad (4.7\text{-}5c)$$

where subscripts B, S and E are designated for the body-, shoulder- and elbow-joint systems respectively. These translated transfer functions will be analysed in next chapter.

CHAPTER FIVE

MODEL ANALYSIS AND COMPENSATOR DESIGN

5.1 ANALYSIS OF IDENTIFIED MODELS

After the identification of the discrete-time pulse transfer function, the transfer function is translated into its corresponding continuous-time form. Each dc motor subsystem obtained is a third order system having one pole at s=0 and generally two zeros. The form of the translated transfer function, $G_1(s)$, from the identification and the translation processes can be written as:

$$G_1(s) = \frac{k_1 (b_2 s^2 + b_1 s + 1)}{s(sT_3 + 1)(sT_4 + 1)} \qquad (5.1\text{-}1)$$

where k_1 represents the velocity constant; $b_2 s^2 + b_1 s + 1$ represents the numerator polynomial; and $-1/T_3$ and $-1/T_4$ represent the poles of the continuous-time transfer function of the dc motor subsystem.

The conversion process in Appendix C shows that the poles of the translated continuous-time transfer function depend on the poles of the identified discrete-time transfer function only. But the zeros of the translated continuous-time transfer function depend on both the zeros and poles of the identified discrete-time transfer function in a complex fashion.

For a system having no zeros, the translated values of b_2 and b_1 will only vanish when the following conditions are valid:

$$\text{(i)} \qquad F_1 = (a_3 F_2 + a_2 F_3)T \qquad (5.1\text{-}2a)$$

$$\text{(ii)} \qquad (a_2 + a_3)F_1 = a_2 a_3 (F_2 + F_3)T \qquad (5.1\text{-}2b)$$

where $F_1 = \dfrac{B_1 + B_2 + B_3}{(C_2 - 1)(C_3 - 1)}$

$$F_2 = \frac{B_1 C_2^2 + B_2 C_2 + B_3}{(C_2 - 1)^2 (C_2 - C_3)}$$

$$F_3 = \frac{B_1 C_3^2 + B_2 C_3 + B_3}{(C_3 - 1)^2 (C_3 - C_2)}$$

$$a_2 = -\ln[C_2]/T$$

$$a_3 = -\ln[C_3]/T$$

where B_1, B_2 and B_3 are the identified zeros; C_2 and C_3 are the identified poles; and T is the sampling period. Because of errors in model structure and the finite word length representation of numbers in digital computers, the coefficients b_2 and b_1 do not normally vanish. While in this case, the resulting translated coefficients b_2 and b_1 are small, and the original continuous-time transfer function has no zero, the non-zero values of b_2 and b_1 can be considered as parametric errors introduced in the identification and the translation processes and will be neglected.

5.2 EFFECTS OF SATURATION NON-LINEARITY ON STEP RESPONSE

From the previous assumptions and analysis, a joint system can be schematically represented by the block diagram shown in Figure (5.2-1).

The symbols used in Figure (5.2-1) are related by the following equations:

$$\varepsilon(t) = r(t) - c(t) \qquad (5.2\text{-}1)$$

$$u(t) = \begin{cases} A\varepsilon(t) & \text{for } -Q < A\varepsilon(t) < Q \quad (5.2\text{-}2a) \\ Q & \text{for } A\varepsilon(t) > Q \quad (5.2\text{-}2b) \\ -Q & \text{for } A\varepsilon(t) < -Q \quad (5.2\text{-}2c) \end{cases}$$

$$C(s) = U(s)G(s) \quad (5.2\text{-}3)$$

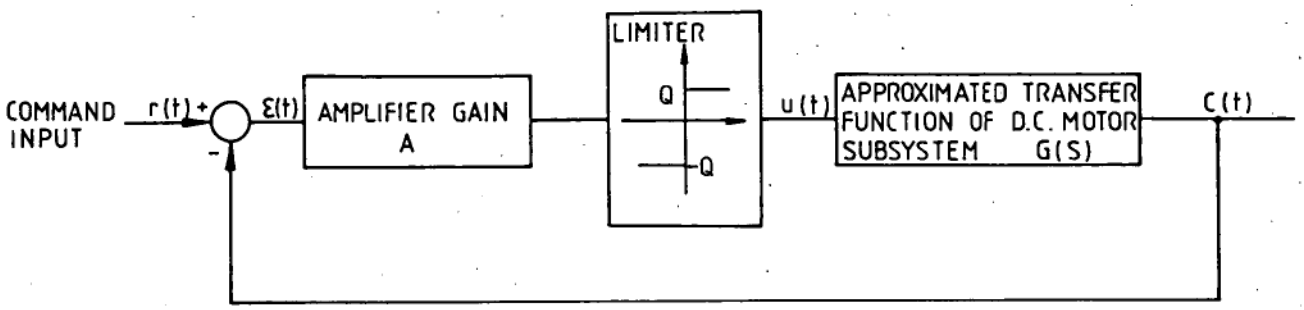


Figure (5.2-1): Block Diagram of A Joint System

Suppose at t=0 and c(t)=0, a step input r(t)=R is applied. The step response of a joint system can be divided into two stages. In the first stage, the amplifier is saturated and the system output is the response of the dc motor subsystem, G(s), subjected to an input step of Q (or -Q) volts. In the second stage, the amplifier becomes unsaturated and the system output is equivalent to the closed-loop response of the system subjected to the step input R. If t_1 is the time when the amplifier becomes unsaturated, then the step response of a joint system can be expressed as:

$$c(t) = L^{-1}\{G(s)R/s\} \qquad \text{for } 0 \le t \le t_1 \quad (5.2\text{-}4)$$

where L^{-1} is the inverse Laplace transform operator. For $t_1 \le t$, the output time response, c(t), can be obtained by solving the appropriate differential equations shown in section 4.1 with initial conditions equal to the end point conditions of equation (5.2-4).

In the analysis, the step response is assumed to be over-damped. For under-damped response, the two stages will be occurring alternatively. It is important to realize that

the end point boundary conditions of one stage becomes the initial point boundary conditions of the next stage.

If the gain of the amplifier is large, the transient response of a step input to a joint system will be dominated by the step response of the dc motor subsystem, G(s), which can be expressed in the form:

$$G(s) = \frac{C(s)}{U(s)} = \frac{k}{s(sT_a+1)(sT_b+1)} \qquad (5.2\text{-}5)$$

where T_a, T_b are poles of the transfer function.

When the amplifier is saturated, the input to the dc motor subsystem is a step of magnitude Q. The transient response of a joint system can be expressed as::

$$C(s) = \frac{kQ}{s^2(sT_a+1)(sT_b+1)} \qquad (5.2\text{-}6)$$

or,

$$c(t) = -kQ[T_a+T_b] + kQt + \frac{kQT_a^2}{T_a-T_b}e^{-t/T_a} + \frac{kQT_b^2}{T_a-T_b}e^{-t/T_b} \qquad (5.2\text{-}7)$$

If T_a and T_b are small compared to the transient period of the response, the transient output of a joint system, c(t), can be approximated as:

$$c(t) = kQt \qquad (5.2\text{-}8)$$

Q represents the saturation voltage of the amplifier and is unaltered once the circuit is built, while k is the velocity constant of the dc motor subsystem and is an assumed constant. Hence, the transient response of a joint system can be approximated by a ramp function with slope kQ.

From the analysis of the process of translating transfer function from discrete-time to continuous-time, it has been shown that the translated velocity constant of a dc motor subsystem is subjected to relatively large errors

compared with the translated poles. Futhermore, it has also been shown[R4] that the velocity constant is affected by the dead-space and the backlash non-linearities of the joint systems. Equation (5.2-8) offers another approach to obtain the velocity constant of a dc motor subsystem by measuring the transient behaviour of an actual step response of a joint system. Thus, the accuracy of the translated velocity constant can be investigated.

The actual step responses of the three joint systems are shown in Figures (5.2-2) and the transient response characteristics are listed in Table (5.2-1). By measuring

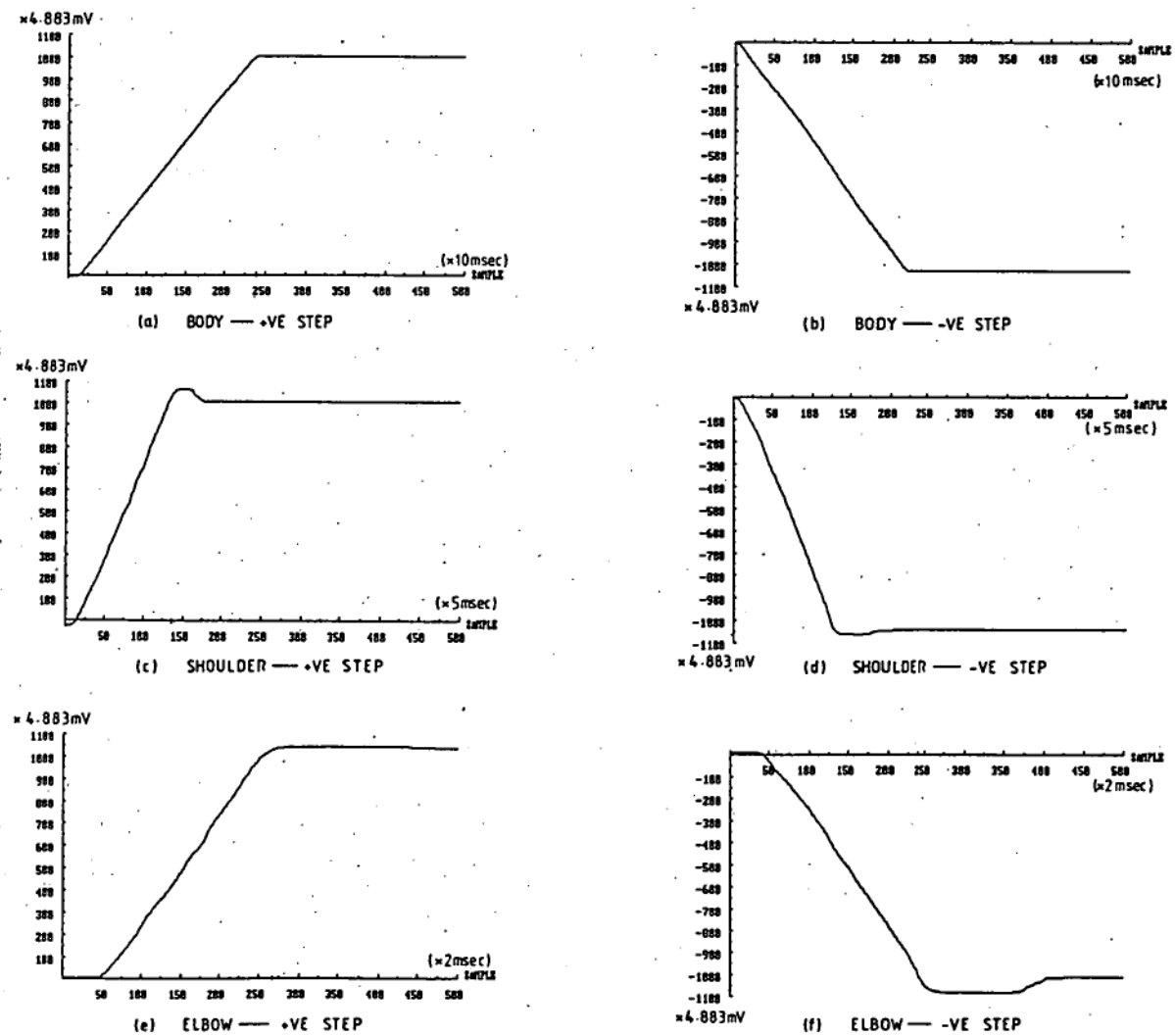


Figure (5.2-2): Actual Step Responses of The Three Joint Systems

JOINT SYSTEM	APPLIED STEP/V	LINEAR PORTION SLOPE V/SEC	PERCENTAGE OVERSHOOT	FIGURE SHOWN
BODY	5	2·2	~ 0 %	5·2-3a
	- 5	- 2·3	~ 0 %	5·2-3b
SHOULDER	5	8·0	6·2%	5·2-3c
	- 5	- 8·3	2·4%	5·2-3d
ELBOW	5	11·0	0 %	5·2-3e
	- 5	- 12·5	8·2%	5·2-3f

Table (5.2-1): Step Response Characteristics of The Joint Systems Deduced From Figure (5.2-3)

the slopes of the transient responses and the saturation voltages of the amplifiers, the velocity constants of the three joint systems can be found from equation (5.2-8) and are shown in Table (5.2-2).

JOINT SYSTEM	MODELLED VELOCITY CONSTANT , K	MEASURED AMPLIFIER SATURATION VOLTAGE		ESTIMATED VELOCITY CONSTANT FROM ACTUAL STEP RESPONSE		
		+VE	- VE	from +ve step	from -ve step	AVERAGE
BODY	0·149	12·00V	- 12·25V	0·183	0·187	0·185
SHOULDER	0·493	12·30V	- 12·50V	0·650	0·665	0·658
ELBOW	0·692	11·75V	- 12·80V	0·936	0·976	0·956

Table (5.2-2): Predicted velocity constants of The Three Joint Systems

Since each joint system is controlled by discrete signal commands from the host computer, the step response characteristics of each joint system is of the utmost important. In order that the transfer functions of each joint system can represent the step response of the acutal system more accurately, the velocity constants are replaced by those deduced from the actual step responses using equation (5.2-8). The modified transfer functions for the dc motor subsystems of the three joints are:

$$G_B(s) = \frac{0.185}{s(0.02145s+1)(0.05192s+1)} \qquad (5.2\text{-}9)$$

$$G_S(s) = \frac{0.658}{s(0.02893s+1)(0.03867s+1)} \qquad (5.2\text{-}10)$$

$$G_E(s) = \frac{0.956}{s(0.01038s+1)(0.07682s+1)} \qquad (5.2\text{-}11)$$

Using these modified transfer functions ,the step response of each joint system can be simulated. A technique very similar to analog computer technique was used in simulating a joint system in digital computer. This technique is provided by a commercially available simulation package - TUTSIM - from Meerman Automation[R31]. The functional blocks used are shown in Figure (5.2-3).

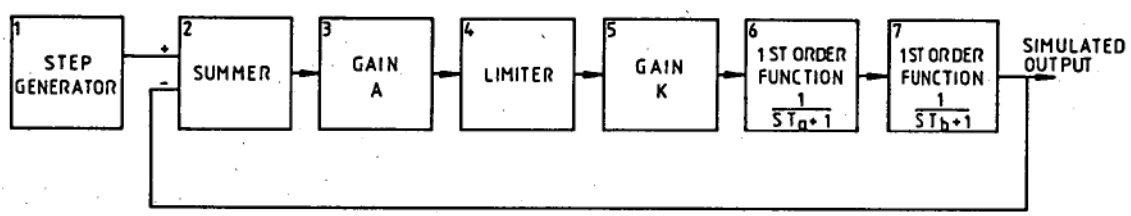


Figure (5.2-3): Functional Blocks of A Simulated Joint System

The results of the simulated step response for each joint system are shown in Figure (5.2-4a) to (5.2-4f). The deduced step response characteristics are listed in Table (5.2-3).

By comparing the characteristics of the simulated step responses with those of the acutal responses, the modified transfer function can be used to predict the behaviour of the actual systems. The three modified transfer functions shown in equations (5.2-9) to (5.2-11) are used to approximate the actual systems and to help design suitable compensators for the three joint systems.

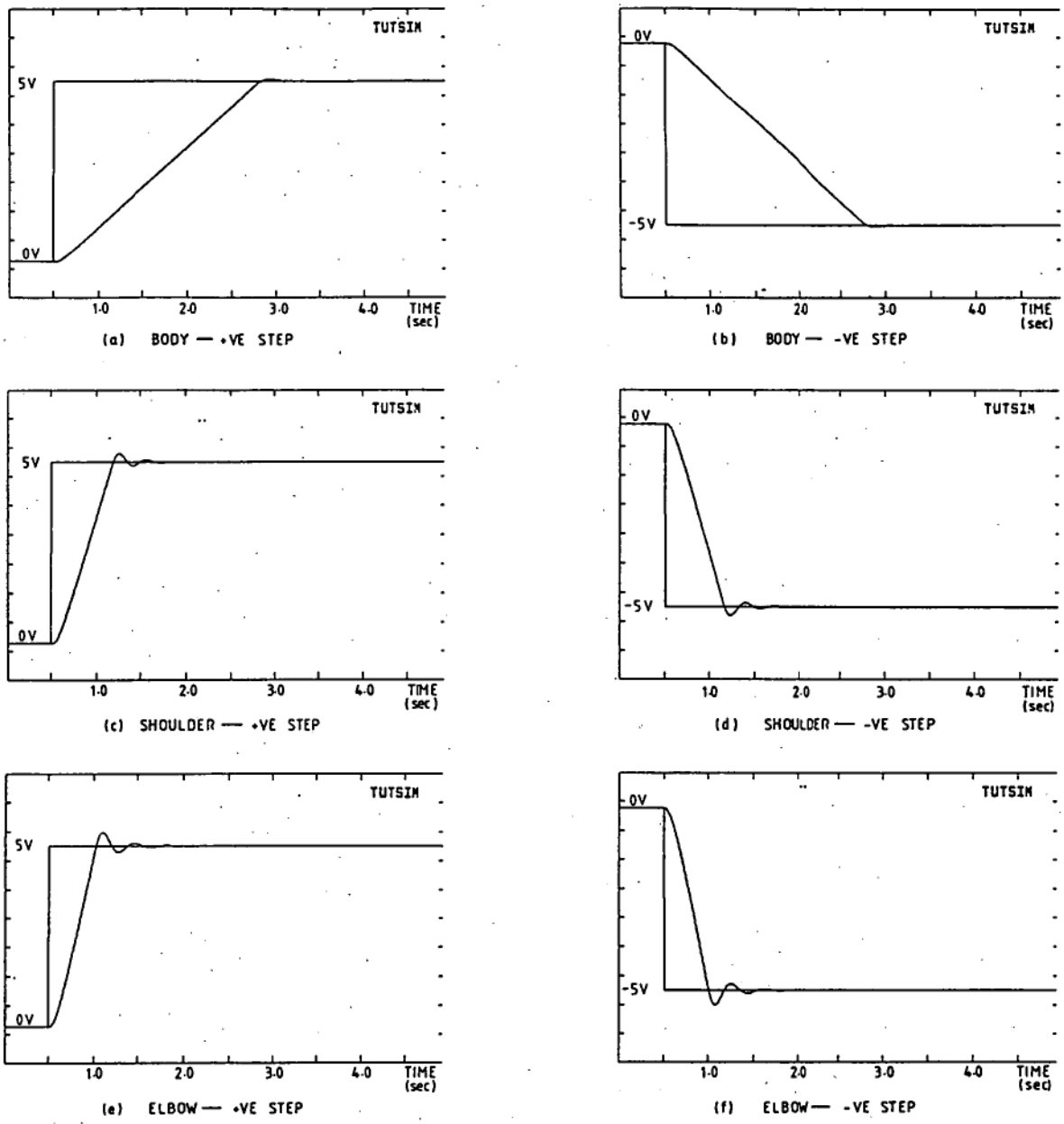


Figure (5.2-4): Simulated Step Responses of The Three Joint Systems

JOINT SYSTEM	LINEAR REGION SLOPES V/SEC		PERCENTAGE OVERSHOOT %	
	+VE STEP	-VE STEP	+VE STEP	-VE STEP
BODY	2·12	- 2·17	1·0 %	1·0 %
SHOULDER	7·60	- 8·50	4·8 %	4·8 %
ELBOW	11·09	-11·09	7·2 %	7·6 %

Table (5.2-3): Step Response Characteristics of The Simulated Joint Systems

5.3 EFFECTS OF DISTURBANCE TORQUE

During the identification process and the step response analysis, gravitational disturbance had been eliminated by aligning the joint axis with the vertical. However, in practice, rotating axes for shoulder and elbow joints are horizontal. The weight of the link and its neighbouring links, plus the load carried, impose a torque onto the joint.

This torque is time-varying in nature. It varies with the configuration of the arm as well as the load carried. The effect of this disturbance torque on a joint system can be seen in Figure (4.1-4). It can also be represented by an equivalent voltage, V_d, applied at the input of the motor subsystem as shown in Figure (5.3-1).

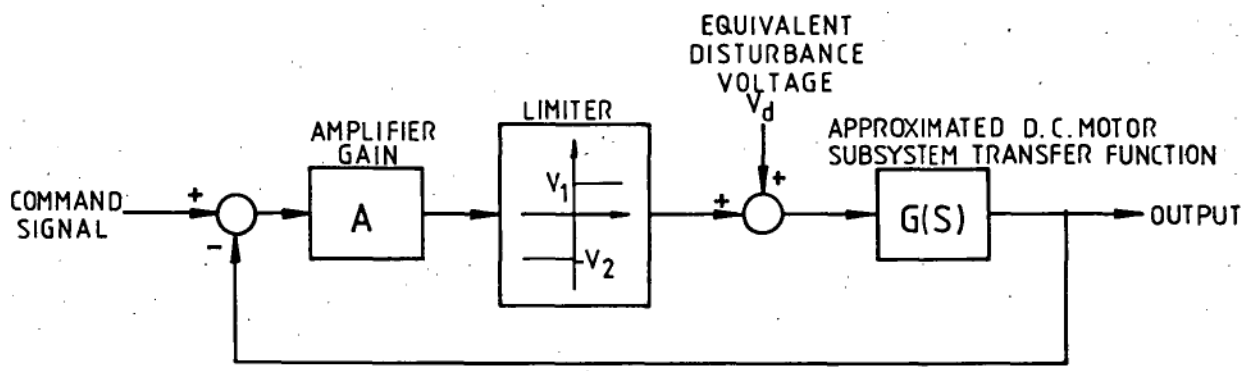


Figure (5.3-1): Block Diagram of A Joint System With Disturbances

From Figure (5.3-1), it can be seen that the disturbance will increase the steady state error in a step response since there must exist an error voltage to compensate for the disturbance voltage, V_d. The disturbance also affects the shape of the transient response of a step input as it alters the actual voltage input to the dc motor transfer function, G(s). The latter effect enables the value

of V_d to be estimated by comparing the transient responses of a joint system with and without the disturbance effects.

The responses without the disturbance had been shown in Figures (5.2-2a) to (5.2-2f) and in Table (5.2-1). The responses with disturbance effects were found by setting the shoulder and elbow to operate in normal positions, i.e. with horizontal axes of rotation. The step response of the joints are shown in Figures (5.3-2a) to (5.3-2d). It can be seen

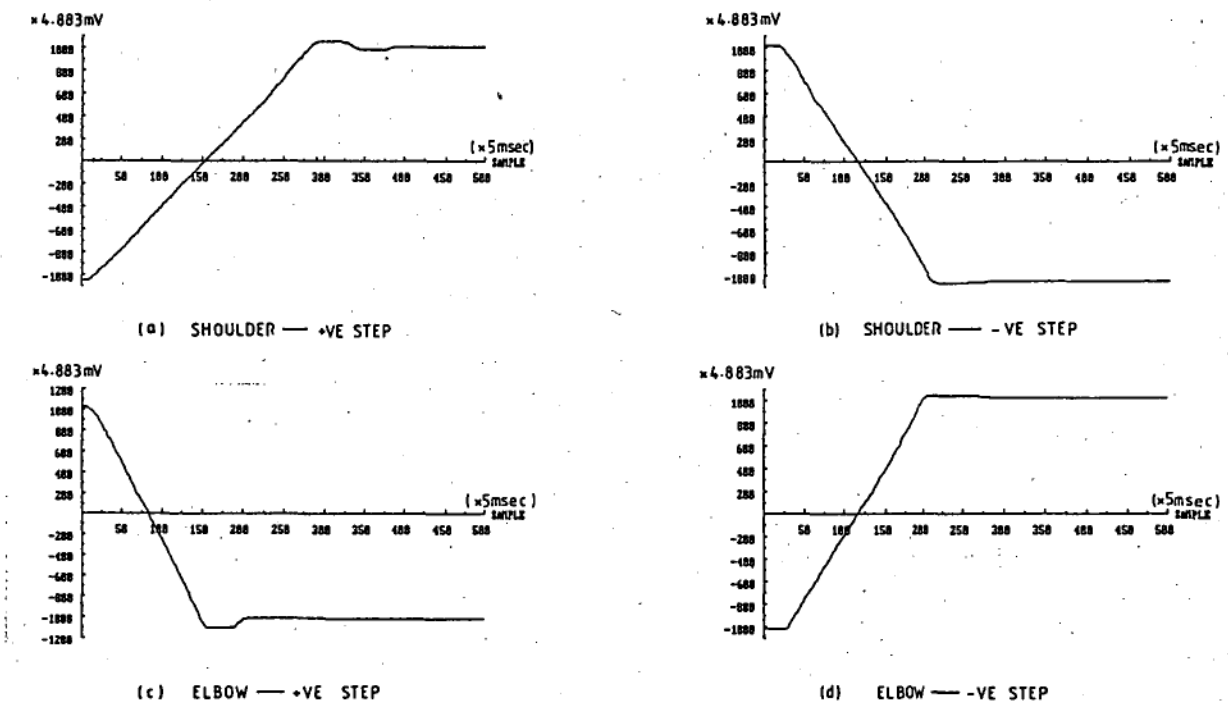


Figure (5.3-2): Actual Step Responses of The Shoulder And The Elbow Joint Systems In Normal Positions

from Figures (5.3-2) that the transient slopes are also constant in the responses with disturbance effects. This suggests that the time-varying effect of the disturbance torque caused by changes of arm configuration is small and can be neglected. V_d can, therefore, be approximated by a constant. By measuring the transient slopes and by using equation (5.2-8), the effective values of the input step applied to the dc motor subsystem can be evaluated. V_d for

each joint can be calculated from the difference between the effective step input and the saturation voltage of the amplifier. The results are listed in Table (5.3-1).

JOINT SYSTEM	STEP APPLIED/V	SLOPE DEDUCED V/SEC	ESTIMATED V_d V	FIGURE SHOWN
SHOULDER	+5	6.96	-1.73	5.3-2a
	-5	-10.19	-2.98	5.3-2b
ELBOW	+5	11.11	-0.34	5.3-2c
	-5	-14.30	-2.26	5.3-2d

Table (5.3-1): Results of Step Responses Under Disturbances For The Shoulder And The Elbow Joints

Simulation results with disturbance effects are shown in Figure (5.3-3a) to (5.3-3d). The responses agree to the actual responses, and the slopes predicted are accurate to within 8%.

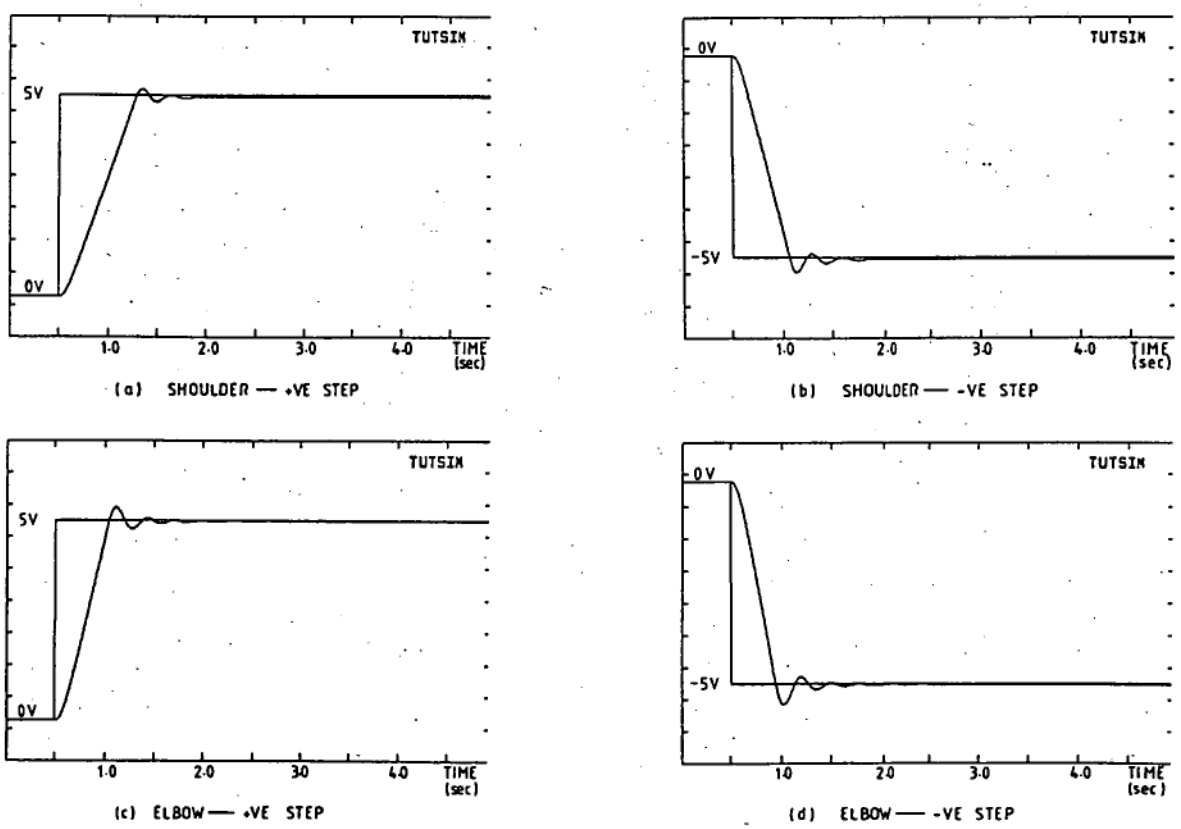


Figure (5.3-3): Simulated Step Responses of The Shoulder And Elbow Joint Systems With Disturbances

The steady state error, ε_1 (in degree), due to the effects of the disturbance torque (or the equivalent disturbance voltage, V_d, is illustrated in Figure (5.3-4) and can be expressed as:-

$$\varepsilon_1 = \frac{V_d}{AK_b} \qquad (5.3\text{-}1)$$

where K_b is the position transducer constant in v/deg. Hence, the steady state error, ε_1, can be reduced by increasing the amplifier gain A.

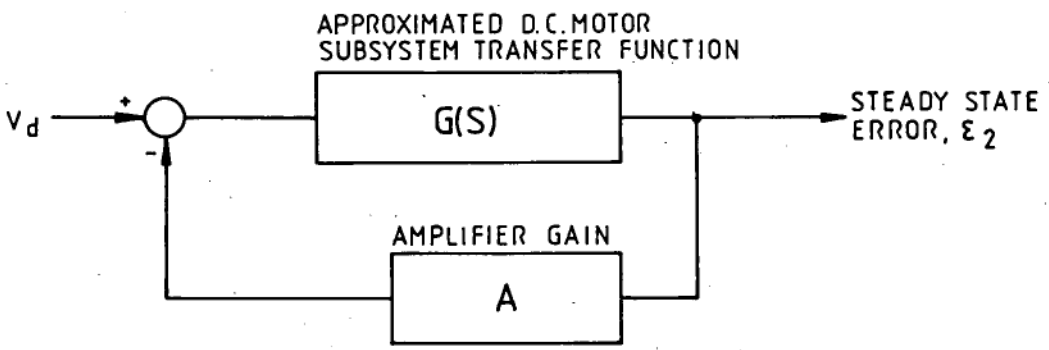


Figure (5.3-4): Block Diagram of A Joint System Due Only To The Effects of Disturbance At Steady State

However, the actual steady state error depends on the sophistication of the mechanical structure of a joint, such as the amount of backlash in the gearings and the rigidity of the arm structure. As with the Tasrobot0 arm which mechanical structure has not been emphasized, the error tolerance is selected to be about 1 degree. To reduce the effect of the disturbance torque on the steady state error to achieve this error tolerance, the minimum amplifier gains required for the shoulder and the elbow joints can be calculated from equation (5.3-1) as 13 and 10 respectively.

5.4 EFFECTS OF DEAD-SPACE NON-LINEARITY

The dead-space non-linearity is caused by static friction in a joint. The parameters for this non-linearity model can be estimated by direct measurement.

As static friction in a joint is not constant, a symmetrical dead-space characteristics is assumed and the maximum magnitude of the dead-space voltage is measured and adopted as the model parameter. If D is the maximum magnitude of dead-space voltage measured, the steady state error, ε_2 (in degree), due to the dead-space voltage D can be found as:

$$\varepsilon_2 = \frac{D}{AK_b} \qquad (5.4\text{-}1)$$

Equation (5.4-1) is similar to equation (5.3-1) and the error due to dead-space non-linearity can be reduced by increasing the amplifier gain A. For an error tolerance of 1 degree, the minimum amplifier gain required to reduce dead-space non-linearity effect on the steady state error can be found using equation (5.4-1) and is shown in Table (5.4-1).

JOINT SYSTEM	D, MEASURED DEAD SPACE VOLTAGE (V)	TOTAL FEEDBACK VOLTAGE (V)	TOTAL RANGE OF MOTION (°)	FEEDBACK CONSTANT (V/°)	MINIMUM AMPLIFIER GAIN REQUIRED
BODY	4.6	20	270	0.07274	62
SHOULDER	8.0	20	90	0.22222	18
ELBOW	8.0	20	90	0.22222	18

Table (5.4-1): The Measured Dead-Space Voltages, Feedback Constants And Minimum Gains of The Three Joint Systems For An Error Tolerence of One Degree

5.5 DESIGN OF CONTROLLERS

Because of limited accuracy of measurements, variations of disturbance torque and variations of system parameters, the steady state error tolerances for the joints may not be achieved if the minimum required amplifier gains are used. Also, the total maximum steady state error is the sum of the error due to the disturbance torque and the error due to the dead-space non-linearity. To ensure the steady state error to be within the error tolerance, amplifier gain larger than the minimum required will be used. Table (5.5-1) shows the amplifier gain used for each joint and their resulting maximum steady state errors.

JOINT SYSTEM	AMPLIFIER GAIN USED	ERROR DUE TO DEAD SPACE, ε_2	ERROR DUE TO DISTURBANCE, ε_1	TOTAL MAXIMUM ERROR $\varepsilon_1 + \varepsilon_2$
BODY	124	0·50 °	0 °	0·50 °
SHOULDER	70	0·51 °	0·19°	0·70 °
ELBOW	54	0·67 °	0·19°	0·86 °

Table (5.5-1): Steady State Errors Due To Dead-Space Non-Linearity And Disturbances For The Three Joint Systems

After selecting the amplifier gains to ensure that the open-loop gains for the systems are adequate for accuracy, it will often be found that the system transient performance is not satisfactory without modification. In order for the system to meet the requirements of stability and accuracy, certain types of compensator must be added to the system.

In the compensator design, since each joint system recieves discrete signal commands from the host computer, the step response characteristics of the joint system will mainly be concerned. The compensator required for each joint system will base on stability and step response of the

system. Moreover, the transient behaviour of the step response of each joint system will also be concerned to give short and smooth transient and low percentage overshoot.

Because of the presence of non-linearities in the joint systems, classical linear feedback control theories cannot be used directly. However, using those classical linear theories as guidelines and basing on the simulated step response of each compensated system, suitable compensator can be designed.

The bode plot of each joint system is shown in Figure (5.5-1). From these plots, the phase margin and the gain margain for each joint system can be estimated and are shown in Table (5.5-2). The bode plot of the shoulder joint shows

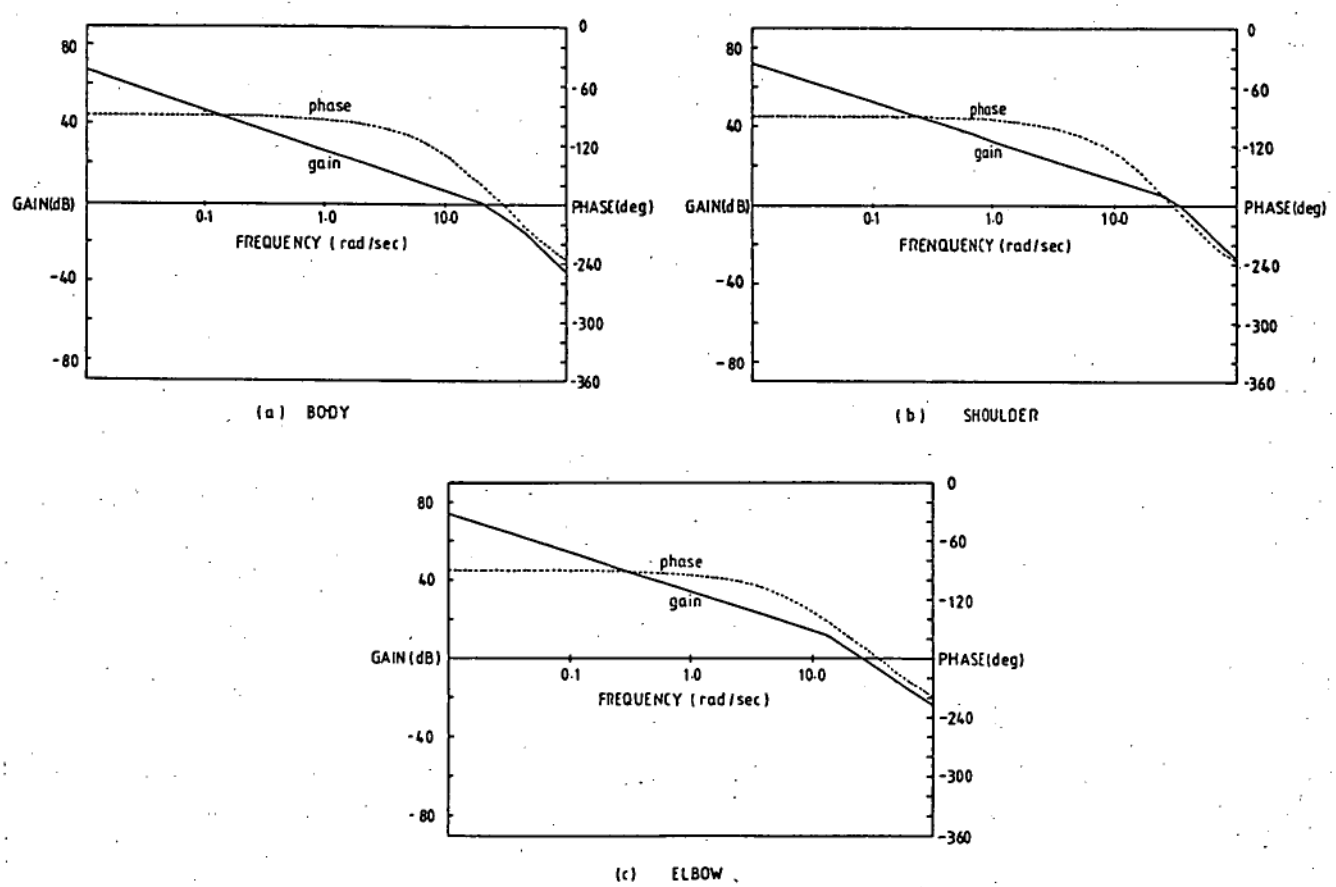


Figure (5.5-1): Bode Plots of The Uncompensated Joint Systems

JOINT SYSTEM	PHASE MARGIN	GAIN MARGIN	STABILITY SHOWN IN BODE PLOT	FIGURE SHOWN
BODY	19 °	6 dB	STABLE	5.5 - 1 a
SHOULDER	/	/	UNSTABLE	5.5 - 1 b
ELBOW	14 °	5 dB	STABLE	5.5 - 1 c

Table (5.5-2): Phase And Gain Margins of The Three Uncompensated Joint Systems

that the shoulder joint is unstable. Although the body and elbow systems appear to be stable from the plots, their low phase- and gain-margins indicate relatively low degree of stability.

Using the simulation technique discussed before, simulated step responses of the three joint system without compensators are shown in Figures (5.5-2) to (5.5-4). From these figures, the step response characteristics of each joint system can be estimated and are shown in Table (5.5-3). The simulated elbow joint system is unstable because the velocity and acceleration, as shown in Figure (5.5-4b), are in constant amplitude oscillations. The simulated shoulder joint system is stable. Although these might be caused by the effect of saturation non-linearity and truncation errors during the simulation process, these figures reflect critical stability of the two systems.

All joint systems have time-varying moment of inertias. To reduce the effects of these open-loop time-varying parameters on the dynamic response of the closed-loop systems, phase-lag series compensators were designed to allow sufficient gain- and phase- margins. The time-varying effects of the parameters on the responses of the compensated systems will be investigated in next section.

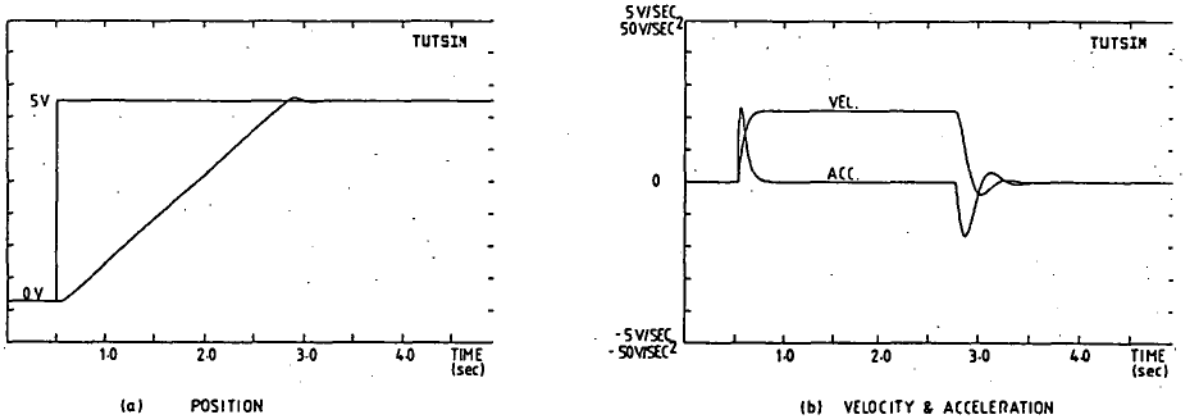


Figure (5.5-2): Simulated Step Response of The Uncompensated Body Joint System

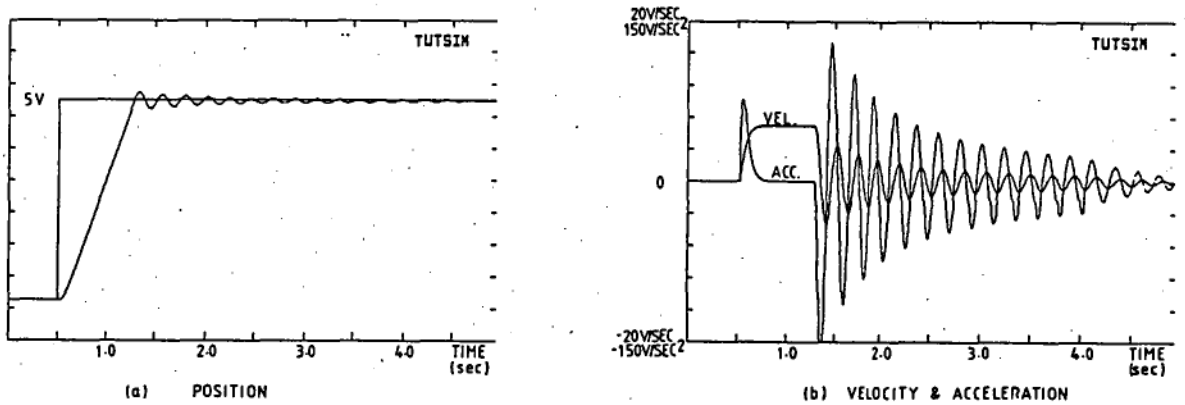


Figure (5.5-3): Simulated Step Response of The Uncompensated Shoulder Joint System

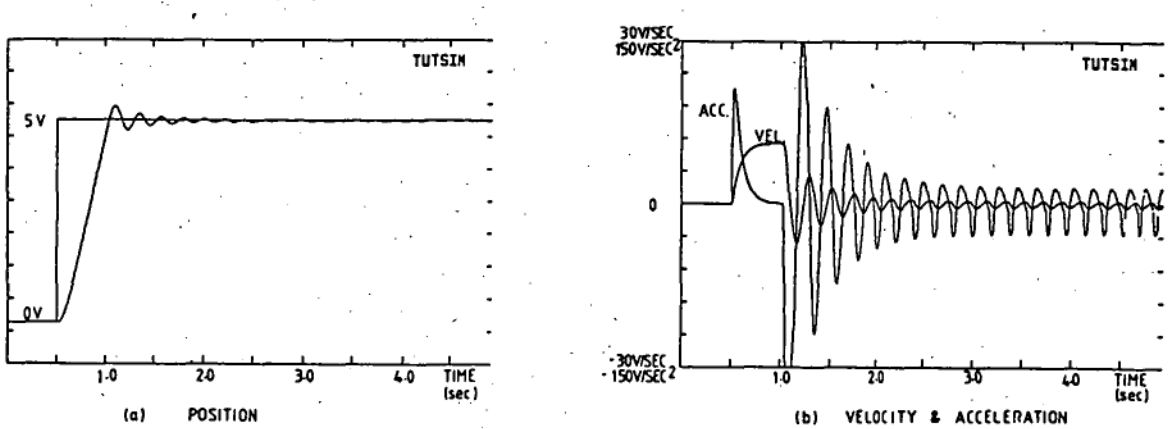


Figure (5.5-4): Simulated Step Response of The Uncompensated Elbow Joint System

JOINT SYSTEM	STEP INPUT APPLIED V	MAXIMUM ACCELERAT'N °/SEC^2	MAXIMUM RETARDAT'N °/SEC^2	MAXIMUM VELOCITY °/SEC	PERCENTAGE OVERSHOOT %	RISE-TIME SEC	TRANSIENT OSCILLATION TIME SEC	FIGURE SHOWN
BODY	+5	311	311	30	1·1	2·4	~0·5	5·5-2a-b
SHOULDER	+5	573	>675	32	3·8	0·8	>4·2	5·5-3a-b
ELBOW	+5	>675	>675	49	5·8	0·5	>4·5	5·5-4a-b

Table (5.5-3): Step Response Characteristics From The Simulation of The Three Uncompensated Joint Systems

The bode plots of the compensated joint systems are shown in Figures (5.5-5a) to (5.5-5c). The compensator pole

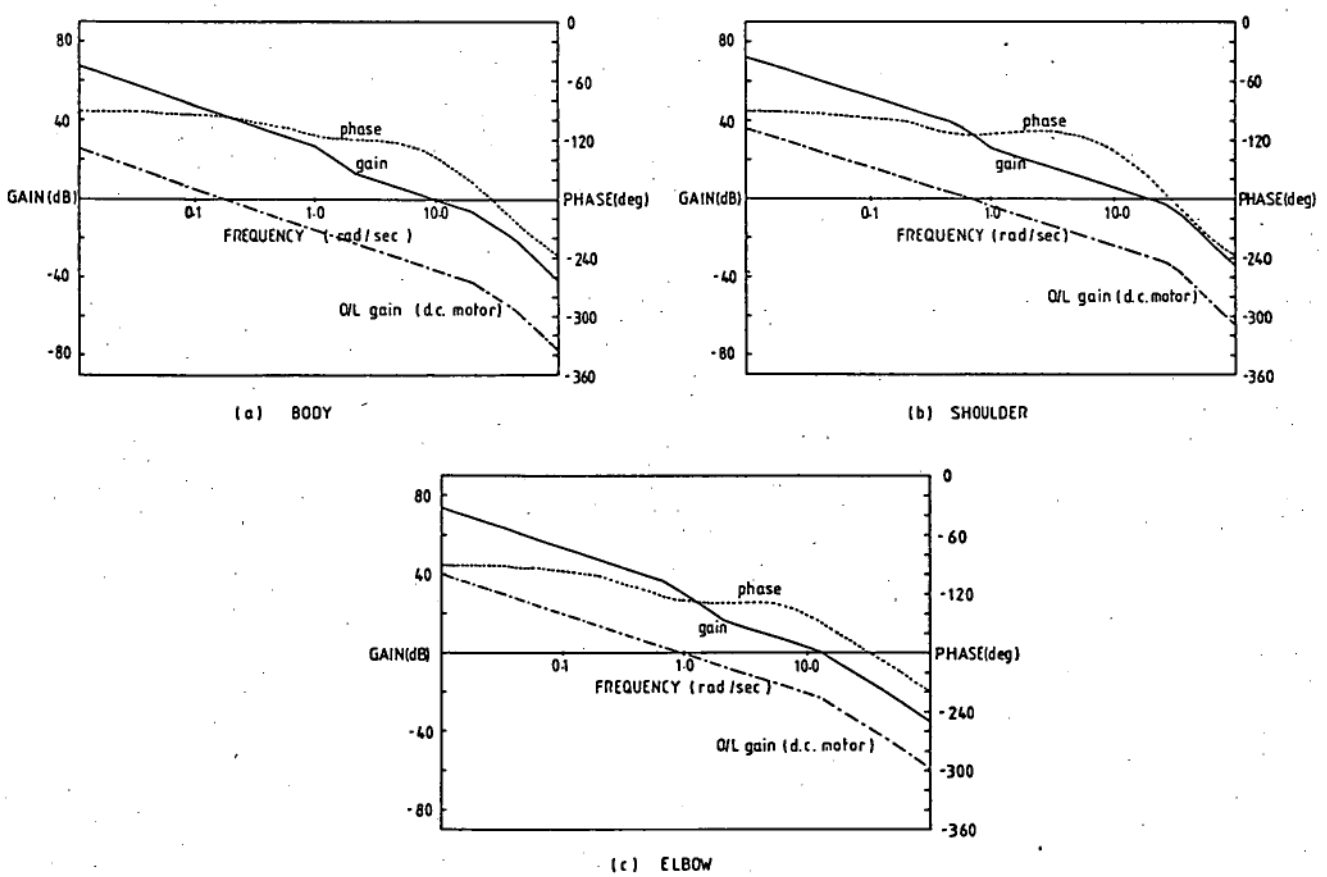


Figure (5.5-5): Bode Plots of The Compensated Joint Systems

and zero designed for each joint system and the resulting phase- and gain-margins are shown in Table (5.5-4). The simulated step responses of the compensated joint systems are shown in Figures (5.5-6) to (5.5-8). The step response characteristics deduced are shown in Table (5.5-5).

JOINT SYSTEM	COMPENSATON TRANSFER FUNCTION , G (S)	PHASE MARGIN	GAIN MARGIN	STABILITY	FIGURE SHOWN
BODY	(0.47 S + 1) / (S + 1)	44 °	12 d B	STABLE	5.5 - 4 a
SHOULDER	(S + 1) /(2.2 S + 1)	22 °	5 d B	STABLE	5.5 - 4 b
ELBOW	(0.47 S + 1) /(1.5 S + 1)	30 °	15 d B	STABLE	5.5 - 4 c

Table (5.5-4): Compensators Designed And The Corresponding Phase And Gain Margins of The Three Joint Systems

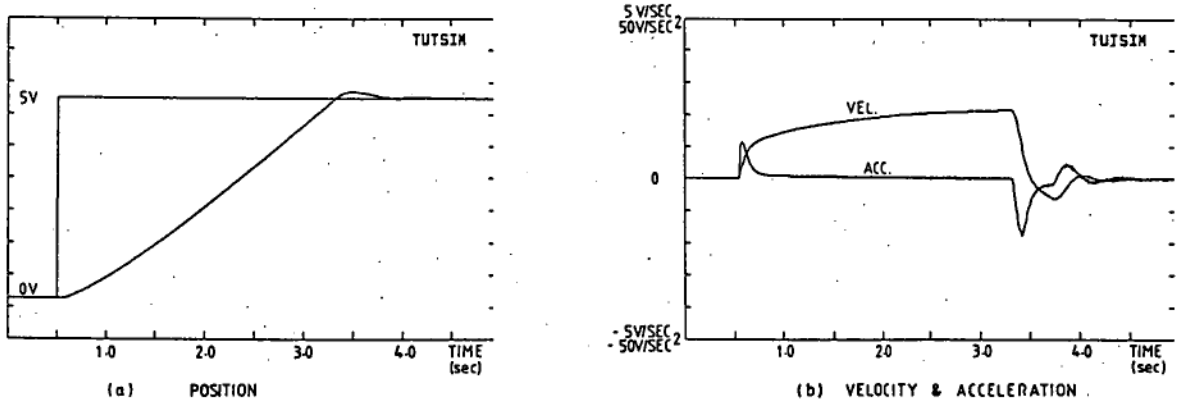


Figure (5.5-6): Simulated Step Response of The Compensated Body Joint System

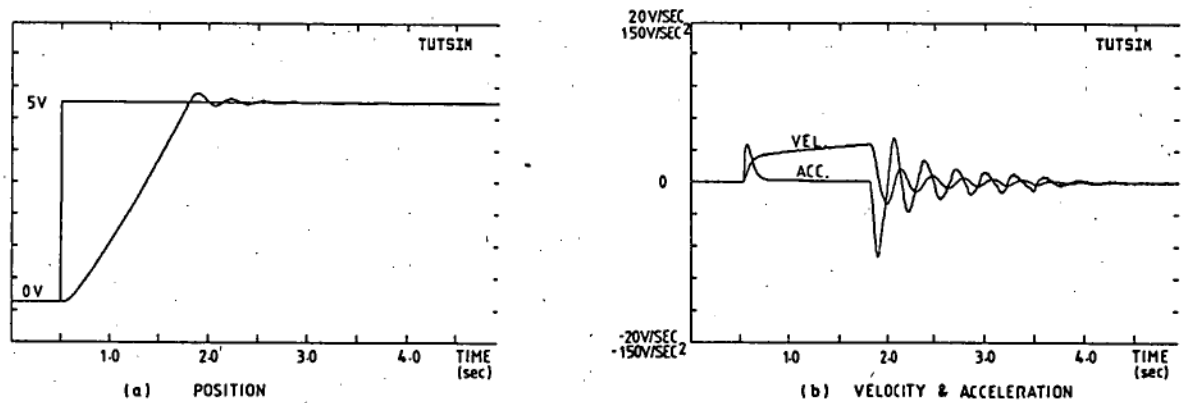


Figure (5.5-7): Simulated Step Response of The Compensated Shoulder Joint System

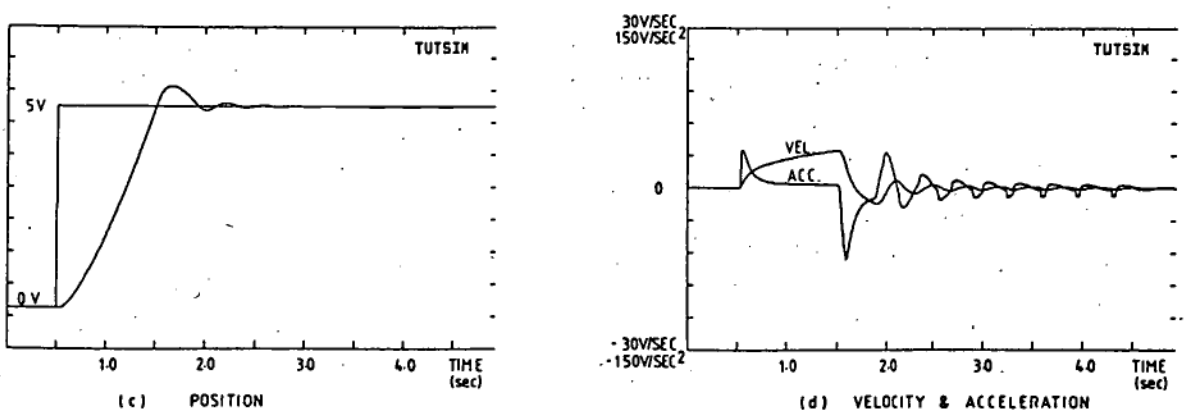


Figure (5.5-8): Simulated Step Response of The Compensated Elbow Joint System

JOINT SYSTEM	STEP INPUT V	MAXIMUM ACCELERATION °/SEC^2	MAXIMUM RETARDATION °/SEC^2	MAXIMUM VELOCITY °/SEC	PERCENTAGE OVERSHOOT %	RISE-TIME SEC	TRANSIENT OSC. TIME SEC	FIGURE SHOWN
BODY	+ 5	155	237	28.6	2.8	2.9	1.0	5.5-6a-b
SHOULDER	+ 5	180	319	21.8	3.9	1.3	2.1	5.5-7a-b
ELBOW	+ 5	155	311	27.8	9.7	1.0	3.1	5.5-8a-b

Table (5.5-5): Step Response Characteristics From Simulated Joint System With Designed Controllers

When the designed compensators were implemented on the joint systems, the actual system responses are shown in Figures (5.5-9a) to (5.5-9c). From these figures, the step response characteristics are estimated and shown in Table (5.5-6).

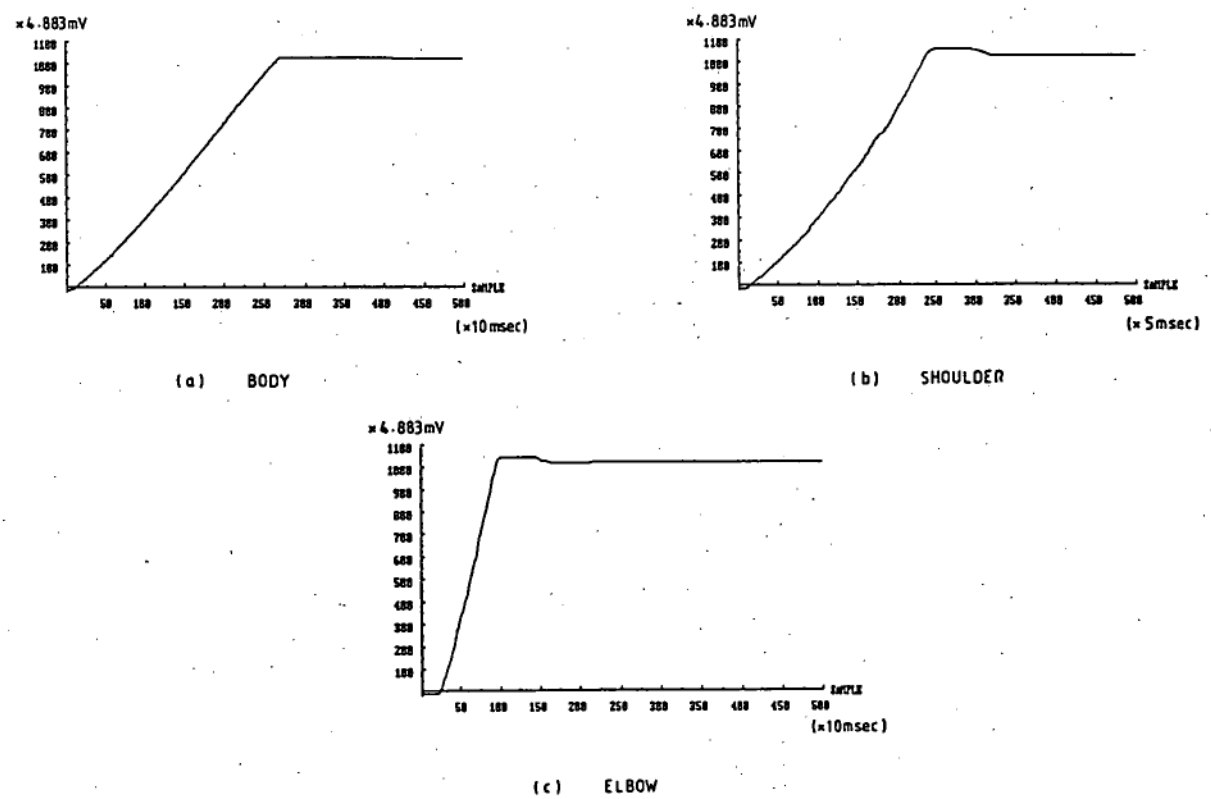


Figure (5.5-9): Actual Step Responses of The Three Compensated Joint Systems

JOINT SYSTEM	PERCENTAGE OVERSHOOT %	RISE - TIME SEC	TRANSIENT OSCILLATION TIME SEC	FIGURE SHOWN
BODY	- 1.0	2.5	1.5	5.5 - 9 a
SHOULDER	3.4	1.3	0.4	5.5 - 9 b
ELBOW	1.7	2.0	1.3	5.5 - 9 c

Table (5.5-6): Actual Step Response Characteristics of The Three Compensated Joint Systems

The characteristics values and shapes of the actual responses and the simulated responses match, yet overall actual responses exhibit slightly less overshoots, less transient time, and faster response.

The design of compensators were based on positive step responses. However, the actual systems are also stable in negative step responses except that the responses are a little different from the positive step responses due to the different disturbance effects. The negative step responses for the actual joint systems are shown in Figure (5.5-10a) to (5.5-10c).

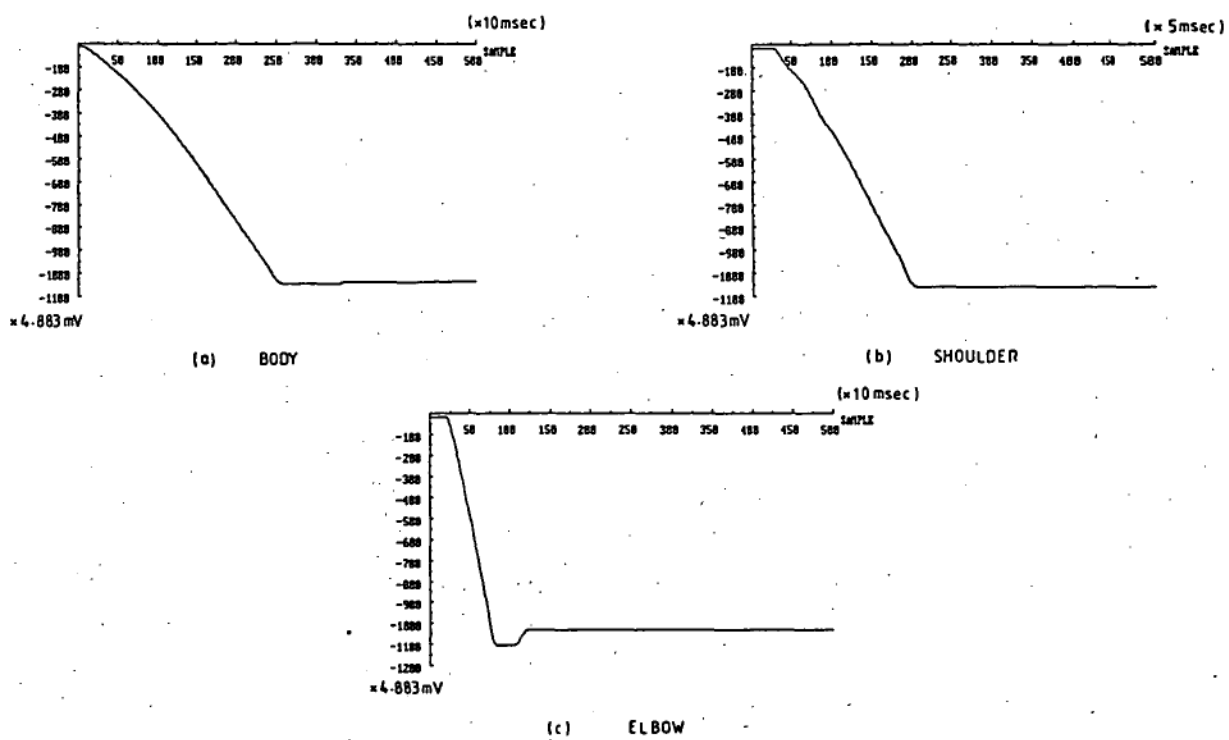


Figure (5.5-10): Actual Step Responses of The Three Compensated Joint Systems With Negative Step

5.6 EFFECTS OF VARIATION OF EFFECTIVE INERTIA

It has been assumed that variations of joint effective inertia has negligible effect on the dynamic response of the closed-loop system. To check this assumption, the discrete-time pulse transfer functions of the joint systems having links configured to give the maximum and the minimum moment of inertias were identified using the same technique discussed in the chapter.

For the body-joint system, the identified pulse transfer function were found to be:-

$$G_{Bmin}(z)=\frac{0.0419657z^2+0.03937502z+0.0131784}{z^3-1.387503z^2+0.3520289z+0.03551182} \qquad (5.6\text{-}1)$$

$$G_{Bmax}(z)=\frac{0.01868824z^2+0.05040691z+0.03426383}{z^3-1.458098z^2+0.5169129z-0.05875563} \qquad (5.6\text{-}2)$$

where the subscripts max and min designate the pulse transfer functions corresponding to the maximum and the minimum moment of inertia respectively.

The designed controller for the body joint was converted into discrete-time using bilinear-transformation[R23]. The step responses in discrete-time were then simulated and are shown in Figure (5.6-1). Similarly, the step responses for the shoulder joint under maximum and minimum moment of inertias were simulated as shown in Figure (5.6-2). As the variations of effective inertia of the elbow joint system is only due to the position change of the small and light weight wrist, the amount of variation can be neglected.

From Figures (5.6-1) and (5.6-2), it can be seen that although the body-joint and the shoulder-joint are subjected to quite large variations of moment of inertias, the effects on closed-loop performance of the joint systems have been significantly reduced by the designed compensators which give large phase- and gian-margins to the joint systems.

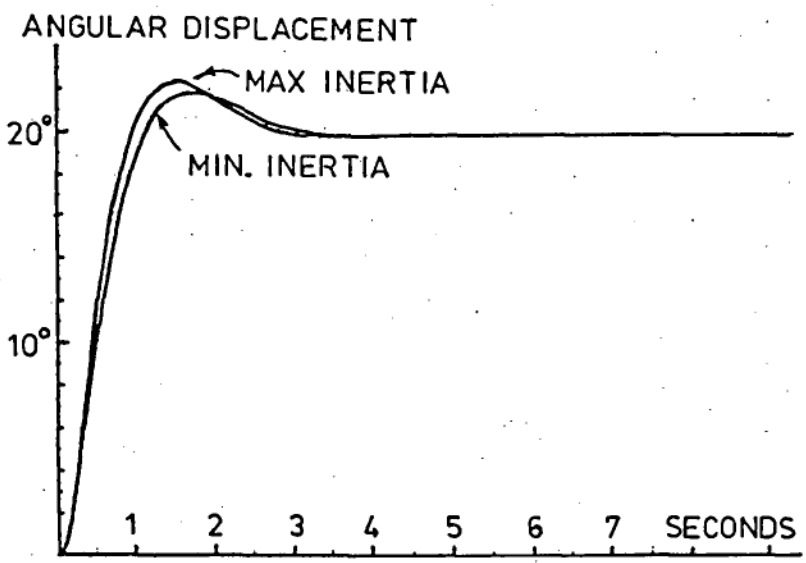


Figure (5.6-1): Step Response of The Body Joint System Under Maximum and Minimum Moment of Inertias

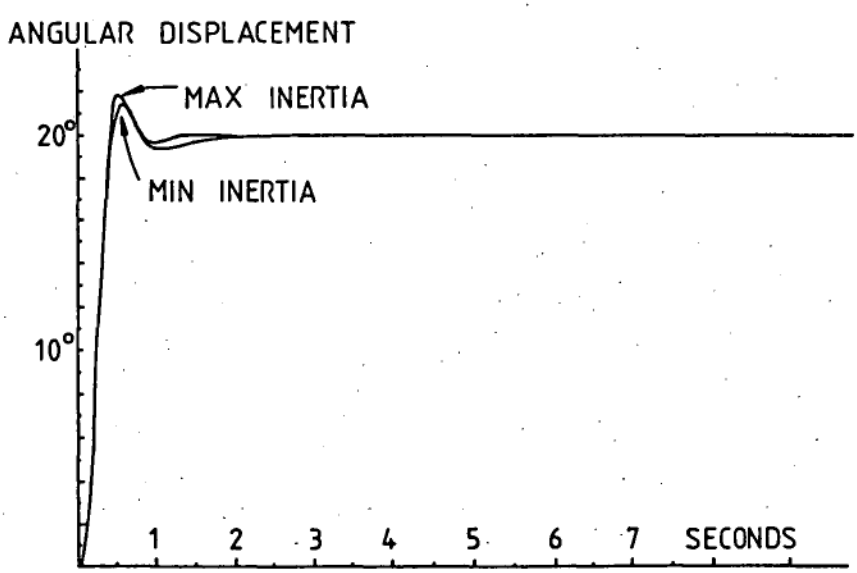


Figure (5.6-2): Step Response of The Shoulder Joint System Under Maximum and Minimum Moment of Inertias

CHAPTER SIX

TRAJECTORY PLANNING

In most robot applications, it is necessary for a manipulator to follow a planned path; in some cases, the manipulator is required to follow the shape of an object on which it is working, in other cases, it has to avoid obstacles during the execution of a task. In order to define the path for a manipulator such that the robot will accurately follow the shape of an object or will safely avoid collisions with obstacles, critical points along the desired path have to be specified by the robot operator.

Positions and orientations of an attached working tool of a robot arm at any point in space can be specified in two different ways. Either, one can specify directly such positions and orientations of the working tool in a coordinate system in which the robot users can visualize and measure, such as the Cartesian coordinates system. Or, one can specify the respective joint positions at that instant such that particular positions and orientations of the working tool can be produced by a combination of the joint positions.

Following the ways the critical points are specified, there are two methods of planning a trajectory. The first method is to plan the trajectory in space coordinates so that the working tool can move along a path containing all the specified point coordinates. The second method is to

plan a trajectory for each joint so that each joint will arrive at its specified corresponding joint positions simultaneously.

The first method results in Cartesian coordinate motion which is a natural consequence of Cartesian coordinates. The manipulator will move along straight lines and rotate about fixed axes in space. This method, however, has a number of disadvantages. It has the burden of converting between Cartesian and joint coordinates during each sampling period. This is necessary because to calculate the errors in the Cartesian path, current joint positions must be converted to their equivalent Cartesian coordinates using Jacobian transformations[R1] (direct kinematics problem). To determine the equivalent errors in joint coordinates, one has to use the inverse Jacobian transformations which converts the current Cartesian coordinates into their equivalent joint positions (inverse kinematics problem). Then, necessary torques for the joints are calculated using the robot's dynamic equations. The complexity of these calculations severely limits the sampling frequency of the robot controller. Also, it is impossible to predict whether a trajectory segment for Cartesian motion will involve excessive joint rates of change before it is executed. It is difficult to estimate motion times and accelerations as Cartesian velocities and accelerations are related to the limiting joint velocities and accelerations in a complicated manner, and depend on the configurations of the manipulator.

The second method, planning the trajectory in joint coordinates, results in joint coordinate motion. This type

of motion is comparatively less expensive both in computation time and effort since complicated transformations are not required during motion execution. Joint accelerations and velocities can be checked against their limits before the motion is executed. A disadvantage of this method is that the resulting Cartesian motion is not defined and straight line path characteristics can no longer be attained. Nevertheless, deviations from a desired Cartesian path can be reduced by specifying more points. Because of the efficient algorithm, trajectories can be planned and implemented in real time. This second method is used to implement the actual motion of the Tasrobot0 system.

Although the planned trajectory is based on specified joint coordinates, specified points can be input by an operator in Cartesian coordinates, which will then be converted into functionally equivalent path in joint coordinates using inverse Jacobian transformation. However, a more direct method is to input the specified points in joint coordinates. This method of specifying data is a natural consequence to robots requiring direct teaching, as in the case of the Tasrobot0 system. During the teaching process, the robot computer will record every joint position specified by the operator.

6.1 PATH APPROXIMATION

In order to approximate a Cartesian path by functions in joint variables, m approximation functions are required for an m-joint manipulator - one for each joint. The approximation function for a particular joint must pass

through all the specified joint coordinates. Also, the function must be continuous in position, velocity and acceleration in order to remain within the physical limitations of the robot. These conditions can be met by deriving a single polynomial which passes through all the specified points; but such a function will be complex and difficult to fit. In addition, it is likely to contain extrema between specified points. These extrema would have to be checked against the limits of the robot.

A better approach is to define a separate polynomial joining two consecutive specified points for each joint, with the constraint that specified corresponding joint coordinates must be reached at the same instant of time. Therefore the corresponding Cartesian path will, at least, pass through all the specified coordinates. This method of connecting data points with smooth curves is widely used in the field of computer graphics. These curves, known as spline functions, have been thoroughly studied and investigated and found to provide the shortest path which satisfies the continuity constraints[R6].

Terms, which are frequently used during the derivations and disscussions of the method are defined as follows:-

Point - a joint coordinate data specified by operator.

Goal point - point specified by operator at which the arm must be temporarily stopped, for example, to open or close the grippers.

Intermdeiate point - critical point in between goal points.This point must be reached during the motion.

Segment - a path joining two consecutive points.

End segment - a path joining one intermediate point and a goal point.

Intermediate segment - a segment joining two consecutive intermediate points.

Section - a path joining two goal point possibly passing through a series of intermediate points.

In general, the motion of the robot may consist of several sections. Each section must contain at least two points, i.e. the start and the end points of a section. For a section consisting of n points, where $n \geq 2$, there are (n-1) segments and (n-2) intermediate points.

For a section with n points, the following notations are used:-

X_1 - starting position of a joint in a section.

X_k - intermediate k^{th} position of a joint within a section.

X_n - end position of a joint in a section.

α_{k+1} - parametric variable with value denoting the time interval taken for a joint to go from position X_k to position X_{k+1}.

$x(\alpha)$ - position of a joint as a function of α, where $0 \leq \alpha \leq \alpha_{k+1}$.

X'_k - velocity of a joint at position X_k.

$x'(\alpha)$ - velocity of a joint as a function of α, where $0 \leq \alpha \leq \alpha_{k+1}$.

X''_k - acceleration of a joint at position X_k.

$x''(\alpha)$ - acceleration of a joint as a function of α, where $0 \leq \alpha \leq \alpha_{k+1}$.

6.2 SPLINE FUNCTION FOR EACH SEGMENT

Cubic splines were used to join all the intermediate points. For the end segments, there is an additional requirement that the velocities and the accelerations at the end points of a section must be zero. Thus, a fourth-order polynomials are required for the end segments.

The proposed method of fitting spline functions to the specified points requires at least five points to define a section, two goal points and three intermediate points. For cases where only three or four points are specified for a section, additional points are inserted to satisfy this requirement. The generation of the additional points are included in the designed software which will be discussed in next chapter. However, in some applications, the robot may only be required to go from one point to the other point, such as in pick-and-place operation in a clear environment. Breaking the two specified points into five or more points in order to satisfy the requirement is unnecessary and leads to slow motion. Therefore, a two point section is also included. For a section with two points, a fifth-order polynomial is used to approximate a two-point section to satisfy the six boundary conditions: two positions specified by operator, zero velocities and accelerations at the two end points.

For the sake of simplicity, the trajectory planning method for a single-joint manipulator is first considered. The method extended to a multi-jointed arm and will be described later in this section.

(a) Cubic Splines For Intermediate Segments

The equation of a cubic spline between two intermediate points X_k and X_{k+1}, where $2 \leq k \leq n-2$, of an n-point trajectory, where $n \geq 5$, consisting of (n-2) spline segments, may be written for a single joint as :

$$x(\alpha) = B_1 + B_2\alpha + B_3\alpha^2 + B_4\alpha^3 \qquad (6.2a\text{-}1)$$

The boundary conditions for this segment are :

$$
\begin{aligned}
x(0) &= X_k \\
x'(0) &= X'_k \\
x(\alpha_{k+1}) &= X_{k+1} \\
x'(\alpha_{k+1}) &= X'_k
\end{aligned}
\qquad (6.2a\text{-}2)
$$

Solving for B's and expressing in matrix form gives:

$$
\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \dfrac{-3}{\alpha_{k+1}^2} & \dfrac{3}{\alpha_{k+1}^2} & \dfrac{-2}{\alpha_{k+1}} & \dfrac{-1}{\alpha_{k+1}} \\ \dfrac{2}{\alpha_{k+1}^3} & \dfrac{-2}{\alpha_{k+1}^3} & \dfrac{1}{\alpha_{k+1}^2} & \dfrac{1}{\alpha_{k+1}^2} \end{bmatrix} \begin{bmatrix} X_k \\ X_{k+1} \\ X'_k \\ X'_{k+1} \end{bmatrix} \qquad (6.2a\text{-}3)
$$

In order to calculate the cubic spline coefficients, α_{k+1}, X'_{k+1} and X'_{k+1} must be determined. The value of α_{k+1} is governed by the velocity and acceleration constraints of the joint. For a system with multi-degree of freedom, the value of α_{k+1} must be the same for every joint in order for the tool-tip to reach the specified position in space. The value of α_{k+1} also depends on the velocity and acceleration constraints of other joints. Its value must be chosen so that the maximum velocity and acceleration of each joint in each segment is within the physical limit of that joint. However, the maximum velocity and acceleration of a joint in a segment can only be determined after the spline function has been established for that joint. Therefore, the value of α_{k+1} must first be expressed in some unit of time to be determined afterward. The value of α_{k+1} is made equal to the norm of $(X_k - X_{k+1})$ as :

$$\alpha_{k+1} = [\Sigma (X_{k+1} - X_k)^2]^{\frac{1}{2}} \qquad (6.2a\text{-}4)$$

where the summation is taken over all the joints.

The value of X'_k and X'_{k+1} can be determined by the acceleration continuity constraint at the intermediate points. For two consecutive segments joining three consecutive intermediate points X_k, X_{k+1} and X_{k+2}, the acceleration at the end of one segment must be equal to the acceleration at the beginning of the next segment.

For segment joining X_k and X_{k+1}, the acceleration at the end point of the segment can be written as :

$$x''(\alpha_{k+1}) = \frac{1}{\alpha_{k+1}}\left[\frac{6(X_k - X_{k+1})}{\alpha_{k+1}} + 2X'_k + 4X'_{k+1}\right] \quad (6.2a\text{-}5)$$

Similarly, for the segment joining X_{k+1} and X_{k+2}, the acceleration at the beginning of the segment can be written as :

$$x''(0) = \frac{1}{\alpha_{k+2}}\left[\frac{3(X_{k+2} - X'_{k+1})}{\alpha_{k+2}} - 2X'_{k+1} - X'_{k+2}\right] \quad (6.2a\text{-}6)$$

Equating equations (6.2a-5) and (6.2a-6), and rearranging terms gives :

$$\alpha_{k+2}X'_k + 2(\alpha_{k+1}+\alpha_{k+2})X'_{k+1} + \alpha_{k+1}X''_{k+2} = \frac{3}{\alpha_{k+1}\alpha_{k+2}}\left[\alpha_{k+1}^2(X_{k+2} - X_{k+1}) + \alpha_{k+2}^2(X_{k+1} - X_k)\right] \quad (6.2a\text{-}7)$$

Expressing equation (6.2a-7) for all intermediate segments in matrix form yields :

$$\begin{bmatrix} \alpha_4 & 2(\alpha_3+\alpha_4) & \alpha_3 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \alpha_5 & 2(\alpha_4+\alpha_5) & \alpha_4 & 0 & \dots & 0 & 0 & 0 \\ & & \dots\dots\dots\dots & & & & & & \\ 0 & \dots\dots & 2(\alpha_{n-3}+\alpha_{n-4}) & \alpha_{n-3} & 0 & & & & \\ 0 & \dots\dots & \alpha_{n-1} & 2(\alpha_{n-2}+\alpha_{n-1}) & \alpha_{n-2} & & & & \end{bmatrix} \begin{bmatrix} X'_2 \\ X'_3 \\ \dots \\ X'_{n-2} \\ X'_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{\alpha_4 \alpha_3}\left[\alpha_3^2 (X_4 - X_3) + \alpha_4^2 (X_3 - X_2)\right] \\ \frac{3}{\alpha_5 \alpha_4}\left[\alpha_4^2 (X_5 - X_4) + \alpha_5^2 (X_4 - X_3)\right] \\ \vdots \\ \frac{3}{\alpha_{n-1} \alpha_{n-2}}\left[\alpha_{n-2}^2 (X_{n-1} - X_{n-2}) + \alpha_{n-1}^2 (X_{n-2} - X_{n-3})\right] \end{bmatrix} \qquad (6.2a\text{-}8)$$

Equation (6.2a-8) represents a system of (n-5) algebraic equations with (n-3) unknowns. In order to obtain a solution to equation (6.2a-8), it is necessary to state the boundary conditions at the end points of the cubic spline functions. These end point conditions are provided by the two end segments and will be derived in the following subsections.

(b) Fourth-order Spline For The Beginning And End Segments

For the first and last segments of the trajectory, additional constraints of zero velocity and zero acceleration are required at the end points (goal points), i.e.

$$X'_1 = X''_1 = X'_n = X''_n = 0 \qquad (6.2b\text{-}1)$$

This requires a fourth-order spline segment of the form:

$$x(\alpha) = B_1 + B_2 \alpha + B_3 \alpha^2 + B_4 \alpha^3 + B_5 \alpha^4 \qquad (6.2b\text{-}2)$$

For the first segment, the boundary conditions are:

$$\begin{aligned} x(0) &= X_1 \\ x(t_2) &= X_2 \\ x'(0) &= 0 \\ x'(t_2) &= X'_2 \\ x''(0) &= 0 \end{aligned} \qquad (6.2b\text{-}3)$$

Solving for B's in the first segment, gives:

$$B_1 = X_1 \qquad (6.2b\text{-}4)$$

$$B_2 = 0 \qquad (6.2b\text{-}5)$$

$$B_3 = 0 \qquad (6.2b\text{-}6)$$

$$B_4 = \frac{4}{\alpha_2^3}(X_2 - X_1) - \frac{1}{\alpha_2^2}X'_2 \qquad (6.2b\text{-}7)$$

$$B_5 = \frac{3}{\alpha_2^4}(X_1 - X_2) + \frac{1}{\alpha_2^3}X'_2 \qquad (6.2b\text{-}8)$$

The acceleration at the end point of the first segment may be written as :

$$x''(\alpha_2) = \frac{12}{\alpha_2^2}(X_1 - X_2) + \frac{6}{\alpha_2}X'_2 \qquad (6.2b\text{-}9)$$

The initial acceleration of the second segment can be written as :

$$x''(0) = \frac{2}{\alpha_3}\left[\frac{3(X_3 - X_2)}{\alpha_3} - 2X'_2 - X'_3\right] \qquad (6.2b\text{-}10)$$

From the acceleration continuity constraint, the acceleration at the end point of the first segment must equal to the acceleration at the beginning of the second segment, and hence:

$$(2\alpha_2 + 3\alpha_2)X'_2 + \alpha_2 X'_3 = \frac{3}{\alpha_2 \alpha_3}\left[\alpha_2^2(X_3 - X_2) + 2\alpha_3^2(X_2 - X_1)\right] \qquad (6.2b\text{-}11)$$

Similarly, for the last segment, the boundary conditions are:

$$\begin{aligned} x(0) &= X_{n-1} \\ x(\alpha_n) &= X_n \\ x'(0) &= X'_{n-1} \\ x'(\alpha_n) &= 0 \end{aligned} \qquad (6.2b\text{-}12)$$

$$x''(\alpha_n) = 0$$

Solving for B's in the last segment gives:

$$B_1 = X_{n-1} \qquad (6.2b\text{-}13)$$

$$B_2 = X'_{n-1} \qquad (6.2b\text{-}14)$$

$$B_3 = \frac{1}{\alpha_n^2}\left[6(X_n - X_{n-1}) - 3X'_{n-1}\alpha_n\right] \qquad (6.2b\text{-}15)$$

$$B_4 = \frac{1}{\alpha_n^3}\left[8(X_{n-1} - X_n) + 3X'_{n-1}\alpha_n\right] \qquad (6.2b\text{-}16)$$

$$B_5 = \frac{1}{\alpha_n^4}\left[3(X_n - X_{n-1}) - X'_{n-1}\alpha_n\right] \qquad (6.2b\text{-}17)$$

Again, from the acceleration continuity constraint, the acceleration at the end point of the second last segment must be equal to the acceleration at the beginning of the last segment, and it can be shown that:

$$\alpha_n X'_{n-2} + (3\alpha_{n-1} + 2\alpha_n)X'_{n-1} = \frac{3}{\alpha_{n-1}\alpha_n}\left[2\alpha_{n-1}^2(X_n - X_{n-1}) + \alpha_n^2(X_{n-1} - X_{n-2})\right] \qquad (6.2b\text{-}18)$$

Combining the results of equations (6.2a-8), (6.2b-11) and (6.2b-18) allows the unknown intermediate point velocities $X'_2, X'_3, \ldots\ldots X'_{n-1}$, to be solved; and these (n2) equations can be expressed as :

$$\begin{bmatrix} 2(\alpha_2+3\alpha_3) & \alpha_2 & 0 & \cdots\cdots & 0 \\ \alpha_4 & 2(\alpha_3+\alpha_4) & \alpha_3 & 0 \ \cdots & 0 \\ 0 & \alpha_5 & 2(\alpha_4+\alpha_5) & \alpha_4 \ \cdots & 0 \\ \cdots\cdots & & & & \\ 0 & \cdots\cdots & 2(\alpha_{n-3}+\alpha_{n-4}) & \alpha_{n-3} & 0 \\ 0 & \cdots\cdots & \alpha_{n-1} & 2(\alpha_{n-2}+\alpha_{n-1}) & \alpha_{n-2} \\ 0 & \cdots\cdots & 0 & \alpha_n & (3\alpha_{n-1}+2\alpha_n) \end{bmatrix} \begin{bmatrix} X'_2 \\ X'_3 \\ X'_4 \\ \cdots \\ X'_{n-3} \\ X'_{n-2} \\ X'_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{\alpha_2 \alpha_3}\left[\alpha_2^2 (X_3 - X_2) + 2\alpha_2^3 (X_2 - X_1)\right] \\ \frac{3}{\alpha_3 \alpha_4}\left[\alpha_2^3 (X_4 - X_3) + \alpha_2^4 (X_3 - X_2)\right] \\ \frac{3}{\alpha_4 \alpha_5}\left[\alpha_2^4 (X_5 - X_4) + \alpha_2^5 (X_4 - X_3)\right] \\ \vdots \\ \frac{3}{\alpha_{n-2} \alpha_{n-1}}\left[\alpha_{n-2}^2 (X_{n-1} - X_{n-2}) + \alpha_{n-1}^2 (X_{n-2} - X_{n-3})\right] \\ \frac{3}{\alpha_{n-1} \alpha_n}\left[2\alpha_{n-1}^2 (X_n - X_{n-1}) + \alpha_n^2 (X_{n-1} - X_{n-2})\right] \end{bmatrix} \quad (6.2b\text{-}19)$$

Or, symbolically:

$$[A][X'] = [D] \quad (6.2b\text{-}20)$$

where [A] is an (n-2)x(n-2) matrix

[X'] is an (n-2)x1 matrix

[D] is an (n-2)x1 matrix

[X'] can be obtained by solving the system of linear algebraic equations shown in equation (6.2b-19). Since the [A] matrix is a tridiagonal matrix, there is a very efficient algorithm for solving this particular type of system of equations[R29]. Given the equations:

$$\begin{aligned} b_1 z_1 + c_1 z_2 &= d_1, \\ a_2 z_1 + b_2 z_2 + c_2 z_3 &= d_2, \\ \vdots \quad & \quad \vdots \quad \quad \vdots \\ a_{n-1} z_{n-2} + b_{n-1} z_{n-1} + c_{n-1} z_n &= d_{n-1}, \\ a_n z_{n-1} + b_n z_n &= d_n, \end{aligned} \quad (6.2b\text{-}21)$$

operators u_k, v_k and w_k can be formed such that:

$$u_k = a_k v_{k-1} + b_k, \qquad (v_0 = 0) \quad (6.2b\text{-}22)$$

$$v_k = -c_k / u_k, \quad (6.2b\text{-}23)$$

$$w_k = (d_k - a_k w_{k-1})/u_k, \quad (w_0 = 0) \qquad (6.2b\text{-}24)$$

for k = 1,2,3,.....,n.

Successive elimination of $z_1, z_2, ..., z_{n-1}$ from 2nd, 3rd,....., nth equations yields the equivalent system equation:

$$z_k = v_k z_{k+1} + w_k \quad , \text{ for } k = 1,2,....n-1 \qquad (6.2b\text{-}25)$$

$$z_n = w_n \qquad (6.2b\text{-}26)$$

whence $z_n, z_{n-1}, ...z_1$ can be successively evaluated.

For matrices with dominant main diagonal, this procedure is stable[R26] in the sense that all errors rapidly damp out ($0 < c_k/u_k < 1$).

Solving equation (6.2b-19) allows coefficients of the spline functions for each segment to be found. These results for a single joint system can be extended to a multi-degree of freedom system with m joints simply by duplicating the spline function generation procedure for each joint and using the same values of α_{k+1}. The [A] matrix for multi-joint system will be unaltered, and [X'] and [D] matrices will become (n-2)xm matrices. A path through n points for a robot with m joints will consist of 2m unique fourth-order spline functions and mx(n-3) unique cubic spline functions.

The efficient alogorithm discussed above for solving single-joint systems is also applicable to multi-joint systems. Since equations (6.2b-22) and (6.2b-23) do not depend on the d's in equations (6.2b-21), and the [A] matrix is unchanged, the intermediate velocities for other joints can be evaluated using the same alogrithm and the same values of u_k and v_k. Only the values of w_k are renewed for each joint.

(c) Fifth-order Spline Function For A Two-point Section

For a section with only two defining points, i.e. only the goal points, there are six boundary conditions to be satisfied A fifth order spline function is required to specify the section. The form of the fifth-order spline is given by:

$$x(\alpha) = B_1 + B_2\alpha + B_3\alpha^2 + B_4\alpha^3 + B_5\alpha^4 + B_6\alpha^5 \qquad (6.2c\text{-}1)$$

The boundary conditions required are :

$$\begin{aligned} x(0) &= X_1 \\ x(\alpha 2) &= X_2 \\ x'(0) &= 0 \\ x'(\alpha_2) &= 0 \\ x''(0) &= 0 \\ x''(\alpha_2) &= 0 \end{aligned} \qquad (6.2c\text{-}2)$$

Substituting equations (6.2c-2) into (6.2c-1) and solving for B's gives :

$$B_1 = X_1 \qquad (6.2c\text{-}3)$$

$$B_2 = 0 \qquad (6.2c\text{-}4)$$

$$B_3 = 0 \qquad (6.2c\text{-}5)$$

$$B_4 = \frac{10(X_2 - X_1)}{\alpha_2{}^3} \qquad (6.2c\text{-}6)$$

$$B_5 = \frac{-15(X_2 - X_1)}{\alpha_2{}^4} \qquad (6.2c\text{-}7)$$

$$B_6 = \frac{6(X_2 - X_1)}{\alpha_2{}^5} \qquad (6.2c\text{-}8)$$

6.3 TIME SCALE FACTOR

Once the spline functions are defined for all manipulator joints, the parametric variable α defined in

equation (6.2a-4) must be associated with some unit of physical time. This can be done by replacing the parametric variable α in the spline segments by t/S, where t is the elapsed physical time since the beginning of the spline segment and S is a scale factor to determined.

Without loss of generality, a cubic spline segment is considered:

$$x(\alpha) = B_1 + B_2\alpha + B_3\alpha^2 + B_4\alpha^3 \qquad (6.3\text{-}1)$$

Replacing α by t/S in equation (6.3-1), gives :

$$x(t) = C_1 + C_2 t + C_3 t^2 + C_4 t^3 \qquad (6.3\text{-}2)$$

where
$$C_1 = B_1$$
$$C_2 = B_2/S$$
$$C_3 = B_3/S^2 \qquad (6.3\text{-}3)$$
$$C_4 = B_4/S^3$$

The scale factor, S, must be assigned one value for each section, such that S is large enough to prevent the maximum velocity or acceleration to be exceeded by any of the robot's joints during the execution of the section. Also, it should be as small as possible in order that the motion of that section could be completed in a reasonable short time. Therefore, the maximum value of S necessary to stay within the physical limits of the robot's joints must be found.

The real time velocity can be expressed as :

$$x'(t) = \left.\frac{x'(\alpha)}{S}\right|_{\alpha = t/S} \qquad (6.3\text{-}4)$$

In order to stay within the velocity constraint, the real time velocity for a joint must stay below the velocity constraint, V, of that joint during motion. Hence, S must be

greater than or equal to x'(α)/V. To stay within the velocity constraint, the minimum value of S, S_v, is given by:

$$S_v = \max_j \left\{ \frac{\max_i \left[\max X'_{ij} \right]}{V_j} \right\} \qquad (6.3\text{-}5)$$

where $\max X'_{ij}$ represents the maximum velocity of the i^{th} spline segment for the j^{th} joint, and V_j represents the velocity constraint for the j^{th} joint.

Similarly, the real time acceleration for a joint can be written as :

$$x''(t) = \left. \frac{x''(\alpha)}{S^2} \right|_{\alpha = t/S} \qquad (6.3\text{-}6)$$

To stay within the acceleration constraint, the minimum value of S, S_a, is:

$$S_a = \max_j \left\{ \frac{\max_i \left[\max X''_{ij} \right]}{A_j} \right\}^{1/2} \qquad (6.3\text{-}7)$$

where $\max X''_{ij}$ represents the maximum acceleration of the i^{th} spline segment for the j^{th} joint; and A_j represents the acceleration constraint for the jth joint.

In general, revolute joints, particularly those with horizontal axes of rotation, have different velocity constraints in different direction of rotation. Their acceleration and retardation constraints are also different due to the effects of external torques. Therefore, there are two values of velocity constraints - one for each direction, and two values of acceleration constraints - one for acceleration and one for retardation, for every revolute joint. This results in four values of S necessary to stay

within the constraints - two for velocity constraints, denoted by S_{v1} and S_{v2} and two for acceleration constraints denoted by S_{A1} and S_{A2}.

In some cases, there may also be time constraint in addition to velocity and acceleration constraints mentioned above. This time constraint is mainly to adjust the motion execution time so that the robot can be synchronized with the working environment. In the case of the Tasrobot0 system, the time constraint is introduced to synchronize the positioning process of the body-, shoulder- and elbow-joints with the orientation process of the pitch and the roll rotations. These two processes in the Tasrobot0 system are controlled individually but carried out at the same time.

If T_{ij} represents the minimum time required by the i^{th} segment of the j^{th} joint to carry out its motion, then the minimum value of S necessary to give the required amount of time, S_T, is:

$$S_T = \max_j \left\{ \max_i \left[\frac{T_{ij}}{\alpha_{i+1}} \right] \right\} \qquad (6.3\text{-}8)$$

In some applications, the time constraint may only be imposed on the total time taken for the motion of a section. For example, during pick-and-place operations, for stepper motor controlled wrist joints, only the wrist joint positions at the end points of a section are essential. That is, only the number of steps for each wrist joint from initial position to final position of a section will be required.

If T_j represents the minimum time required for the j^{th} joint to complete a section, then the minimum value of S necessary to give the required amount of time, S_T, is:

$$S_T = \underset{j}{MAX} \left\{ \frac{T_j}{\Sigma \alpha_{i+1}} \right\} \qquad (6.3\text{-}9)$$

During the trajectory planning process, either equation (6.3-8) or equation (6.3-9) can be used to specify the time constraint.

The desired minimum value of the scale factor S can be obtained by choosing the maximum value amoung the five constraints, i.e.

$$S = MAX\{S_{V1}, S_{V2}, S_{A1}, S_{A2}, S_T\} \qquad (6.3\text{-}110)$$

Once S is evaluated, the spline functions in real time, t, of each segment for each joint can be re-established, as shown in equations (6.3-2) and (6.3-3). Although only third order spline functions are shown, the fourth- and the fifth-order polynomial functions are similar.

6.4 EVALUATION OF MAXIMUM VELOCITY AND ACCELERATION FOR SPLINE SEGMENTS

The evaluation of S requires maximum velocity and acceleration to be evaluated for each spline segment of each joint within a specified section. To find the maximum velocity or acceleration for each joint, the parametric variable, α_{max}, at which maximum velocity or acceleration occurs should be determined first. In cases where the calculated α_{max} falls outside the α-range of a spline segment, the maximum value will occur at either end points of that spline segment.

Maximum velocity occurs within the α range of the i^{th} segment for the j^{th} joint can be expressed as :

$$\max X'_{ij} = x'_{ij}(\alpha)\,|_{\alpha=\alpha_{vmax}} \qquad (6.4\text{-}1)$$

where α_{vmax} is the parametric variable, α, at which velocity is maximum and $0 \le \alpha_{vmax} \le \alpha_{i+1}$

Similarly, maximum acceleration within the i^{th} segment for the j^{th} joint is given by :

$$\max X''_{ij} = x''_{ij}(\alpha)\,|_{\alpha=\alpha_{amax}} \qquad (6.4\text{-}2)$$

where α_{amax} is the parametric variable, α, at which acceleration is maximum and $0 \le \alpha_{amax} \le \alpha_{i+1}$.

Table (6.4-1) shows the values of α_{vmax} and α_{amax} for each segment of a section.

	i^{th} INTERMEDIATE SEGMENT	FIRST SEGMENT	LAST SEGMENT	TWO-POINT SECTION
t_{vmax}	$-B_3/3B_4$	$-B_4/2B_5$	$B_3/6B_5 t_n$	$t_2/2$
t_{amax}	t_{i+1}	$-B_4/4B_5$	$-B_4/4B_5$	$\frac{1}{2}(1-\frac{1}{\sqrt{3}})t_2$

Table (6.4-1): Equations For Maximum Velocities And Maximum Accelerations To Occur In Segment Functions

6.5 METHODS OF IMPLEMENTATION

The trajectory control process can be divided into two stages. The first stage, which is the trajectory planning stage, is to design a desired path from the specified points by calculating the necessary coefficients of each spline or polynomial segment function for each joint. The second stage is to implement the desired path, i.e. the path execution stage.

The first stage is implemented off-line. That is, a desired path is planned after all the specified points for a

path are input. The design of a path for a given section using the discussed technique can be summarized by a series of steps. Firstly, the parametric value α, as shown in equation (6.2a-4), is caluclated. Secondly, the intermediate velocities, as shown in equation (6.2b-19), are evaluated so that the coefficients of each approximating segment function can be calculated; this step is not required for a two-point section. Thirdly, the maximum velocities and accelerations are calculated from the evaluated segment functions so that the time scale factor can be determined. Then, from the determined time scale factor, the real time required to execute each segment function can be found; ; nd the segment functions can be converted into real time functions. These steps can be summerized in the flow-chart shown in Figure (6.5-1).

The spline segment functions for a desired path obtained from the trajectory planning stage are real time functions. Although the functions are continuous in nature, the representation of the functions by digital computer can only be in discrete points. The more the points are used, the more accurate the approximated function will be. However, the more points used, the more memory for storing the points are required. As an example, for a three-joint system, 9 additions and about 15 multiplications are required to compute a set of joint data. The 8087 numeric processor unit, which is installed in the host computer of the Tasrobot0 system, can compute an addition in about 21 microseconds and a multiplication in 24 microseconds. The total time required to calculate a set of joint data for the

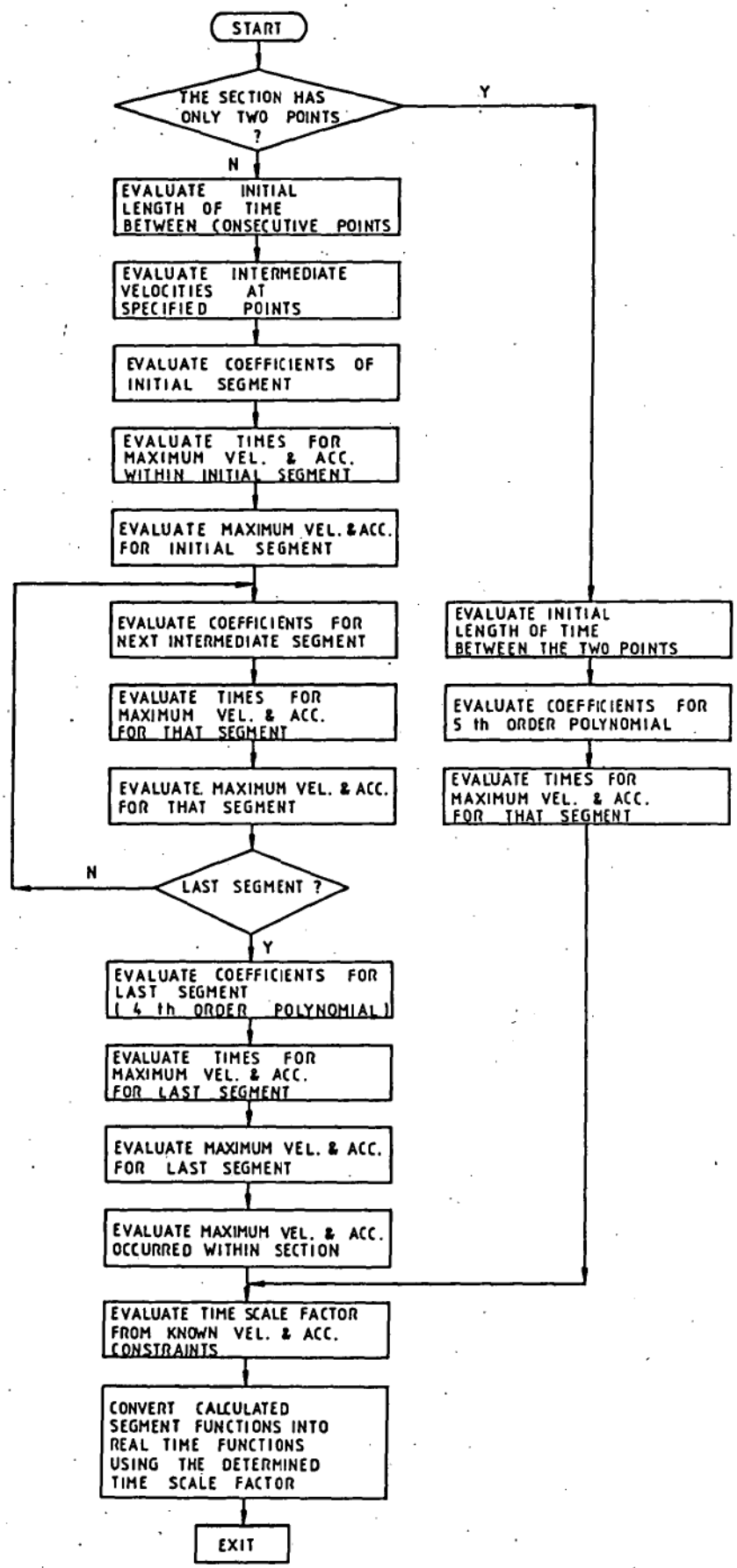


Figure (6.5-1): Flow-chart of The Trajectory Planning Process

three-joint system is about 549 microseconds. For a job requiring 30 seconds to complete, the total number of data calculated is more than 163,000. A large memory space will be required to store the necessary joint data for that job if off-line method is used. On the other hand, if the path execution stage is implemented on-line, only the coefficients of the segment functions required to be stored and generating a command data set in 549 microseconds is sufficiently fast for controlling most mechanical systems. Therefore, the path execution stage is implemented using on-line method.

The implementation of the path execution stage requires a real time clock which provide real time values. Upon executing each segment function, the clock is reset and started. Joint position data are then calculated by reading the real time values from the clock and substituting into the corresponding real time segment function. The data calculated is output to that joint. A flow-chart of the path execution process is shown in Figure (6.5-2).

Figures (6.5-1) and (6.5-2) only summarize the technique of the trajectory control process. A more detailed account for the actual implementation of the process will be discussed in next chapter.

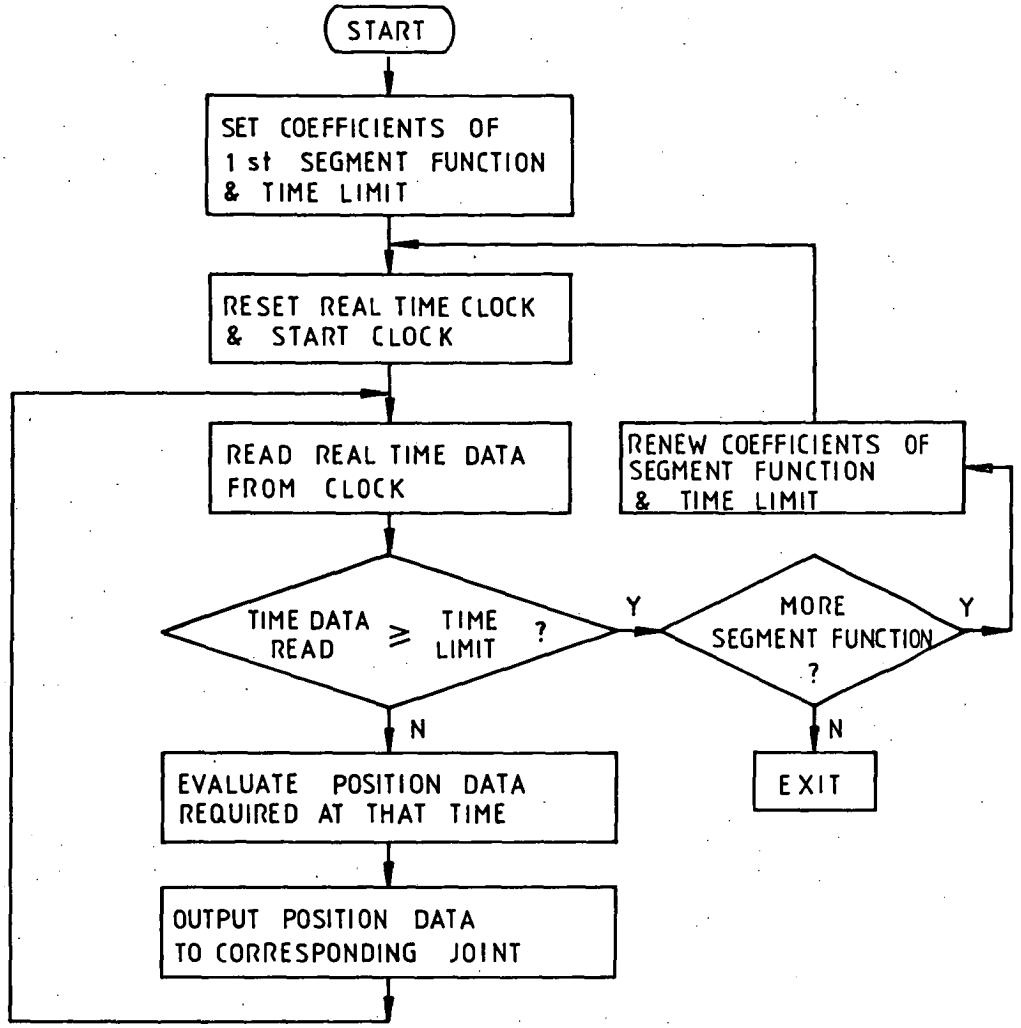


Figure (6.5-2): Flow-chart of The Path Execution Process

CHAPTER SEVEN

SOFTWARE CONTROL OF THE TASROBOT0 SYSTEM

The software control of the Tasrobot0 system consists of eight main programs written in Fortran-77 language. These programs can be executed on the host computer of the Tasrobot0 system, which is an IBM/PC microcomputer. The programs are interconnected by a batch file called TASROBOT.BAT, to form the software control system of the Tasrobot0 manipulator.

In addition to the Lab-pac subroutines, seven public subroutines written in 8088 assembly language and Fortran-77 language were developed to access to hardwares in the IBM microcomputer and hardwares in the Lab Master interface board, and to increase program execution speed.

7.1 SUBROUTINES

There are six assembly language subroutines: STPORT, DIGOUT, POSITN, WRIST, ARMOUT and BEEP; and one Fortran subroutine, SELFADJ. These subroutines will be discussed in this section. The Lab-pac subroutines will not be discussed as they are detailed in its user' guide[R32]. There are specific procedures required in the assembly language subroutines for linking with the eight Fortran main programs are discussed in the Microsoft Fortran Complier User's Guide[R34].

The listing of these subroutines can be found in Appendix B.

(a) The STPORT Subroutine

This subroutine reprograms the 8255 parallel port in the Lab Master Board so that its 24 I/O pins can be re-arranged as 8 bit-wide input port and 16 bit-wide output port. This call is necessary to override the initialization call of the Lab-pac subroutines which uses 16 bit-wide input port and 8 bit-wide output port.

(b) The DIGOUT Subroutine

The subroutine call is in the format DIGOUT(CMD,LATCH). CMD is a 12-bit integer command signal to be output to control the motion of a joint. LATCH is a 3-bit integer address of a joint register as shown in Table (7.1b-1).

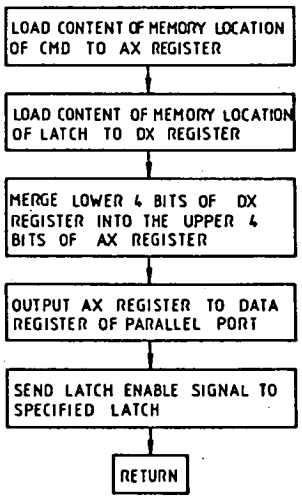
The two input values, CMD and LATCH, are concatenated into a 16-bit data such that the lower 12 bits (bit0-bit11) contains the command signal and the upper 3 bits (bit12-bit14) contains the address signal. The latch enable signal for a specified register must be in the correct sequence as described in section 3.3a. Figure (7.1b-1) illustrates the flow chart of the DIGOUT subroutine.

(c) The POSITN, WRIST and ARMOUT Subroutines

The functions of these subroutines are very similar to functions of the DIGOUT subroutine except that the command signals for these subroutines are in integer array form. The POSITN subroutine call is in the format POSITN(CMDP), where CMDP is an integer array of three elements representing the

LATCH NO.	ENERGIZED JOINT
0	BODY
1	WRIST / PITCH
2	SHOULDER
3	WRIST / ROLL
4	ELBOW
5	GRIPPER

Table(7.1b-1): Joint Energized Corresponding To Latch Number



Figure(7.1b-1): Flow-chart of The DIGOUT Subroutine

command signal for the body-, shoulder- and elbow-joints respectively.

The wrist subroutine call is in the format WRIST(CMDP), where CMDP is an integer array of two elements representing the command signals for the pitch and roll rotations respectively.

The ARMOUT subroutine call is in the format ARMOUT(CMDP), where CMDP is an integer array of five elements representing the command signal for the body-, shoulder- and elbow-joints as well as the pitch and roll rotations repectively.

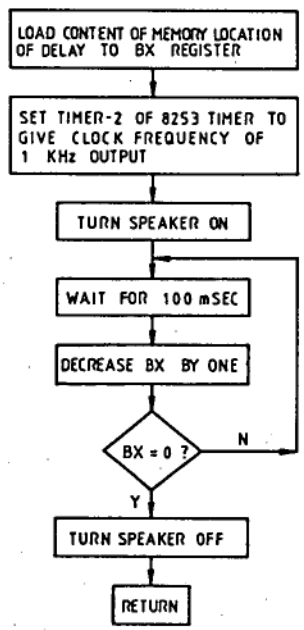
The main purpose of these subroutines is to reduce the time in outputing command signals. Since, in many cases, more than one joint are energized at a time, these subroutines reduce the need of calling the DIGOUT subroutine recursively.

(d) The BEEP Subroutine

The subroutine BEEP accesses the sound generating hardwares inside the IBM microcomputer. The subroutine call is of the format BEEP(DELAY), where DELAY is an integer

constant or integer variable representing the duration of the sound in multiples of 0.1sec.

This subroutine is used in programming or teaching stage to acknowledge the operator that positions and orientations specified by the operator are stored. It is also used to remind the operator that the memory assigned for storing joint data are running low or full. Audio signal is considered to be more effective than visual display during teaching stage since the operator can devote his time in adjusting proper arm configuration without having to scan the screen frequently for warning signals. Figure (7.1d-1) illustrates the flow chart of the BEEP subroutine.

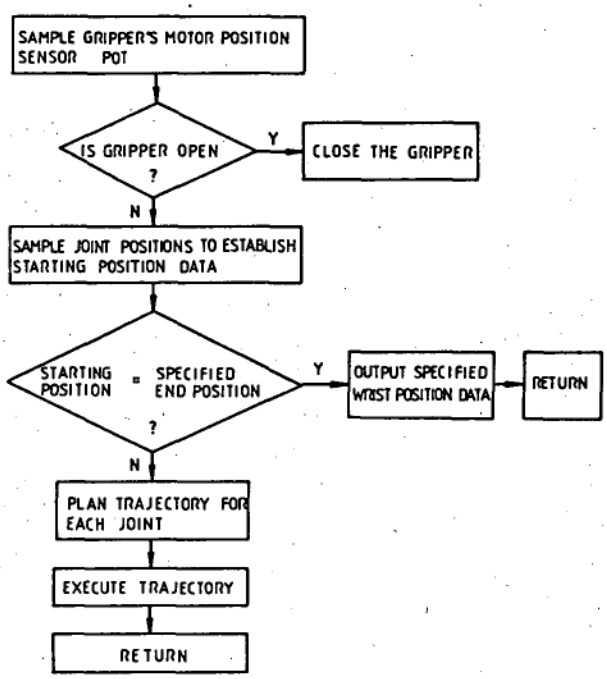


Figure(7.1d-1): Flow-chart of The BEEP Subroutine

(e) The SELFADJ Subroutine

This subroutine is written in Fortran-77 language. It brings the manipulator from its current position to a specified position. This function is required in both training mode and operating mode. In training mode, the manipulator has to be aligned with the teaching arm. In operating mode, the manipulator has to be brought to the starting position of a specified task. The SELFADJ subroutine implements the trajectory control algorithm, discussed in Chapter 6, in real time.

The subroutine call is in the format SELFADJ(F2), where F2 is an integer variable array containing specified positions of the five manipulator joints. The current position of the arm is obtained by sampling the position sensors mounted on the manipulator.



Figure(7.1e-1): Flow-chart of The SELFADJ Subroutine

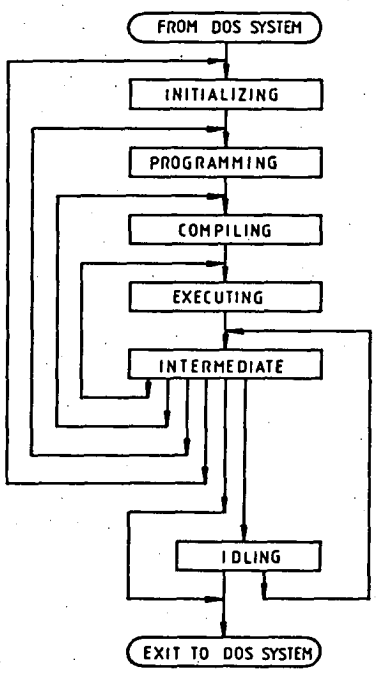
Owing to the inherent dependence of the effective length of the gripper steel string on arm configuration, as discussed in section 2.1c, the subroutine will always ensure that the gripper is in closed position before any motion is executed. Figure (7.1e-1) illustrates the flow chart of the SELFADJ subroutine.

7.2 MAIN PROGRAMS

The eight main programs for the software control of the Tasrobot0 manipulator can be divided into six stages, namely, the initializing stage, the programming stage, the compiling stage, the executing stage, the intermediate stage and the idling stage.

The initializing stage enables the manipulator to be programmed for a specific task. Data specified in the programming stage will be trimmed and processed in the compiling stage to generate executable data for the executing stage. After the completion of a task, options are

provided in the intermediate stage to re-direct to one of the stages. When the robot is not intended to be used in a short period, it can be put in to the idling stage. Then, the robot will not response, but the power for the electronics will be maintained to enable the system to be reactivated without going through the initializing stage.



Figure(7.2-1): Software Control System of The Tasrobot0 Manipulator

Each stage consists of one or more main programs and the ending of one stage can trigger the starting of next by an autoexecuting batch file, called TASROBOT.BAT. By running this batch file, each of the eight programs will be initialized in a sequence illustrated in Figure (7.2-1).

(a) The Initializing Stage

This stage consists of program ALIGN. This program provides step by step instructions for setting the reference positions of the pitch and roll axes of the wrist which are shown in Figure (7.2a-1). These reference settings are important to produce absolute position codes.

The pitch rotation has a range of motion of 140° and the roll rotation has a range of motion of 270° as shown in Figure (7.2a-2). Since each stepper motor has step angle of 0.6°, the pitch position can be discretized into 234 levels

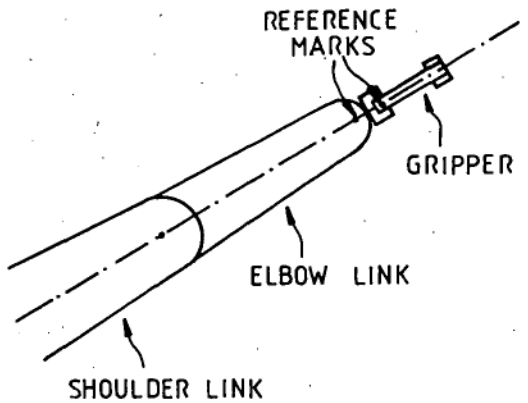
and the roll position into 450 levels, with each level equivalent to one step angle as shown in Table (7.2a-1).

After the reference positions are set, the codes 100 decimal and 150 decimal are sent to the pitch and the roll registers respectively. It is important that these codes are sent after the registers are powered; otherwise, these signals will be lost. Therefore, the power ON signal will be checked before the reference position codes are sent. Flow chart of the ALIGN program is shown in Figure (7.2a-3).

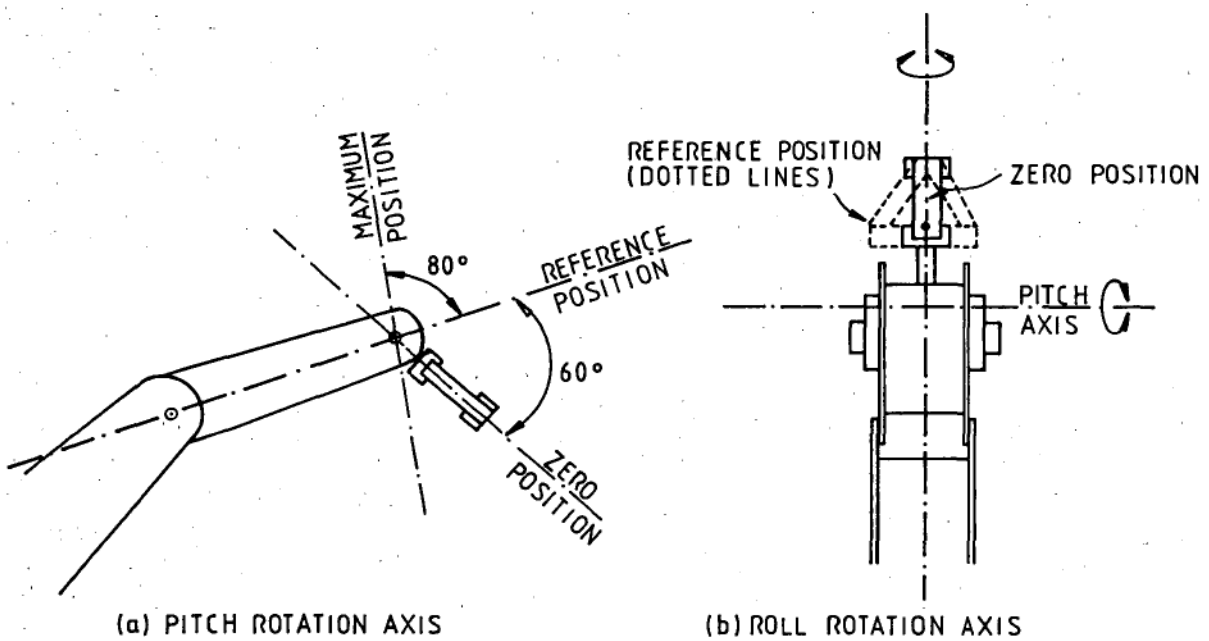
(b) The Programming Stage

The programming of the Tasrobot0 manipulator is through on-line method using a teaching arm which contains five position sensors and two switches as shown in Figure (2.4-1). During programming stage, the manipulator will follow the configuration of the teaching arm. A specific arm configuration can be recorded by pressing the MEM-switch, and the gripping function of the manipulator is controlled by the GRIP-switch.

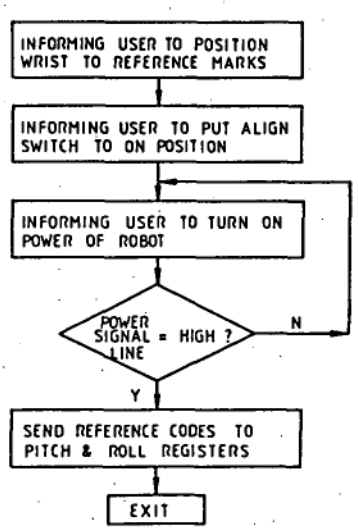
The programming stage consists of two programs, OPFILE and LEARN. The program OPFILE is to enable users to specify a global filename for storing specified data points which may be required by other programs in the other stages. Before a file with a specified filename is opened, the existence of a file with the same name will be checked to ensure that the existing file will not be overwritten leading to loss of data in that file. If a file with the same filename exists, options are provided to re-specify a filename or to allow the existing file to be overwritten.



Figure(7.2a-1): Reference Positions of The Wrist Joints



Figure(7.2a-2): Definitions of The Wrist Positions



Figure(7.2a-3): Flow-chart of The ALIGN Subroutine

	ZERO POSITION CODE	REFERENCE POSITION CODE	MAXIMUM POSITION CODE
PITCH AXIS	0	100	242
ROLL AXIS	0	150	300

Table(7.2a-1): Position Codes of The Wrist Axes

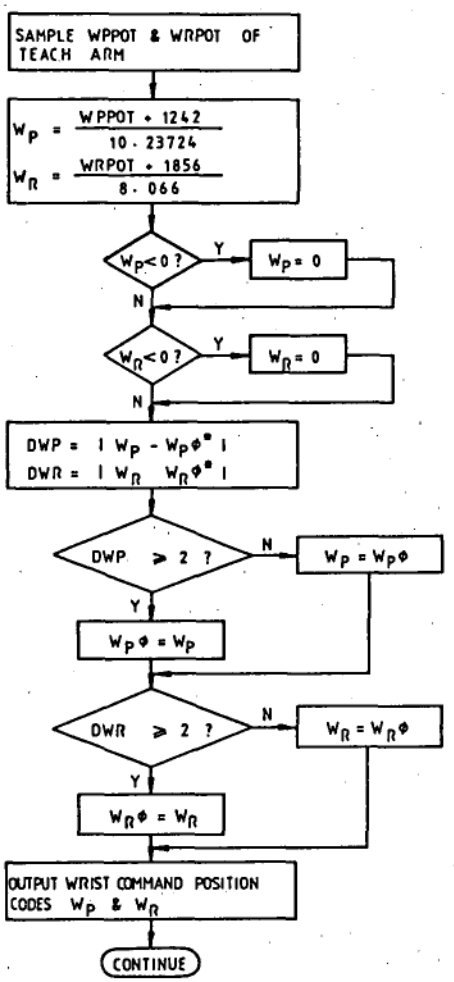
The actual programming process is implemented by the program LEARN. To track the configuration of the teaching arm, the five position sensors in the teaching arm are sampled. Since the body-, shoulder- and elbow-joint sensors in the teaching arm are the same as those in the manipulator, the sampled position codes are used as control codes to drive these three joints of the manipualtor. However, the sampled position codes from the wrist sensors in the teaching arm are different from the control codes for controlling the manipulator wrist. Therefore, transformations are required to convert the sampled position codes to the control codes for controlling the wrist motion. The transformations are:

$$WP = (WPPOT + 1242)/10.23724 \qquad (7.2b\text{-}1)$$

$$WR = (WRPOT + 1856)/8.066 \qquad (7.2b\text{-}2)$$

where WP and WR are integer constants representing the control codes for the pitch and the roll rotations respectively; and WPPOT and WRPOT are integer constants representing the sampled position codes from the pitch and the roll axes of the teaching arm. Because of sampling errors and rounding off errors, the transformed wrist control code may vary by plus or minus one unit even if the wrist positions of the teaching arm remains unchanged. This results in oscillation of the wrist rotations. The problem is solved by energizing the wrist rotations only when there are more than one unit of change in the converted control codes. It is also important that the control codes are positive to ensure correct operation of the magnitude

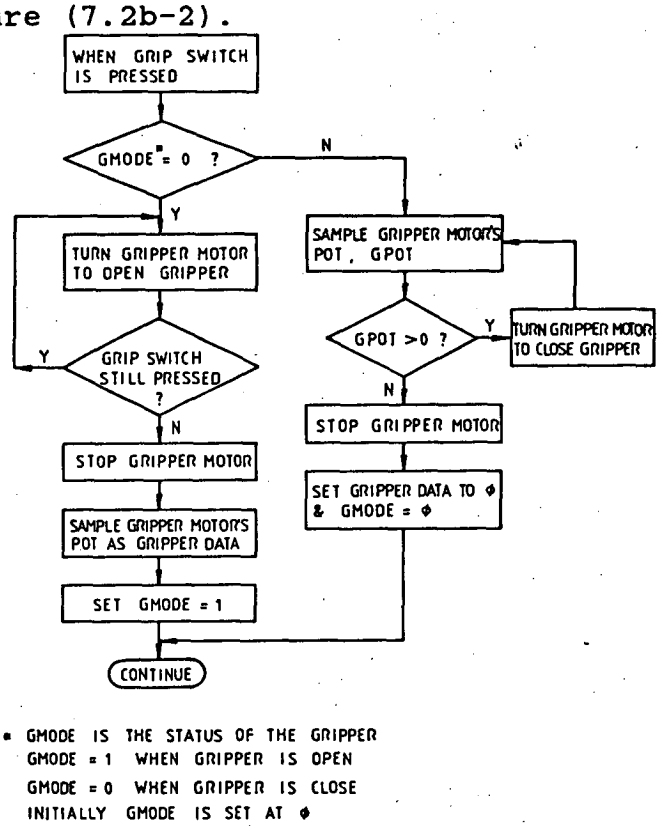
comparators used in the wrist control hardware. The algorithm of the transformation is shown in Figure (7.2b-1).



Figure(7.2b-1): Algorithm For Transforming Sampled Wrist Values To Wrist Command Codes

During the execution of the LEARN program, the status of the two switches in the teaching arm are also examined. The control codes of the five joints are recorded to the specified file when the MEM-switch is pressed. When the GRIP-switch is pressed, the state of the gripper will be checked. If the gripper is opened (or closed) the gripper motor will be energized to close (or to open) the gripper. Because of the mechanism of the gripper as discussed in section 2.1c, the amount for the gripper motor to turn to

open or close the gripper depends on the arm configuration. Therefore, during the opening of the gripper, the gripper motor will be energized as long as the GRIP-switch is pressed. The amount of opening by the gripper motor in a specific arm configuration is recorded. To close the gripper, the gripper motor is rewinded to the pre-programmed position at which the gripper is closed regardless of arm configuration. The routine when a GRIP-switch is pressed is shown in Figure (7.2b-2).



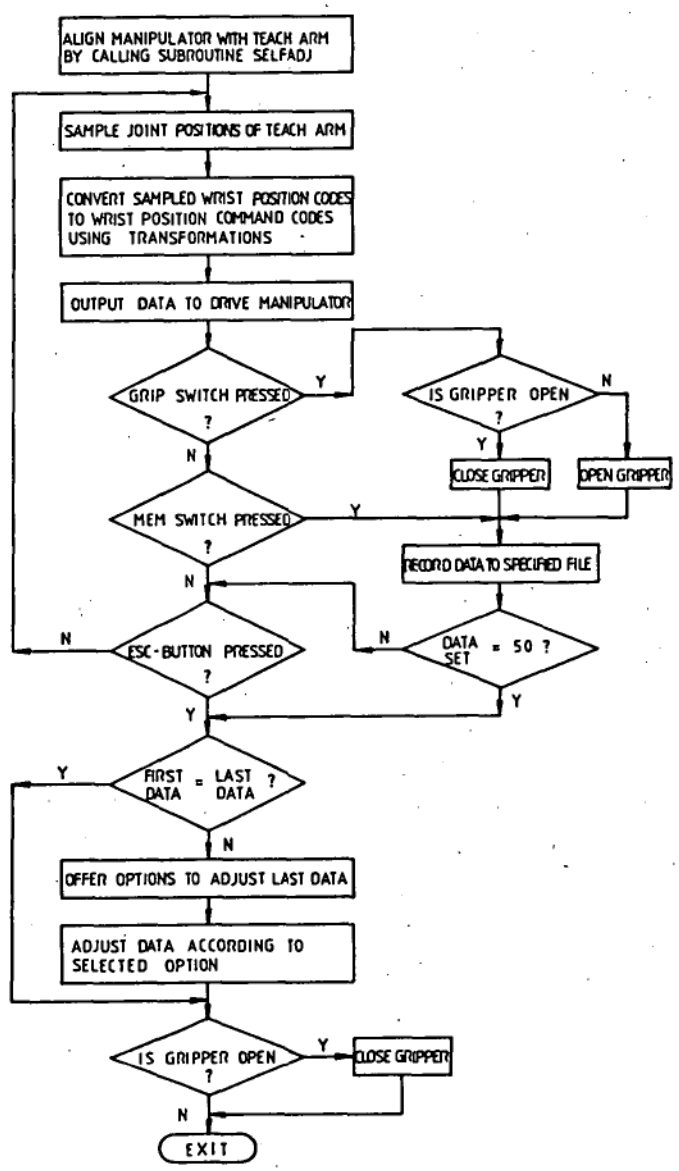
Figure(7.2b-2): Flow-chart of The GRIP-switch Subroutine

The program provides a maximum of 50 sets of data to be recorded to the specified file. Each set of data is recorded whenever the MEM-switch or the GRIP-switch is pressed and is acknowledged by a short beeping sound. The teaching process can be terminated by pressing the ESC-button on the keyboard. When 45 sets of data are recorded, the user will be warned by a beeping sound of about 5 seconds. When 50

sets of data are recorded, the user will be informed by a long beeping sound of about 10 seconds and the teaching process will be terminated. At the same time, options are provided to re-start the programming process or to carry on to the next stage.

In many robot applications, it is necessary for the manipulator to return to its starting position after it finishes a specific job so that the job can be repeated. When a teaching process is terminated, the first specified data and the last specified data are compared. If the two data are the same, the programming stage will be completed. Otherwise, options are provided to allow user to adjust the end points in four different ways. The first way simply to ignores the difference between the first and the last specified data. However, this will prevent the manipulator from repeating the job druing the executing stage. The second way is to replace the last data by the first data. This will usually be used because it is usually more difficult to bring the teach arm back to the same initial point. The third option allows the manipulator to go back to the starting position by inserting the first data set to the end of the record. The last option enable the manipulator to avoid the same obstacles when the arm is on its way back to the starting position by duplicating the same specified set of data in reverse order. This option also doubles the memory available for storing specified data. For example, if 50 sets of data are specified, the actual file will contain 99 sets of data.

Before teaching process starts, the manipulator is aligned with the teaching arm. This is carried out by the subroutine SELFADJ as described in section 7.1e. Before the LEARN program terminates, the gripper will be ensured to be closed. The overall function of the LEARN program is summarized by a flow chart shown in Figure (7.2b-3).



Figure(7.2b-3): Flow-chart of The LEARN Program

(c) The Compiling Stage

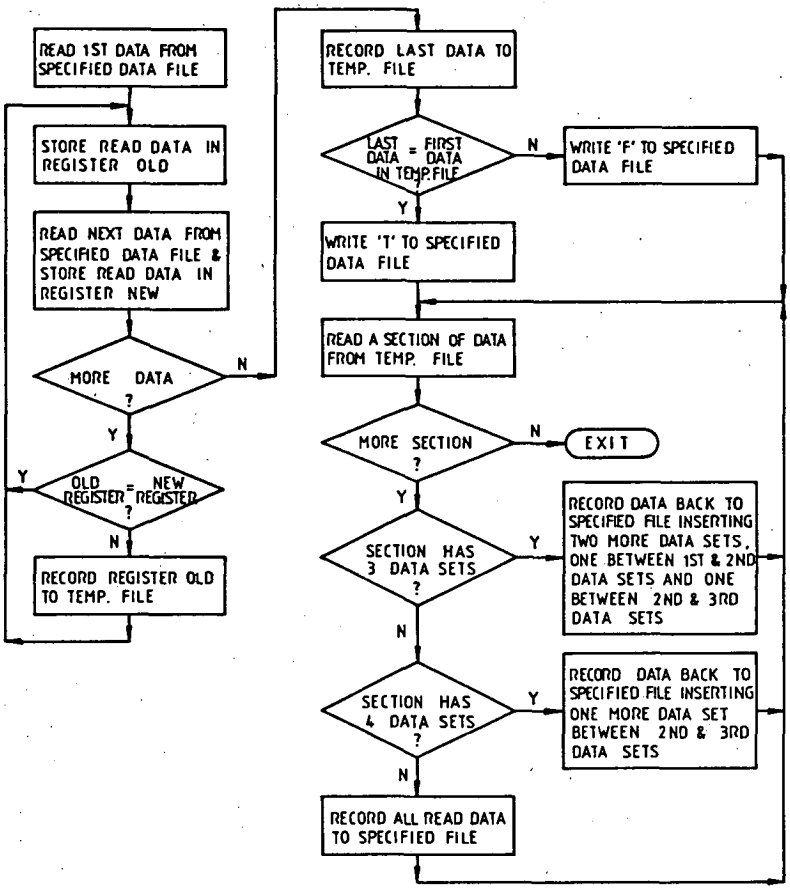
The compiling stage processes data obtained in the programming stage so that these data can be interpolated by segment functions as described in chapter six. This stage

consists two main programs, TRIM and SMOOTH. The TRIM program processes the data obtained in the programming stage to ensure conditions required in trajectory planning, as descirbed in chapter six, to be satisfied. The SMOOTH program implements the trajectory planning process.

In Tasrobot0, a typical task may be defined by a few sections. Each section consists of a number of user-defined data sets. Each date set consists of the five joint control codes and a gripper motor position code. To implement the trajectory planning technique, the data set must satisfy two conditions. First, a section must consist either two data sets or at least five data sets. Second, no two consecutive data sets can have identical control codes. The TRIM program ensures the first criterion to be achieved by inserting additional data sets to the specified data file; and satisfy the second criterion by deleting one of the data sets when two consecutive data sets are identical. The flow chart of the TRIM program is shown in Figure (7.2c-1).

Data processed by the TRIM program are recorded back to the specified file. In addition to the processed data, the recursive nature of the data, which is indicated by the identical data sets at the beginning and end of the record, will be stored at the beginning of the file in a flag. This information will pass on to the program SMOOTH to help identify a processed data file, and to generate executable data for the executing stage.

The program SMOOTH evaluates approximating functions for joining specified data points. Since a programed task



Figure(7.2c-1): Flow-chart of The TRIM Program

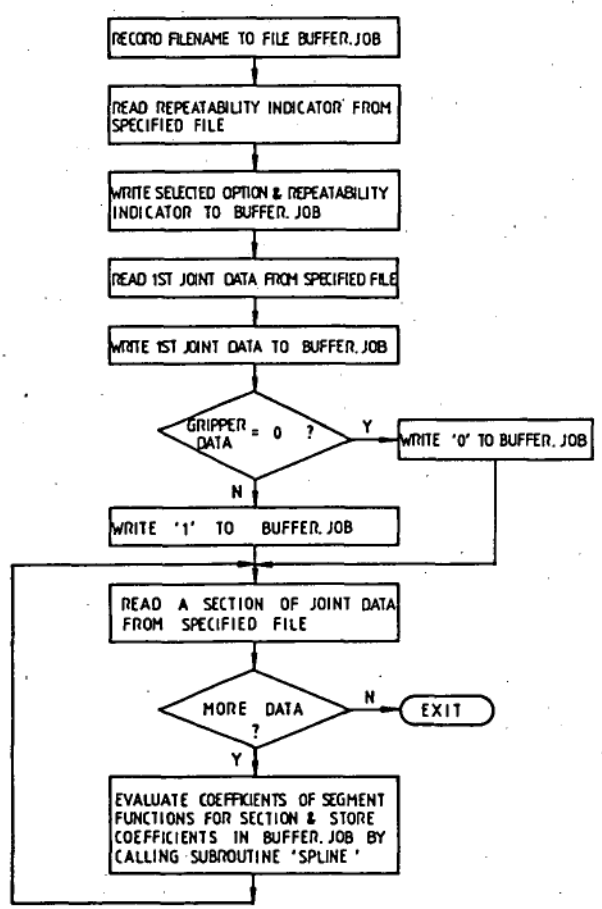
usually consists of several sections, the program identifies each sections in a task and performs trajectory planning.

The coefficients of the segment functions for each section are evaluated by the subroutine, SPLINE, of the program. The evaluated coefficients of each segment functions are stored in an executable data file called BUFFER.JOB. No specific new file is generated for storing the evaluated coefficients of the segment functions for a specific task because such a file occupies a lot more space than a file containing only joint coordinates. Also, in many robot applications, once the executable file is generated, it will be used for a period of time. The time required to generate an executable file is usually relatively short compared to the "life" time of the executable file. Therefore, it is preferrable to store the specified joint

coordinates for a specific task, although it requires replanning of a trajectory before the task can be executed.

Two options are provided in programming the orientation of the gripper during task execution. First, the wrist joints are programmed to rotate to all specified positions. However, in some robot application, such as in pick-and-place operation, the actual wrist orientation is only essential at the time when the robot starts to pick or to place an object. The second option provided will allow the manipulator to ignore the intermediate wrist positions and only wrist positions for the gripping or releasing will be implemented during the execution of a task.

The flow chart of the SMOOTH program is shown in Figure (7.2c-2).

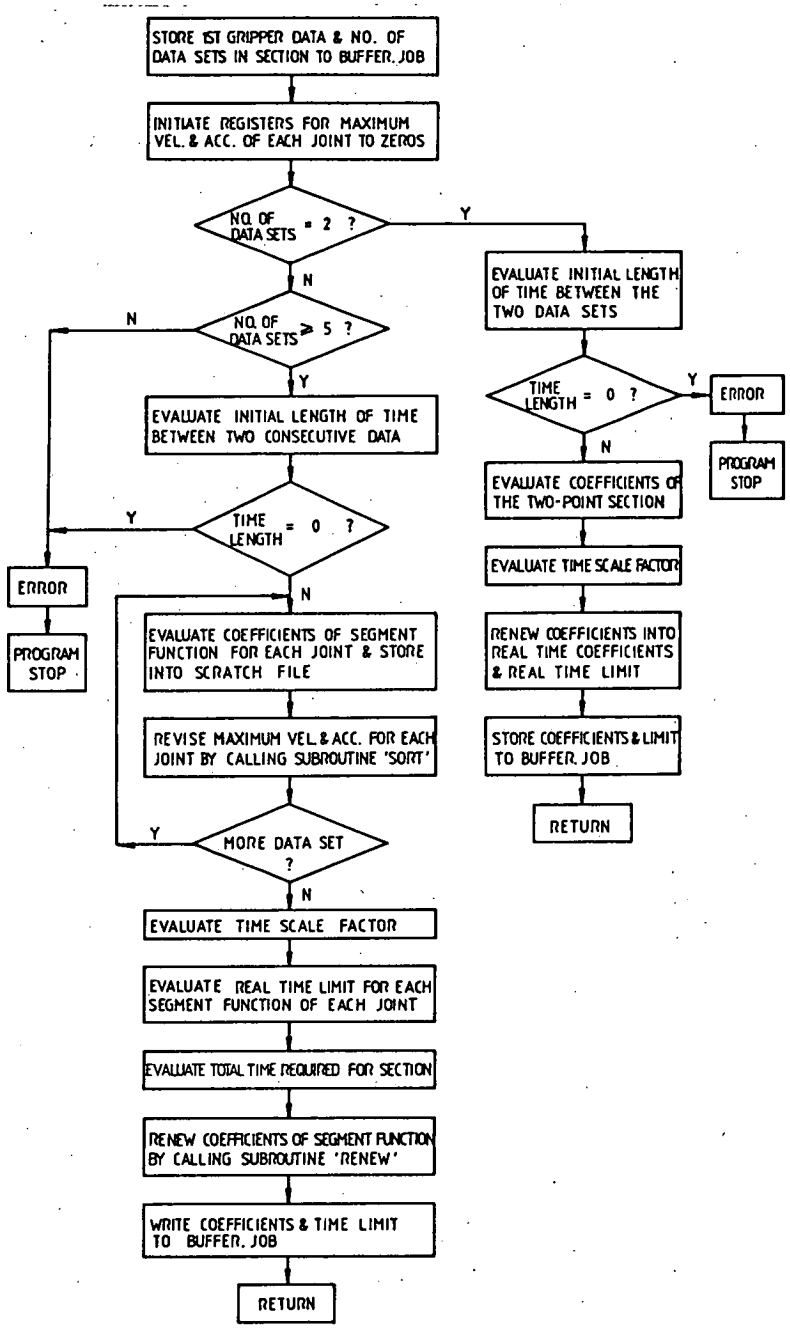


Figure(7.2c-2): Flow-chart of The SMOOTH Program

The subroutine call, SPLINE, of the SMOOTH program uses the technique discussed in Chapter 6 to evaluate the coefficients of segment functions. In this subroutine call, the number of points in a section is checked to ensure that only two-point or at least five points are in a section. For a two point section, the coefficients of the required fifth-order polynomial function are evaluated for each joint by initializing a routine. For a section with more than or equal to five defining points, another routine is initialized to evaluate the necessary coefficients of the segment functions for each joint. To reduce the dynamic memory required and hence reducing the size of the exectable program, the initial coefficients of the segment functions are temporarily stored in a scratch file. They are converted into coefficients of real time functions when the time scale factor is evaluated. The subroutine call SORT evaluates the maximum velocity and acceleration of each joint within a segment to give the time scale factor of the section. The subroutine call RENEW converts the approximating functions into real time functions using the evaluated time scale factor as discussed in section 6.3. Figure (7.2c-3) shows the flow chart of the SPLINE subroutine.

(d) The Executing Stage

The executing stage executes a programmed task through a pre-planned path. This stage consists of the main program GO. This program reads information from the exectable file - BUFFER.JOB, and generates commands, in real time, to the controller unit of the Tasrobot0 system to drive the actuators of the manipulator.



Figure(7.2c-3): Flow-chart of The SPLINE Subroutine

When the program GO is executed, it first accesses the default executable file - BUFFER.JOB, and identify the data filename from which the executable data are generated. Options are provided to allow other executable files to be specified.

Before the specified task is executed, the arm is brought to the starting position of the task by calling the

subroutine SELFADJ. If the task is a repeatable type, the user can also specify a desirable number of times of repetition.

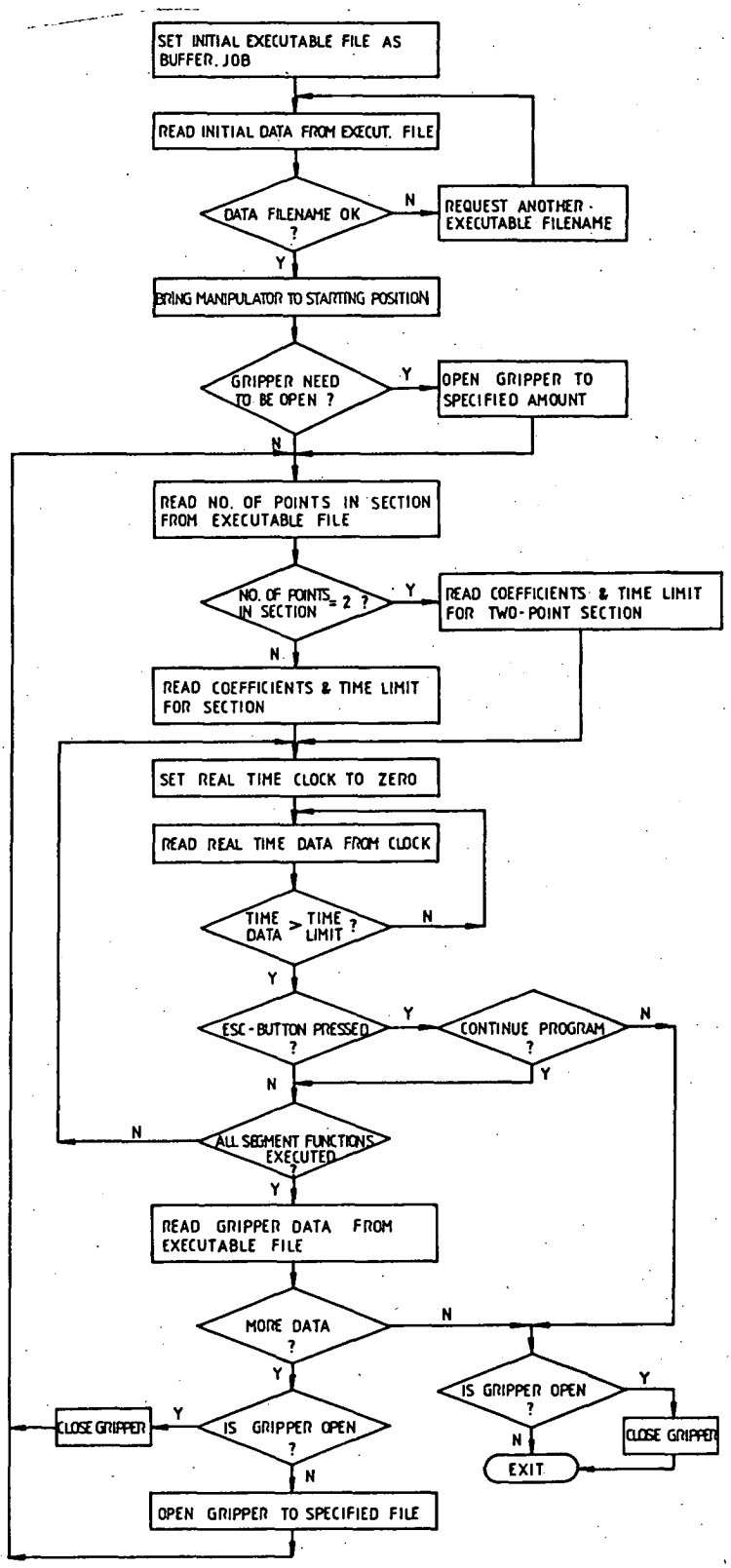
The task is executed by sending position commands to the joints of the manipulator. The commands are generated from real-time segment functions formed during the trajectory planning process. The real-time data are obtained from the real-time clock provided by the Lab Master Board and from the subroutines provided by the Lab-pac softwares. The program GO has been designed to increase as far as possible the rate of generating position commands, since the higher the rate of generating the commands, the closer the actual path to the desired path will be. During execution, the ESC-key provides emergency stop function. Before the program is terminated, the gripper will again be ensured to be closed.

The flow chart of the program is shown Figure (7.2d-1).

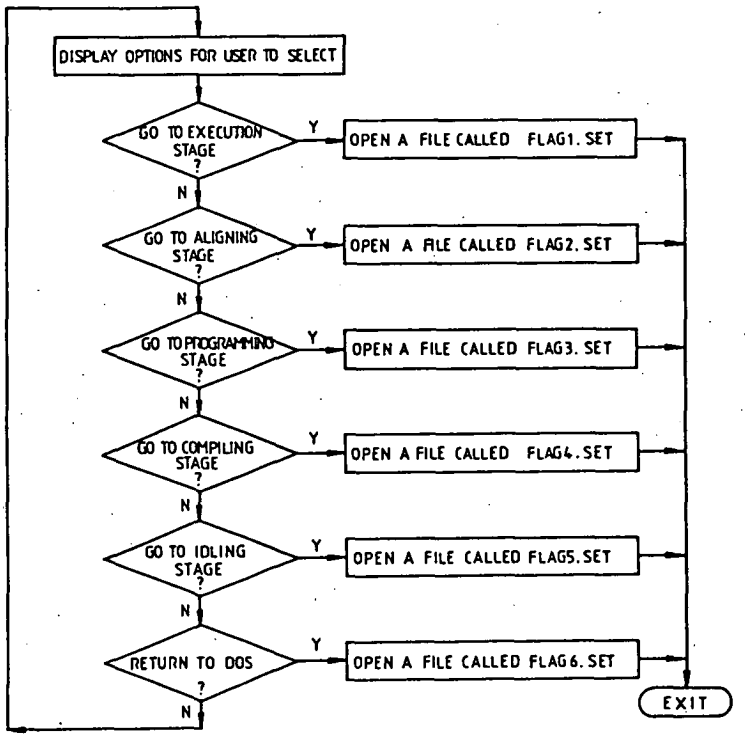
(e) The Intermediate Stage

The intermediate stage allows users to go to one of the five stages excluding the intermediate stage itself. In additions, this stage also enables users to return to the DOS system.

This stage is only required when the batch file is used to execute the eight main programs. Since batch file commands do not provide proper communication between the computer and the user, the program - SETFLAGS - is used to display a list of options for the user to select and set appropriate flag to initialize the required stage. Figure (7.2e-1) shows the flow chart of the program SETFLAGS.



Figure(7.2d-1): Flow-chart of The GO Program



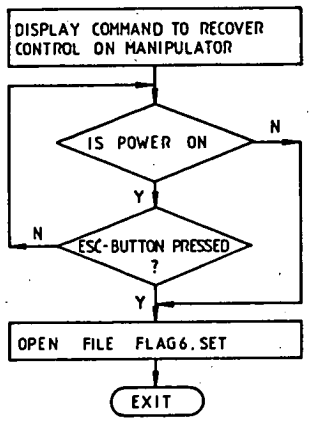
Figure(7.2e-1): Flow-chart of The SETFLAG Program

(f) The Idling Stage

The idling stage is to temporarily shut down most of the functions of the robot system. This stage may be energized when the programmed task is completed and the manipulator is not required to perform any function for a short period of time. This stage has two main functions. First, it preserves the absolute position codes for the wrist joints. Second, it retains the software control system on the computer. In other word, the computer will still be the host computer of the robot system.

The program SLEEP is used in this stage. It ensures the power supply to the electronic hardwares of the manipulator is maintained so that the absolute position codes of the wrist joints are valid. If the power supply is switched off, the software control system of Tasrobot0 will be terminated. During the idling stage, the ESC-key can be pressed to

retrieve full control of the manipulator. Figure (7.2f-1) shows the flow chart of the SLEEP program.

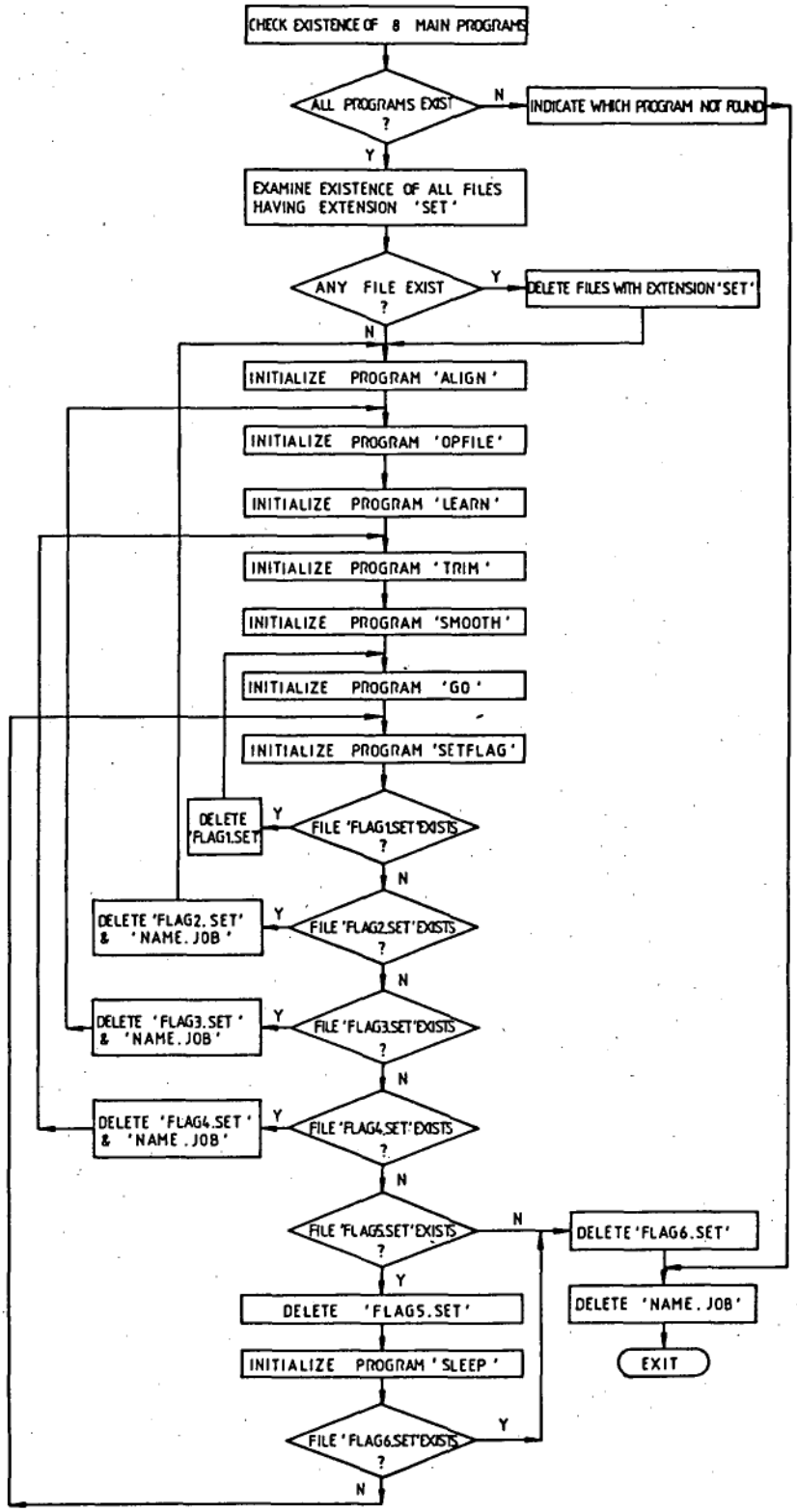


Figure(7.2f-1): Flow-chart of The SLEEP Program

7.3 THE SOFTWARE CONTROL BATCH FILE

With the use of batch subcommands and conditional execution of commands, the eight main programs can be interconnected to form the network of the software control system of the Tasrobot0 system. The software control is initialized by using the TASROBOT batch file.

Before executing the first main program, the eight main programs are examined. When all the main programs are found and all files that are used as flags in the intermediate stage are removed, the batch file will proceed. The flow chart of the batch file is shown in Figure (7.3-1).



Figure(7.3-1): Flow-chart of The TASROBOT Batch File

CHAPTER EIGHT

CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSION

Tasrobot0 is a robot manipulator system with five degrees of freedom. The roll and pitch rotations of the wrist joints are driven by stepper motors. The rotations of the body-, shoulder- and elbow-joints as well as the gripping jaws are driven by dc motors with constant field excitation. Three types of control scheme were designed and built to drive different joints of the manipulator according to the types and functions of the actuators used.

A complete robot arm system is a non-linear multi-variable time-varying system. For a small and light weight robot arm, the control system can be simplified to a combination of several individual single-input-single-output joint systems with time-varying parameters. Analogue controller were designed for the three joint systems. The effects of time-varying parameters on their closed-loop dynamic responses are significantly reduced. Their steady-state position errors also fall within one degree accuracy .

A technique on trajectroy planning was developed to generate spline segment functions which interploate between specified joint coordinates. This cubic spline trajectory planning technique was applied to the body-, shoulder- and elbow-joints of the Tasrobot0 manipulator and the performance is best illustrated by Figure (8.1-1), which

shows repeated motions of the three joints with respect to the same set of command signals. It shows that the controlled variable of each joint follows the desired trajectory and the repeatability of each controlled joint system is excellent.

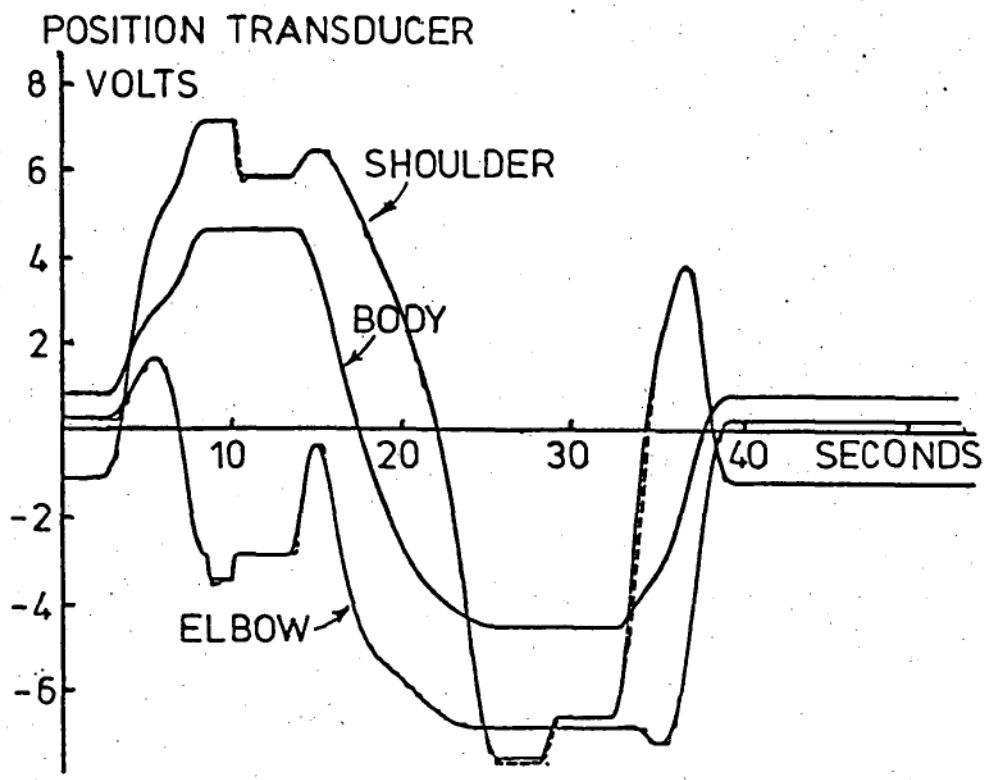


Figure (8-1): Time Responses of The Joint-system After Training

The manipulator in operation is shown in Figure (8.1-2) and the complete Tasrobot0 system is shown in Figure (8.1-3).

8.2 FUTURE WORK

Tasrobot0, which is the first version of Tasrobot manipulator, marks the beginning of research in robotics in the Electrical Engineering Department of University of Tasmania. Based on the study of Tasrobot0, there are a

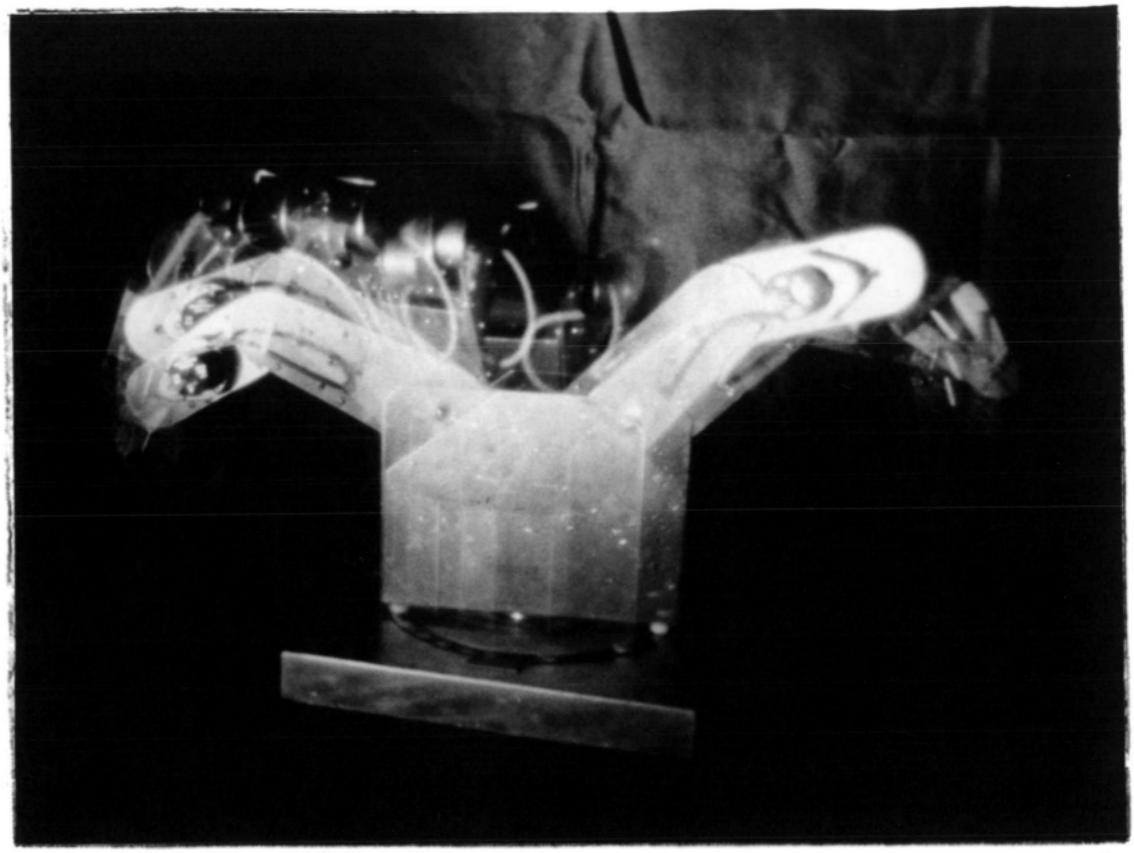


FIGURE (8.1-2) : The Tasrobot0 Manipulator In Operation

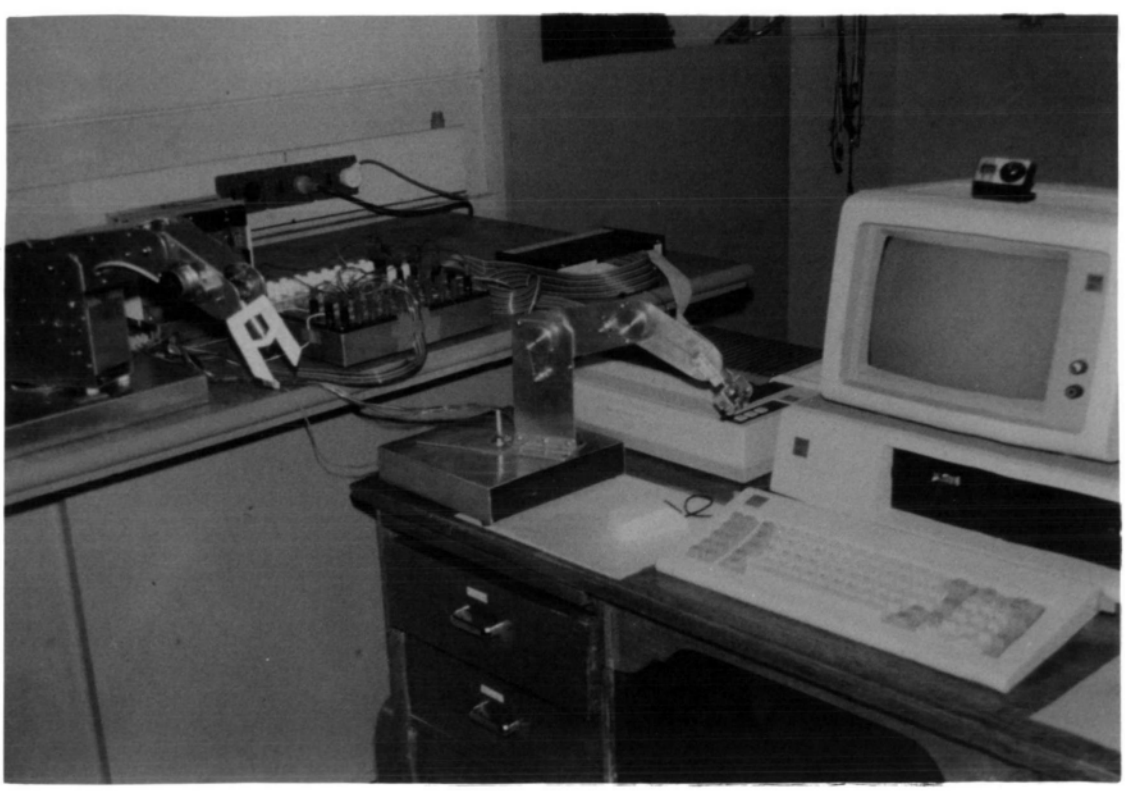


FIGURE (8.1-3) : The Tasrobot0 System, From Left To Right : Manipulator Controller Unit, Teach Arm And Host Computer

number of ideas and suggestions to future development of Tasrobot arms and further investigation in robotics.

(a) On Manipulator Structure

As the Tasrobot0 manipulator is still in the developing stage, the sophistication of mechanical structure of Tasrobot0 has not been emphasized. There are a number of suggestions to the next version of the arm:

The physical size of the arm should be increased to extend the accessible distance of the arm.

The elbow joint should have larger range of motion to increase the work envelope of the arm.

The wrist motion should be redesigned to replace stepper motors by dc servo-motors and to provide yaw-rotation.

The gripper motor should be located as close to the gripper as possible to eliminate the dependence of the effective length of the connected steel string on arm configuration.

(b) On Controller Design

As the level of sophistication of Tasrobot arms improves, more complex controllers will be required. There are a number of control methods to be investigated, such as decentralized control suggested by Prof.M.Vukobrotovic, PID control, computed torque methods, resoled rate and acceleration method, as well as adaptive control technique. Prof.M.vukobratovic pointed out: 'the complexity of the controller would depend on the level of sophistication and autonomy of the robot under consideration'. A suitable

controller should be selected according to the mechanical sophistication of the arm.

(c) On Trajectory Planning

Command signals are generated from real-time segment functions by the host computer. During path execution, the host computer can afford no time to perform other functions. Since the generation of command signals involves only additions and multiplications which are easier and faster to do by pre-wired hardwares, numeric processor or microprocessor. If a microprocessor unit is used as a command generator, the host computer will only need to output the coefficients of the segment functions. The host computer can then spare to monitor other sensor signals like force sensor signal or vision sensor signal during path execution. This will certain improve the control of the Tasrobot arm and widen the scope of the Tasrobot system.

As Prof.Saridis has pointed out: 'the manipulator control problem has not been successfully resolved', solution to control problem on robotics will rely on further research in this area.

REFERENCES

[1] Paul,R.P.:'Robot Manipulators, Mathematics, Programming and Control', MIT press, Cambridge, Mass., 1981.

[2] Wilson,D.R.:'Modern Practice in Servo Design', Pergamon Press, 1970.

[3] Taylor,P.L.:'Servomechanisms', Longmans, 1960.

[4] Shinners,S.M.:'Modern Control System Theory And Application', Addison-Wesley, 1972.

[5] Vukobratovic,M. and Potkonjak,V.:'Applied Dynamics and CAD of Manipulation Robots', Springer-Verlag, 1985.

[6] Ahlberg,J.H., Nilson,E.N. and Walsh,J.L.:'The Theory of Splines and Their Applications', Academic Press, 1967.

[7] Vukobratovic,M. and Kircanski,M.:'Kinematics and Trajectory Synthesis of Manipulation Robots', Springer-Verlag, 1986.

[8] Ranky,P.G. and Ho,C.Y.:'Robot Modelling, Control and Applications With Software', Springer-Verlag, 1985.

[9] Pugh,A.:'Robotic Technology', Peter Peregrinus, 1983.

[10] Koren,Y.:'Robotics For Engineers', McGraw-Hill, 1985.

[11] Coughlin,R.F. and Driscoll,F.F.:'Operational Amplifiers and Linear Integrated Circuits', Prentice-Hall, 1982.

[12] Rehg,J.A.:'Introduction to Robotics', Prentic-Hall 1985.

[13] Aleksander,I.:'Computing Techniques For Robots', Kogan Page, 1985.

[14] Malcolm,D.R.:'Robotics, An Introduction', Breton, 1985.

[15] Cumming,I.G.:'Autocorrelation Function and Spectrum of a Filtered Pseudorandom Binary Sequence', Proc. IEE, vol.114, no.9, September 1967, pp1360-1362.

[16] Karmer,C.:'A Low-Frequency Pseudo-Random Noise Generator', Electronic Engineering, July,1965, pp465-467.

[17] Robert,P.D. and Davis,R.H.:'Statistical Properties of Smoothed Maximal-length Linear Binary Sequences', Proc. IEE, vol.113, no.1, January 1966, pp190-196.

[18] Hazlerigg,A.D.G. and Noton,A.R.M.:'Application of Crosscorrelating Equipment to Linear-system Identification', Proc. IEE, vol.112, no.12,December 1965, pp2385-2400.

[19] Rowe,I.H. and Kerr,I.M.:'A Board-Spectrum Psudorandom Gaussian Noise Generator', IEEE Transactions on Automatic Control, vol.AC-15, no.5, October 1970, pp529-534.

[20] Coffron,J.W.:'Programming The 8086/8088' Sybex, 1983.

[21] Kalaba,R. and Spingarn,K.:'Control, Identification, and Input Optimization', Plenum Press, 1982.

[22] Sage,A.P. and Melsa,J.L.:'System Identification', Academic Press, 1971.

[23] Isermann,R.:'Digital Control Systems', Springer-Verlag, 1981.

[24] Mendel,J.M.:'Discrete Techniques of Parameter Estimation', Marcel Dekker, 1973.

[25] Eykhoff,P.:'System Identification, Parameter and State Estimation', John Wiley & Sons, 1974.

[26] Goodwin,G.C. and Payne,R.L.:'Dynamic System Identification, Experiment Design and Data Analysis', Academic Press, 1977.

[27] Sheingold,D.H.:'Analog-Digital Conversion Handbook', Prentice-Hall, 1986.

[28] Loriferne,B.:'Analog-Digital and Digital-Analog Conversion', Heyden & Son, 1982.

[29] Hildebrand,F.B,:'Introduction to Numerical Analysis', McGraw-Hill, 1974, Chapter 10.

[30] 'IBM Technical Reference', IBM, 1984.

[31] 'TUTSIM User's Manual For IBM/PC Computers', Meerman, Automation, 1983.

[32] 'Lab Pac User Guide', Tecmar, 1984.

[33] 'Lab Master Installation Manual and User's Guide', Scientific Solutions, 1985.

[34] 'Micro-soft Fortran Compiler User's Guide', Microsoft, 1984, Chapter 9.

[35] Snyder,W.E.:'Industrial Robots:Computer Interface And Control', Prentice-Hall, 1986.

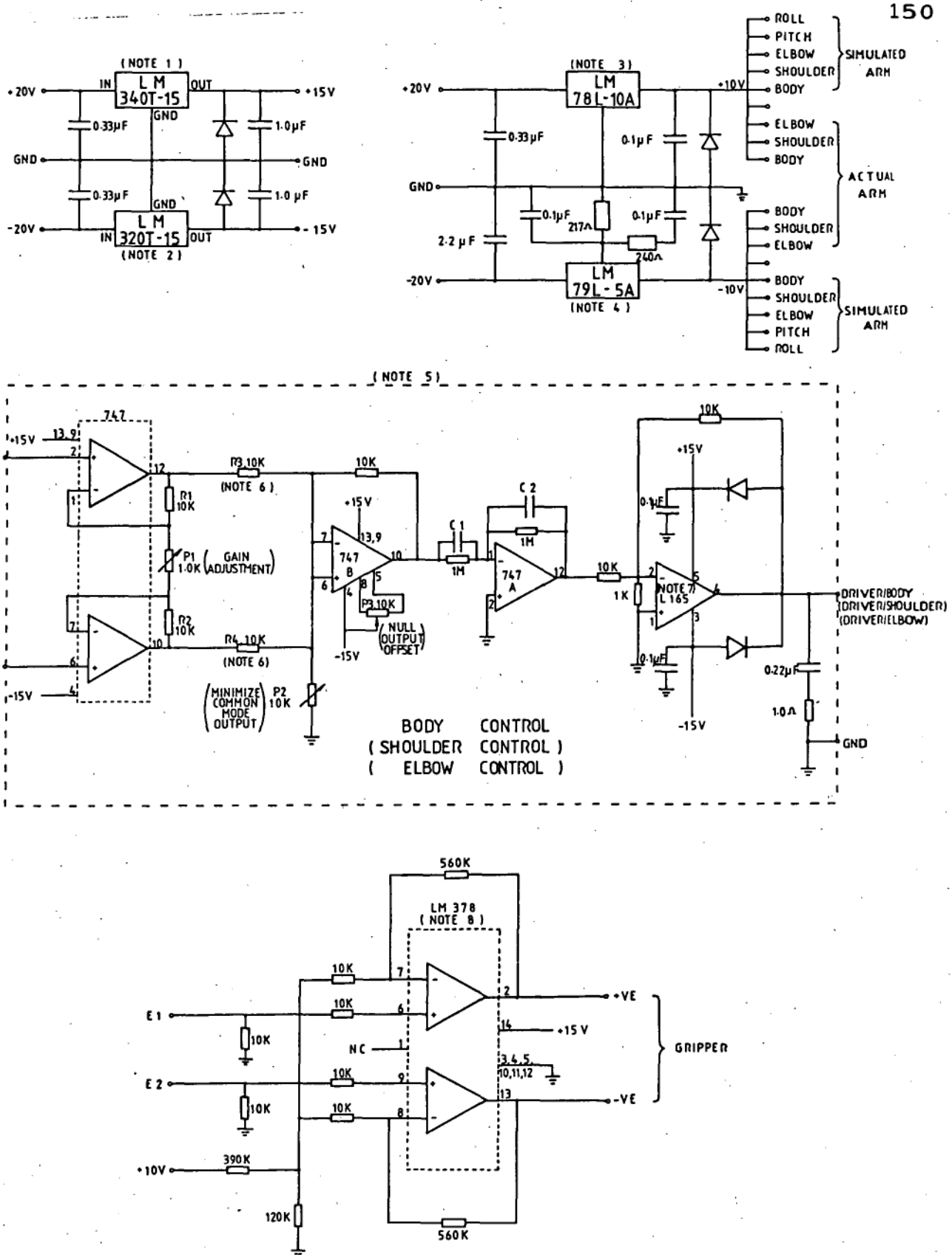
[36] Hollerbach,J.M.:'A Recursive Lagrangian Formulation of Manipulator Dynamics and a Cooperative Study of Dynamics Formulation Complexity', IEEE Trans. On Systems, Man and Cybernetics, vol.10, no.11, November 1980, pp730-736.

[37] Luh,J.Y.S, Walker,M.W. and Paul,R.P.C.:' On-line Computational Scheme For Mechanical Manipulators', Journal of Dynamic Systems, Measurement, and Control, Tran. of ASME, vol.102, no.2, June 1980, pp69-76.

[38] The,G. and Lam,R.:'Identification and Control System Design For A Robot Manipulator', 15th IASTED International Conference On Applied Simulation and Modelling, 1987.

APPENDIX A

CIRCUIT DIAGRAMS OF THE CONTROLLER UNIT



NOTE: (1) MAX CURRENT DRAWN = 1.5 A
(2) MAX CURRENT DRAWN = 1.5 A
(3) MAX CURRENT DRAWN = 10 mA
(4) MAX CURRENT DRAWN = 10 mA
(5) TWO SIMULAR CIRCUITS FOR CONTROLLING SHOULDER & ELBOW
(6) VALUE OF R3 & R4 TO BE MADE AS CLOSE AS POSSIBLE
(7) EACH REQUIRED HEAT SINK OF -
(8) MOUNTED ON V7 HEAT SINK

Figure (A-1): Schematic Circuit Diagram of The Analog Control Board

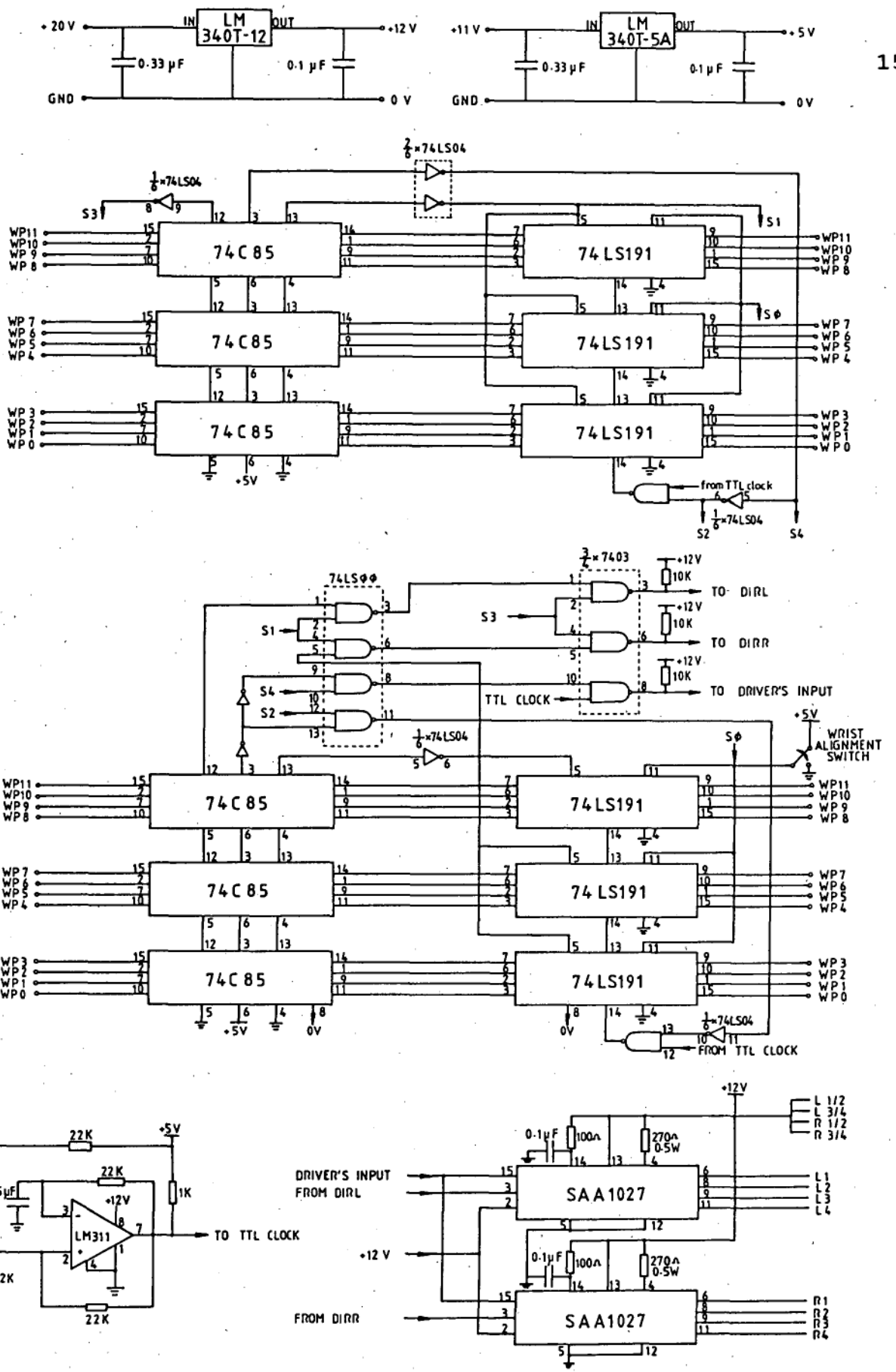
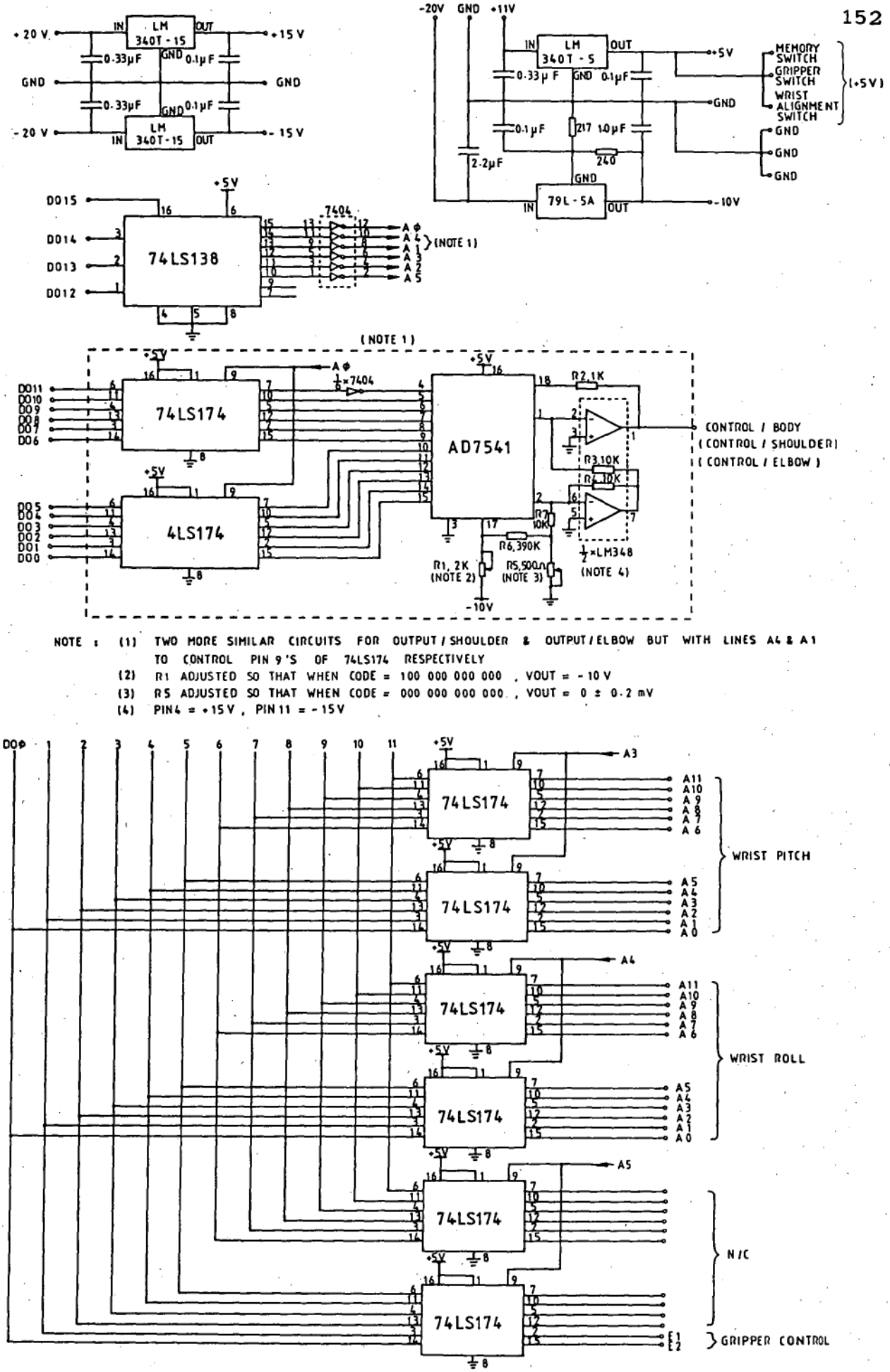


Figure (A-2): Schematic Circuit Diagram of The Digital Control Board



NOTE : (1) TWO MORE SIMILAR CIRCUITS FOR OUTPUT / SHOULDER & OUTPUT / ELBOW BUT WITH LINES A4 & A1 TO CONTROL PIN 9 'S OF 74LS174 RESPECTIVELY

(2) R1 ADJUSTED SO THAT WHEN CODE = 100 000 000 000 , VOUT = - 10 V

(3) R5 ADJUSTED SO THAT WHEN CODE = 000 000 000 000 , VOUT = 0 ± 0.2 mV

(4) PIN4 = +15 V , PIN 11 = - 15 V

Figure (A-3): Schematic Circuit Diagram of The Interface Circuit Board

APPENDIX B

SOFTWARE CONTROL PROGRAMS

```
;****************************
; PUBLIC ASSEMBLY SUBROUTINES
;****************************

DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  STPORT
STPORT  PROC    FAR
        PUSH    BP
        MOV     BP,SP
        MOV     DX,071FH
        MOV     AL,090H
        OUT     DX,AL
        MOV     SP,BP
        POP     BP
        RET
STPORT  ENDP
CODE    ENDS
END

DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  DIGOUT
DIGOUT  PROC    FAR
        PUSH    BP
        MOV     BP,SP
        LES     BX,DWORD PTR [BP+10]
        MOV     AX,ES:[BX]
        OR      AX,0F000H
        LES     BX,DWORD PTR [BP+6]
        MOV     DX,ES:[BX]
        MOV     CL,4
        ROR     DX,CL
        OR      DX,00FFFH
        AND     AX,DX
        AND     AX,07FFFH
        MOV     DX,071DH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     SP,BP
        POP     BP
        RET     08H
DIGOUT  ENDP
CODE    ENDS
END
```



```
DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  POSITN
POSITN  PROC    FAR
        PUSH    BP
        MOV     BP,SP
        LES     BX,DWORD PTR [BP+6]
        MOV     DX,071DH
        MOV     AX,ES:[BX]
        OR      AX,0F000H
        AND     AX,00FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+2]
        OR      AX,0F000H
        AND     AX,02FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+4]
        OR      AX,0F000H
        AND     AX,04FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     SP,BP
        POP     BP
        RET     04H
POSITN  ENDP
CODE    ENDS
END

DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  WRIST
WRIST   PROC    FAR
        PUSH    BP
        MOV     BP,SP
        LES     BX,DWORD PTR [BP+6]
        MOV     DX,071DH
        MOV     AX,ES:[BX]
        OR      AX,0F000H
```

```
        AND     AX,03FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+2]
        OR      AX,0F000H
        AND     AX,01FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     SP,BP
        POP     BP
        RET     04H
WRIST   ENDP
CODE    ENDS
END

DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  ARMOUT
ARMOUT  PROC    FAR
        PUSH    BP
        MOV     BP,SP
        LES     BX,DWORD PTR [BP+6]
        MOV     DX,071DH
        MOV     AX,ES:[BX]
        OR      AX,0F000H
        AND     AX,00FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+2]
        OR      AX,0F000H
        AND     AX,02FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+4]
        OR      AX,0F000H
        AND     AX,04FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
```



```
        OUT     DX,AX
        MOV     AX,ES:[BX+6]
        OR      AX,0F000H
        AND     AX,03FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     AX,ES:[BX+8]
        OR      AX,0F000H
        AND     AX,01FFFH
        OUT     DX,AX
        OR      AX,08000H
        OUT     DX,AX
        AND     AX,07FFFH
        OUT     DX,AX
        MOV     SP,BP
        POP     BP
        RET     04H
ARMOUT  ENDP
CODE    ENDS
END

DATA    SEGMENT PUBLIC 'DATA'
DATA    ENDS
DGROUP  GROUP   DATA
CODE    SEGMENT 'CODE'
        ASSUME CS:CODE,DS:DGROUP,SS:DGROUP
PUBLIC  BEEP
BEEP    PROC    FAR
        PUSH    BP
        MOV     BP,SP
        LES     BX,DWORD PTR [BP+6]
        MOV     AX,ES:[BX]
        MOV     BX,AX
        MOV     AL,10110110B    ;SET TIMER-2,LBS,MBS BINARY
        OUT     43H,AL          ;WRITE THE TIMER MODE REG.
        MOV     AX,533H         ;DIVISOR FOR 1000 HZ
        OUT     42H,AL          ;WRITE TIMER-2 CNT-LSB
        MOV     AL,AH
        OUT     42H,AL          ;WRITE TIMER-2 CNT-MSB
        IN      AL,61H          ;GET CURRENT SETTING OF PORT B
        MOV     AH,AL           ;SAVE THAT SETTING
        OR      AL,03           ;TURN SPEAKER ON
        OUT     61H,AL
        MOV     CX,01999H       ;SET CNT TO WAIT 50MS
L1:     LOOP    L1              ;DELAY BEFORE TURNING OFF
        DEC     BX              ;DELAY CNT EXPIRE?
        JNZ     L1              ;NO-CONTINUE BEEPING SPEAKER
        MOV     AL,AH           ;RECOVER VALUE OF PORT B
        OUT     61H,AL
        MOV     SP,BP
        POP     BP
        RET     04H
```

```
BEEP    ENDP
CODE    ENDS
END
```



```
C     ************************
C     PUBLIC FORTRAN SUBROUTINE
C     ************************

$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE SELFADJ(F2)

      INTEGER F1(6),F2(5),CHNRAY(7)
      DIMENSION F(3),VCMX(3),VCMN(3),ACMX(3),ACMN(3)
      DIMENSION B1(3),B4(3),B5(3),B6(3)

C     SETTING UP CHANNEL POSITIONS.
      CHNRAY(1)=6
      CHNRAY(2)=7
      CHNRAY(3)=8
      CHNRAY(4)=3
      CHNRAY(5)=4
      CHNRAY(6)=5
      CHNRAY(7)=999

C     RECORDING PRESENT ARM POSITION BY SAMPLING.
      CALL INTRON
      CALL ADSWST(CHNRAY,1,F1,6,1)
      CALL INTROFF

C     CLOSING THE GRIPPER IF IT IS OPEN.
      IF(F1(6).NE.0)THEN
         CALL DIGOUT(2,5)
 50      CALL ADIN(5,F1(6))
         IF(F1(6).GT.0)GOTO 50
         CALL DIGOUT(0,5)
      ENDIF


C     CONVERTING WRIST PITCH & ROLL VALUES TO NO. OF STEPS.
      F1(4)=NINT((F1(4)+1242)/10.23724)
      IF(F1(4).LT.0)F1(4)=0
      F1(5)=NINT((F1(5)+1856)/8.06333)
      IF(F1(5).LT.0)F1(5)=0

C     COMPARING POSITION DATA TO SEE IF DATA ARE MATCHED.
      DO 60 J=1,3
         IF(F1(J).NE.F2(J))GOTO 70
 60   CONTINUE

C     OUTPUTING WRIST ORIENTATION IF POSITION DATA MATCHED.
      CALL WRIST(F2(4))

      RETURN

C     SETTINT 6TH ORDER SPLINE COEFFICIENTS TO JOINT THE 2 POINTS.
C     IF POSITION DATA DO NOT MATCH.
 70   DO 75 J=1,3
```

```
      F(J)=REAL(F2(J)-F1(J))
75    CONTINUE
      T=SQRT(F(1)**2+F(2)**2+F(3)**2)
      VCMX(1)=574
      VCMN(1)=-574
      VCMX(2)=1140
      VCMN(2)=-3280
      VCMX(3)=1200
      VCMN(3)=-1800
      ACMX(1)=2332
      ACMN(1)=-2332
      ACMX(2)=8497
      ACMN(2)=-11015
      ACMX(3)=7596
      ACMN(3)=-7452
      S=0
      TVM=T/2
      TAM=0.788675*T
      DO 80 J=1,3
         B1(J)=FLOAT(F1(J))
         B4(J)=10*F(J)/(T**3)
         B5(J)=-15*F(J)/(T**4)
         B6(J)=6*F(J)/(T**5)
         VMAX=3*B4(J)*(TVM**2)+4*B5(J)*(TVM**3)+5*B6(J)*(TVM**4)
         AMAX=6*B4(J)*TAM+12*B5(J)*(TAM**2)+20*B6(J)*(TAM**3)
         IF(VMAX.GT.0)THEN
            SV=VMAX/VCMX(J)
         ELSE
            SV=VMAX/VCMN(J)
         ENDIF
         IF(AMAX.GT.0)THEN
            SA=SQRT(AMAX/ACMX(J))
         ELSE
            SA=SQRT(AMAX/ACMN(J))
         ENDIF
         ST=ABS((REAL(F2(4)+F2(5)-F1(4)-F1(5)))/(20*T))
         S=MAX(SV,SA,ST,S)
80    CONTINUE
      S=1000*S
      TOUT=T*S
      DO 90 J=1,3
         B4(J)=B4(J)/(S**3)
         B5(J)=B5(J)/(S**4)
         B6(J)=B6(J)/(S**5)
90    CONTINUE

C     OUTPUTING SPLINE FUNCTION IN REAL TIME.
      CALL WRIST(F2(4))
      IF(TOUT.GT.30000.0)GOTO 120
      CALL TIMST(0)
100   CALL TIMRD(0,ITMSC)
      T=FLOAT(ITMSC)
      IF(T.GT.TOUT)GOTO 200
      T3=T**3
      T4=T3*T
```



```
      T5=T4*T
      DO 110 J=1,3
         F2(J)=NINT(B1(J)+B4(J)*T3+B5(J)*T4+B6(J)*T5)
110   CONTINUE
      CALL POSITN(F2(1))
      GOTO 100

120   CALL TIMST(0)
130   CALL TIMRR(0,TMSEC)
      CALL M2ISQQ(TMSEC,T)
      IF(T.GT.TOUT)GOTO 200
      T3=T**3
      T4=T3*T
      T5=T4*T
      DO 140 J=1,3
         F2(J)=NINT(B1(J)+B4(J)*T3+B5(J)*T4+B6(J)*T5)
140   CONTINUE
      CALL POSITN(F2(1))
      GOTO 130

200   RETURN

      END
```

```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM ALIGN

      IMPLICIT INTEGER(A-Z)

C     INITIALIZING THE LAB-PACK SUBROUTINES.
      CALL INIT
      CALL INTROFF
      CALL STPORT

      WRITE(*,50)
 50   FORMAT(/////,10X,'PLEASE FOLLOW THE INSTRUCTIONS EXACTLY, AND',
     *//,6X,'BE SURE EACH STEP DONE BEFORE RETURN-BUTTON PRESSED.',
     */////)

      PAUSE ' BRING THE GRIPPERS TO THEIR REFERENCE POSITIONS.'
      WRITE(*,'(///)')

      PAUSE ' SWITCH THE ALIGN SWITCH TO ON POSITION.'
      WRITE(*,'(///)')

      PAUSE ' SWITCH ON THE POWER SUPPLY OF THE ARMS.'
      WRITE(*,'(///)')

      CALL DISTAT(0,7,POWER,0)
      IF(POWER.EQ.0)THEN
         WRITE(*,60)
 60      FORMAT(//,' NO POWER IS DETECTED.',
     *   //,' PLEASE CHECK POWER SUPPLY.',//)
 65      CALL DISTAT(0,7,POWER,0)
         IF(POWER.EQ.0)GOTO 65
      ENDIF
      CALL DIGOUT(100,3)
      CALL DIGOUT(150,1)

      PAUSE ' SWITCH THE ALIGN SWITCH BACK TO OPERATE POSITION.'
      WRITE(*,'(///)')

      WRITE(*,70)
 70   FORMAT(///////,16X,' ALIGNMENT PROCESS FINISHED.',///////)

      END
```



```
$STORAGE:2
      PROGRAM OPFILE
      CHARACTER FNAME*11,KEY*1
      LOGICAL OLDFILE
 50   WRITE(*,60)
 60   FORMAT(///,' PLEASE INPUT A FILE NAME, XXX.DAT, WHICH DATA',
     */,' ARE TO BE STORED & MANIPULATED.',
     *//,' [NOTE: XXX MUST BE LESS THAN 7 CHARACTER.]',/)
      READ(*,'(A)')FNAME
      INQUIRE(FILE=FNAME,EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 80
      WRITE(*,70)
 70   FORMAT(//,' WARNING: FILE WITH THE SAME NAME ALREADY EXISTS.',
     */,' DO YOU WANT TO SPECIFY ANOTHER NAME? (Y/N) ')
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')GOTO 50
 80   OPEN(5,FILE='NAME.JOB',STATUS='NEW')
      WRITE(5,'(A)')FNAME
      CLOSE(5)
      END
```

```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM LEARN

      IMPLICIT INTEGER (A-Z)
      DIMENSION DATRAY(5),JOINT(6,50),CHRAY1(6)
      CHARACTER FNAME*11,KEY*1
      LOGICAL OLDFILE

C     INITIALIZING THE LABPACK ROUTINES
      CALL INIT
      CALL INTROFF
      CALL STPORT
      CALL STTIMEB(2000)'

C     SETTING CHANNEL NOS.
      CHRAY1(1)=0
      CHRAY1(2)=1
      CHRAY1(3)=2
      CHRAY1(4)=3
      CHRAY1(5)=4
      CHRAY1(6)=999
      DATA5=0
      DATA4=0

C     TRYING TO READ FILE NAME.JOB
      INQUIRE(FILE='NAME.JOB',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 30
      OPEN(50,FILE='NAME.JOB')
      READ(50,'(A)',ERR=30)FNAME
      GOTO 41

C     REQUESTING FILENAME TO BE USED FOR STORING DATA
 30   WRITE(*,35)
 35   FORMAT(' PLEASE INPUT FILENAME, XXX.DAT, TO WHICH ',
     */,' JOINT DATA ARE TO BE STORED.',
     *//,' [NOTE:  XXX MUST BE LESS THAN 7 CHARACTERS.]',/)
      READ(*,'(A)')FNAME
      WRITE(*,'(///)')

C     CHECKING INPUT FILE IF ALREADY EXIST.
      INQUIRE(FILE=FNAME,EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 41
      WRITE(*,39)
 39   FORMAT(//,' WARNING: FILE WITH THE SAME NAME ALREADY EXISTS.',
     *//,' DO YOU WANT TO SPECIFY ANOTHER NAME? (Y/N).')
      READ(*,'(A)')KEY
      WRITE(*,'(/////)')
      IF(KEY.NE.'N')GOTO 30

C     ALIGNING THE ACTUAL ARM WITH RESPECT TO THE SIMULATED ARM.
 41   CALL INTRON
      CALL ADSWST(CHRAY1,1,DATRAY,5,1)
      CALL INTROFF
```



```
      DATRAY(4)=NINT((DATRAY(4)+1242)/10.23724)
      IF(DATRAY(4).LT.0)DATRAY(4)=0
      DATRAY(5)=NINT((DATRAY(5)+1856)/8.06333)
      IF(DATRAY(5).LT.0)DATRAY(5)=0

      CALL SELFADJ(DATRAY)

C     OPENING A FILE FOR STORING DATA TO THE SPECIFIED FILENAME
 50   OPEN(5,FILE=FNAME,STATUS='NEW')

C     INFORMING USER WHILE THE SYSTEM IS READY AFTER INITIALIZATION
 55   PAUSE ' THE ROBOT IS READY TO BE TAUGHT.'

      WRITE(*,57)
 57   FORMAT(//////////,17X,' ROBOT IS NOW BEING TAUGHT.',
     *//,7X,' PRESS ESC-BUTTON WHEN FINISH TEACHING PROCESS.',
     */////////////)

C     INITIALIZING THE FLAGS REQUIRED
      M=1
      GRPMOD=0
 60   MEMODE=0

C     DETECTING WHETHER ESC BUTTON IS PRESSED FOR TERMINATION
 70   CALL ESC(PRESSED)
      IF(PRESSED.EQ.1)GOTO 140

C     SAMPLING 5 A/D CHANNELS WITH SAMPLING RATE 1000 SAMPLES/SEC..
      CALL INTRON
      CALL ADSWST(CHRAY1,1,DATRAY,5,1)
      CALL INTROFF

C     CONVERTING WRIST PITCH & ROLL VALUES TO NO. OF STEPS
C     OUTPUTING ONLY IN THE INCREMENT OF 2 STEPS.
      DATRAY(4)=NINT((DATRAY(4)+1242)/10.23724)
      IF(DATRAY(4).LT.0)DATRAY(4)=0
      DATRAY(5)=NINT((DATRAY(5)+1856)/8.06333)
      IF(DATRAY(5).LT.0)DATRAY(5)=0
      DELTA4=DATRAY(4)-DATA4
      IF(ABS(DELTA4).LT.2)THEN
         DATRAY(4)=DATA4
      ELSE
         DATA4=DATRAY(4)
      ENDIF
      DELTA5=DATRAY(5)-DATA5
      IF(ABS(DELTA5).LT.2)THEN
         DATRAY(5)=DATA5
      ELSE
         DATA5=DATRAY(5)
      ENDIF

C     DETECTING WHETHER GRIP OR MEMORY SWITCHES ARE PRESSED
      CALL DISTAT(0,0,GRIP,0)
      CALL DISTAT(0,1,MEMORY,0)
```

```
C     OUTPUTING THE DIGITIZED VALUES TO THE ACTUAL ARM
      CALL ARMOUT(DATRAY)

C     OPENING/CLOSING THE GRIPPER WHEN GRIP SWITCH IS DETECTED
      IF(GRIP.EQ.0)GOTO 100
      IF(GRPMOD.EQ.1)GOTO 90
C     OPENING THE GRIPPER
      CALL DIGOUT(1,5)
 85   CALL DISTAT(0,0,GRIP,0)
      IF(GRIP.EQ.1)GOTO 85
      CALL DIGOUT(0,5)
      CALL ADIN(5,GPOT)
      GRPMOD=1
      GOTO 110
C     CLOSING THE GRIPPER
 90   CALL DIGOUT(2,5)
 95   CALL ADIN(5,DATA)
      IF(DATA.GT.0)GOTO 95
      CALL DIGOUT(0,5)
      GPOT=0
      GRPMOD=0
      GOTO 110

C     STORING JOINT POSITION DATA WHEN MEMORY SWITCH IS DETECTED.
 100  IF(MEMORY.EQ.0)GOTO 60
      IF(MEMODE.EQ.1)GOTO 70
 110  MEMODE=1
      CALL BEEP(1)
      JOINT(1,M)=DATRAY(1)
      JOINT(2,M)=DATRAY(2)
      JOINT(3,M)=DATRAY(3)
      JOINT(4,M)=DATRAY(4)
      JOINT(5,M)=DATRAY(5)
      JOINT(6,M)=GPOT

C     ADVANCING TO THE NEXT MEMORY LOCATION AND LIMITING THE STORED
C     DATA TO 50 SETS
      M=M+1
      IF(M.GT.50)GOTO 130
      IF(M.NE.45)GOTO 70
      CALL BEEP(5)
      WRITE(*,120)
 120  FORMAT(' WARNING: ONLY 5 MORE MEMORIES LEFT.',)
      GOTO 70

 130  M=50
      CALL BEEP(10)
      WRITE(*,135)
 135  FORMAT(///,15X,' MEMORY FULL! LEARN PROCESS TERMINATED.',
     *//,' DO YOU WANT TO KEEP THE DATA ? (Y/N)')
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')GOTO 190
      WRITE(*,137)FNAME
 137  FORMAT(//,' THE FILE ',A,' IS DISCARDED.'//)
      CLOSE(5,STATUS='DELETE')
```



```
      WRITE(*,138)
 138  FORMAT(//,' DO YOU WANT TO TRY AGAIN? (Y/N) ',///)
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')GOTO 50

      GOTO 500

 140  M=M-1
      WRITE(*,150)M
 150  FORMAT(///,' LEARN PROCESS TERMINATED.',
     *//,' NO. OF POINTS RECORDED =',I4,//,)

C     DETECTING IF ENOUGH DATA ARE RECORDED
      IF(M.GT.1)GOTO 190
      WRITE(*,160)
 160  FORMAT(//,' DATA RECORDED WILL NOT BE ENOUGH FOR A JOB.'
     *//,' PLEASE TRY AGAIN.'//)
      REWIND(5)
      GOTO 55

C     STORING DATA TO FILE SPECIFIED
 190  DO 200 I=1,M-1
         WRITE(5,'(I5,5I10,I5)')I,(JOINT(J,I),J=1,6)
 200  CONTINUE


C     DETECTING IF END POINTS ARE MATCHED
      DO 205 J=1,5
        IF(JOINT(J,1).NE.JOINT(J,M))GOTO 210
 205  CONTINUE
      WRITE(5,'(I5,5I10,I5)')M,(JOINT(J,M),J=1,6)
      L=M
      GOTO 400

 210  L=M
      WRITE(*,220)
 220  FORMAT(' THE FINAL POSITION DOES NOT MATCH WITH THE INITIAL.',
     *//,' DOES IT MATTER ? (Y/N)')
      READ(*,'(A)')KEY
      IF(KEY.EQ.'N')GOTO 310
 230  WRITE(*,240)
 240  FORMAT(//,' PLEASE SELECT THE FOLLOWING OPTIONS:-',
     *//,5X,'1 = ADJUSTING THE END POINT TO MATCH THE STARTING POINT.',
     *//,5X,'2 = USING AN ARBITRARY ROUTE TO JOINT THE END POINTS.',
     *//,5X,'3 = FOLLOWING THE SAME ROUTE BACK TO THE STARTING POINT.',
     *//)
      READ(*,*,ERR=230)IBACK
      IF((IBACK.LE.0).OR.(IBACK.GT.3))GOTO 230
      GRIPER=0
      GOTO(260,270,290)IBACK

 260  L=M
      WRITE(5,'(I5,5I10,I5)')M,(JOINT(J,1),J=1,5),JOINT(6,M)
      GOTO 400
```

```
270   L=M+1
      WRITE(5,'(I5,5I10,I5)')M,(JOINT(J,M),J=1,6)
      IF(JOINT(6,M).NE.0)THEN
         DO 280 J=1,5
            JOINT(J,2)=JOINT(J,M)+2
280      CONTINUE
         WRITE(5,'(I5,5I10,I5)')L,(JOINT(J,2),J=1,5),GRIPER
         L=M+2
      ENDIF
      WRITE(5,'(I5,5I10,I5)')L,(JOINT(J,1),J=1,5),GRIPER
      GOTO 400

290   DO 300 I=1,M
         L=M+I-1
         K=M-I+1
         WRITE(5,'(I5,5I10,I5)')L,(JOINT(J,K),J=1,5),GRIPER
300   CONTINUE
      GOTO 400

310   L=M
      WRITE(5,'(I5,5I10,I5)')M,(JOINT(J,M),J=1,6)
      GOTO 400

400   WRITE(*,410)FNAME,L
410   FORMAT(//,' ALL JOINT DATA ARE STORED IN THE FILE:- ',A,
     *//,' TOTAL NO. OF RECORD =',I4,//)
      CLOSE(5)

C     CLOSING THE GRIPPER IF IT IS OPEN BEFORE EXITING THE PROGRAM.
500   CALL ADIN(5,IPOT5)
      IF(IPOT5.GT.0)THEN
         CALL DIGOUT(2,5)
510      CALL ADIN(5,IPOT5)
         IF(IPOT5.GT.0)GOTO 510
         CALL DIGOUT(0,5)
      ENDIF
      CONTINUE

      END
```



```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM TRIM

      IMPLICIT INTEGER(A-Z)
      DIMENSION TRIMDAT(100,6),DATA(6),OLDDAT(6)
      CHARACTER FNAME*11,KEY*1
      LOGICAL OLDFILE,CYCLIC,LAST
      COMMON TRIMDAT

C     CHECKING IF FILENAME HAS BEEN SPECIFIED
      INQUIRE(FILE='NAME.JOB',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 50
      OPEN(50,FILE='NAME.JOB')
      READ(50,'(A)',ERR=50)FNAME
      GOTO 70

C     REQUESTING A FILENAME HAS IT NOT BEEN SPECIFIIED
 50   WRITE(*,60)
 60   FORMAT(//,' PLEASE INPUT A FILENAME, XXX.DAT, IN WHICH',
     */,' DATA ARE TO BE TRIMMED.',
     *//,' [NOTE: XXX MUST BE LESS THAN 7 CHARACTERS.]',//)
      READ(*,'(A)')FNAME

C     CHECKING IF SPECIFIED FILE IS A NEW FILE
 70   INQUIRE(FILE=FNAME,EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 200

      WRITE(*,75)
 75   FORMAT(//////////,20X,'DATA IS BEING TRIMMED.',
     *//,11X,'ANY DUMMY DATA DETECTED WILL BE REMOVED.',//////////////)

C     OPENING FILE-5 DEFINED TO BE THE SPECIFIED FILE
      OPEN(5,FILE=FNAME)

C     SETTING COUNTERS. N FOR DATA & I FOR TRIMMED DATA
      N=1
      I=1

      READ(5,80,IOSTAT=IOCHK,ERR=160)(DATA(J),J=1,6)
 80   FORMAT(5X,5I10,I5)

 90   N=N+1
      DO 100 J=1,6
         OLDDAT(J)=DATA(J)
 100  CONTINUE

      READ(5,80,END=130)(DATA(J),J=1,6)

C     COMPARING TWO CONSECUTIVE DATA
      DO 110 J=1,3
         IF(OLDDAT(J).NE.DATA(J))GOTO 115
 110  CONTINUE
      GOTO 90
```

```
C     STORING THE OLD DATA IF TWO DATA DO NOT MATCH.
 115  DO 120 J=1,6
         TRIMDAT(I,J)=OLDDAT(J)
 120  CONTINUE
      I=I+1
      GOTO 90

C     STORING THE LAST DATA.
 130  DO 132 J=1,6
         TRIMDAT(I,J)=OLDDAT(J)
 132  CONTINUE

C     CHECKING DATA SET TO SEE IF CYCLIC, AND WRITE TO THE FILE.
      DO 134 J=1,5
         IF(TRIMDAT(1,J).NE.TRIMDAT(I,J))GOTO 136
 134  CONTINUE
      CYCLIC=.TRUE.
      GOTO 138
 136  CYCLIC=.FALSE.
 138  REWIND(5)
      WRITE(5,'(L5)')CYCLIC

      N=N-1
      DUMMY=N-I

C     INSERTING DATA TO A SECTION APPROPRIATELY.
      DATINS=0
      LAST=.FALSE.
      JBASE=1
      J=1
 140  IGRIP=TRIMDAT(J,6)
      IDATA=1
 142  IF(TRIMDAT(J+1,6).NE.IGRIP)GOTO 145
      J=J+1
      IF(J.EQ.I)GOTO 144
      IDATA=IDATA+1
      GOTO 142
 144  LAST=.TRUE.
 145  WRITE(5,500)(TRIMDAT(JBASE,M),M=1,6)
 500  FORMAT(5X,5I10,I5)
      IF(IDATA.GE.4)THEN
         DO 146 K=1,IDATA-1
            WRITE(5,500)(TRIMDAT(JBASE+K,M),M=1,6)
 146     CONTINUE
      ELSEIF(IDATA.EQ.3)THEN
         WRITE(5,500)(TRIMDAT(JBASE+1,M),M=1,6)
         JBASE1=JBASE+1
         CALL INSERT(JBASE1)
         WRITE(5,500)(TRIMDAT(JBASE+2,M),M=1,6)
         DATINS=DATINS+1
      ELSEIF(IDATA.EQ.2)THEN
         CALL INSERT(JBASE)
         WRITE(5,500)(TRIMDAT(JBASE+1,M),M=1,6)
         JBASE1=JBASE+1
```



```
         CALL INSERT(JBASE1)
         DATINS=DATINS+2
      ENDIF
 148  IF(LAST)GOTO 150
      J=J+1
      IF(J.EQ.I)GOTO 150
      JBASE=J
      GOTO 140
 150  WRITE(5,500)(TRIMDAT(I,M),M=1,6)
 152  CLOSE(5)

      WRITE(*,155)FNAME,N,DUMMY,DATINS
 155  FORMAT(//,20X,' TRIM PROCESS TERMINATED.',
     *//,10X,' NO. OF RECORD IN ORIGINAL FILE:-',A,' = ',I4,
     *//,10X,' NO. OF DUMMY DATA DELETED = ',I4,
     *//,10X,' NO. OF DATA INSERTED = ',I4,///)
      GOTO 220

 160  IF(IOCHK.NE.0)GOTO 180
      WRITE(*,170)FNAME
 170  FORMAT(//,' THERE IS NO DATA IN FILE ',A)
      GOTO 215

 180  WRITE(*,190)FNAME
 190  FORMAT(//,' FILE ',A,' IS NOT IN APPROPRIATE FORMAT.')
      GOTO 215

 200  WRITE(*,210)FNAME
 210  FORMAT(//,' FILE ',A,' DOES NOT EXIST.')

 215  WRITE(*,217)
 217  FORMAT(//,' DO YOU WANT TO TRY ANOTHER FILE ? (Y/N) ',//)
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')GOTO 50

 220  CONTINUE

      END


      SUBROUTINE INSERT(J)
      INTEGER TEMP(5),TRIMDAT(100,6)
      COMMON TRIMDAT
      DO 50 I=1,5
         TEMP(I)=(TRIMDAT(J,I)+TRIMDAT(J+1,I))/2
  50  CONTINUE
      WRITE(5,'(5X,5I10,I5)')(TEMP(I),I=1,5),TRIMDAT(J,6)
      RETURN
      END
```

```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM SMOOTH

      REAL VCMX(3),VCMN(3),ACMX(3),ACMN(3)
      INTEGER BODY(50),SHODER(50),ELBOW(50),WPITCH(50),WROLL(50)
      INTEGER GRIPER(2)
      CHARACTER FNAME*11,OPMODE*1
      LOGICAL OLDJOB,FAST,CYCLIC

      COMMON BODY,SHODER,ELBOW,WPITCH,WROLL,GRIPER,N,TSUM
      COMMON /BLK3/VCMX,VCMN,ACMX,ACMN

C     DETECTING WHETHER FILENAME HAS BEEN SPECIFIED
      INQUIRE(FILE='NAME.JOB',EXIST=OLDJOB)
      IF(.NOT.OLDJOB)GOTO 30
      OPEN(50,FILE='NAME.JOB')
      READ(50,'(A)',ERR=30)FNAME
      GOTO 46

C     REQUESTING A FILENAME FROM WHICH DATA ARE TO BE SMOOTHED
 30   WRITE(*,40)
 40   FORMAT(' PLEASE INPUT A FILENAME, XXX.DAT, FROM WHICH',
     */,' DATA ARE TO BE CURVE-FITTED.',
     *//,' [NOTE: XXX MUST BE LESS THAN 7 CHARACTERS. ]',//)
      READ(*,'(A)')FNAME

C     CHECKING THE SPECIFIED FILE IF IT IS NEW
 46   INQUIRE(FILE=FNAME,EXIST=OLDJOB)
      IF(.NOT.OLDJOB)GOTO 120

C     SETTING UP VELOCITY & ACCELERATION CONSTRAINTS.
      VCMX(1)=574
      VCMN(1)=-574
      VCMX(2)=1140
      VCMN(2)=-3280
      VCMX(3)=1200
      VCMN(3)=-1800
      ACMX(1)=2332
      ACMN(1)=-2332
      ACMX(2)=8497
      ACMN(2)=-11015
      ACMX(3)=7596
      ACMN(3)=-7452

C     SELECTING OPTIONS.
 48   WRITE(*,49)
 49   FORMAT(//,' PLEASE CHOOSE ONE OF THE FOLLOWING OPTIONS:-',
     *//,5X,' K = KEEP ALL THE WRIST POSITIONS AT SPECIFIED POINTS.',
     *//,5X,' A = KEEP ONLY THE WRIST POSITIONS AT THE GRIP POINTS.',
     *//)
      READ(*,'(A)')OPMODE
      IF(OPMODE.EQ.'K')FAST=.FALSE.
      IF(OPMODE.EQ.'A')FAST=.TRUE.
```



```
      IF((OPMODE.NE.'A').AND.(OPMODE.NE.'K'))GOTO 48

C     OPENING THE FILE-5 DEFINED TO BE THE SPECIFIED DATA FILE
C     FROM WHICH DATA ARE TO BE READ
      OPEN(5,FILE=FNAME)

C     READING DATA TO SEE IF DATA SET IS REPEATABLE.
      READ(5,'(L5)',ERR=180)CYCLIC

C     OPENING THE FILE-20 FOR STORING SMOOTHED DATA
      OPEN(20,FILE='BUFFER.JOB',STATUS='NEW')
      WRITE(20,'(20X,A)')FNAME

C     RECORDING SELECTED OPTION.
      WRITE(20,'(2L5)')FAST,CYCLIC

C     SETTING THE POINTER-N TO 1 FOR THE FIRST READ DATA AND
C     START READING DATA FROM THE SPECIFIED FILE
      TIME=0
      MSECTN=0
      N=1
      READ(5,50,END=160,ERR=180)BODY(N),SHODER(N),ELBOW(N),WPITCH(N),
     *WROLL(N),GRIPER(1)
 50   FORMAT(5X,5I10,I5)

C     WRITING THE INITAIL DATA POINTS TO BUFFER.JOB.
      WRITE(20,'(5I10)')BODY(1),SHODER(1),ELBOW(1),WPITCH(1),WROLL(1)

C     SETTING ACTUAL IGRIP & READING THE SECOND DATA
      IGRIP=0
      IF(GRIPER(1).NE.0)IGRIP=1
      WRITE(20,'(I10)')IGRIP
      N=2
      READ(5,50,END=140,ERR=180)BODY(N),SHODER(N),ELBOW(N),WPITCH(N),
     *WROLL(N),GRIPER(2)

C     DETECTING STATUS OF GRIPPER. IF CHANGED, THE FIRST READ
C     DATA UP TO THE DATA JUST READ ARE TREATED AS A SECTION.
 60   IF(GRIPER(2).NE.0)GOTO 70
      IF(IGRIP.EQ.0)GOTO 80
      CALL SPLINE(FAST)
      IGRIP=0
      GOTO 75

 70   IF(IGRIP.EQ.1)GOTO 80
      CALL SPLINE(FAST)
      IGRIP=1

C     SETTING THE END POINT OF THE LAST SECTION AS THE INITAL
C     POINT OF THE NEXT SECTION.
 75   BODY(1)=BODY(N)
      SHODER(1)=SHODER(N)
      ELBOW(1)=ELBOW(N)
      WPITCH(1)=WPITCH(N)
      WROLL(1)=WROLL(N)
```

```
      GRIPER(1)=GRIPER(2)
      MSECTN=MSECTN+1
      TIME=(TIME+TSUM)
      N=2
      GOTO 85

 80   N=N+1
 85   READ(5,50,END=90)BODY(N),SHODER(N),ELBOW(N),WPITCH(N),WROLL(N),
     *GRIPER(2)
      GOTO 60

C     DETECTING NO. OF DATA READ WHEN END OF SPECIFIED FILE IS
C     ENCOUNTERED. IF NO DATA IS READ, THE PROGRAM WILL BE TERMINATED.
 90   N=N-1
      IF(N.EQ.1)GOTO 100
      CALL SPLINE(FAST)
      MSECTN=MSECTN+1
      TIME=TIME+TSUM

 100  CLOSE(5)
      CLOSE(20)

C     CONVERTING TIME INTO SEC.
      TIME=TIME/1000.0

      WRITE(*,110)MSECTN,TIME
 110  FORMAT(//////////,20X,'SMOOTH PROCESS FINISHED.',
     *//,10X,'DATA ARE PUT TO BUFFER AND READY TO BE USED.',
     *///,21X,'NO. OF SECTIONS = ',I2,
     *//,21X,'TOTAL PATH TIME REQUIRED = ',F5.1,' SEC.',//////////)

      GOTO 200

 120  WRITE(*,130)
 130  FORMAT(' FILE NOT FOUND. PLEASE SPECIFY ANOTHER FILE.',//)
      GOTO 30

 140  WRITE(*,150)FNAME
 150  FORMAT(//,' DATA IN FILE ',A,' IS NOT ENOUGH FOR A JOB.',
     */,' PLEASE TRY ANOTHER FILE.',//)
      GOTO 30

 160  WRITE(*,170)FNAME
 170  FORMAT(//,' THERE IS NO DATA IN FILE ',A,' .',
     */,' PLEASE TRY ANOTHER FILE.',//)
      GOTO 30

 180  WRITE(*,190)FNAME
 190  FORMAT(//,' DATA IN FILE ',A,'IS NOT IN APPROPRIATE FORMAT.',
     */,' PLEASE TRY ANOTHER FILE.',//)
      GOTO 30

 200  CONTINUE

      END
```



```
C     THIS SUBROUTINE IS TO EVALUATE SPLINE FUNCTIONS FOR EACH
C     DATA SEGMENT FOR EACH JOINT.
      SUBROUTINE SPLINE(FAST)

      REAL U(50),V(50),W(50)
      INTEGER H,BODY(50),SHODER(50),ELBOW(50),WPITCH(50),WROLL(50)
      INTEGER GRIPER(2)
      DIMENSION T(50),F(50,3),FDASH(50,3)
      DIMENSION VMX(3),VMN(3),AMX(3),AMN(3),VCMX(3),VCMN(3),ACMX(3)
      DIMENSION ACMN(3)
      LOGICAL FAST

      COMMON BODY,SHODER,ELBOW,WPITCH,WROLL,GRIPER,N,TSUM
      COMMON /BLK1/B1,B2,B3,B4,B5,B6
      COMMON /BLK2/T,VMX,VMN,AMX,AMN
      COMMON /BLK3/VCMX,VCMN,ACMX,ACMN

C     INITIALIZING MAX. & MIN. VELOCITIES & ACCELERATIONS TO ZEROS.
      DO 101 J=1,3
         VMX(J)=0
         VMN(J)=0
         AMX(J)=0
         AMN(J)=0
 101  CONTINUE

C     OPENING A TEMPORARY FILE TO STORE SPLINE COEFFICIENTS.
      OPEN(30,FILE='SPLINE.TMP',STATUS='NEW')

C     WRITING THE AMOUNT THAT THE GRIPPER NEEDED TO BE OPENED.
      WRITE(20,'(I10)')GRIPER(1)

C     WRITING NO. OF POINTS FOR THIS SECTION.
      WRITE(20,'(I10)')N

C     SETTING THE F MATRIX
      DO 30 I=1,N
         F(I,1)=FLOAT(BODY(I))
         F(I,2)=FLOAT(SHODER(I))
         F(I,3)=FLOAT(ELBOW(I))
 30   CONTINUE

C     CALCULATING TIME INTERVAL BETWEEN TWO DATA POINTS FOR
C     EACH SEGMENT
      TSUM=0
      DO 50 K=1,N-1
         SUMSQ=0
         DO 40 J=1,3
            SUMSQ1=(F(K+1,J)-F(K,J))**2
            SUMSQ=SUMSQ+SUMSQ1
 40      CONTINUE
         T(K+1)=SQRT(SUMSQ)
         TSUM=TSUM+T(K+1)
         IF(T(K+1).EQ.0)GOTO 666
```

```
 50   CONTINUE

C     DETECTING IF TWO DATA ARE RECORDED 5-ORDER SPLINE FUNCTION
C     WILL BE USED.
      IF(N.EQ.2)GOTO 200
      IF(N.LT.5)GOTO 999

C     SETTING UP EFFICIENT ALGORITHM OPERATORS.
      N2=N-2
      U(1)=2*T(2)+3*T(3)
      V(1)=-T(2)/U(1)
      DO 60 I=2,N2-1
         U(I)=T(I+2)*V(I-1)+2*(T(I+1)+T(I+2))
         V(I)=-T(I+1)/U(I)
 60   CONTINUE
      U(N2)=T(N)*V(N2-1)+3*T(N-1)+2*T(N)

C     CALCULATING THE INTERMEDIATE VELOCITIES FOR EACH SEGMENT.
      DO 90 J=1,3
         D=3*((T(2)**2)*(F(3,J)-F(2,J))+
     *        2*(T(3)**2)*(F(2,J)-F(1,J)))/(T(2)*T(3))
         W(1)=D/U(1)
         DO 70 I=2,N2-1
            D=3*((T(I+1)**2)*(F(I+2,J)-F(I+1,J))+
     *           (T(I+2)**2)*(F(I+1,J)-F(I,J)))/(T(I+1)*T(I+2))
            W(I)=(D-T(I+2)*W(I-1))/U(I)
 70      CONTINUE
         D=3*(2*(T(N-1)**2)*(F(N,J)-F(N-1,J))+
     *        (T(N)**2)*(F(N-1,J)-F(N-2,J)))/(T(N-1)*T(N))
         W(N2)=(D-T(N)*W(N2-1))/U(N2)
         FDASH(N2,J)=W(N2)
         DO 80 I=1,N2-1
            FDASH(N2-I,J)=V(N2-I)*FDASH(N2-I+1,J)+W(N2-I)
 80      CONTINUE
 90   CONTINUE

C     SETTING UP THE SPLINE FUNCTION FOR THE FIRST SEGMENT
C     FOR EACH JOINT.
      ISEG=1
      DO 103 J=1,3
         B1=F(1,J)
         B2=0
         B3=0
         B4=4*(F(2,J)-F(1,J))/(T(2)**3)-FDASH(1,J)/(T(2)**2)
         B5=3*(F(1,J)-F(2,J))/(T(2)**4)+FDASH(1,J)/(T(2)**3)
         B6=0
         IF(ABS(B5).LT.1E-30)THEN
            TVMAX=T(2)
            TAMAX=T(2)
         ELSE
            TVMAX=-B4/(2*B5)
            TAMAX=-B4/(4*B5)
         ENDIF
         CALL SORT(ISEG,J,TVMAX,TAMAX)
         WRITE(30,'(5E12.5)')B1,B2,B3,B4,B5
```



```
103   CONTINUE

C     SETTING UP THE SPLINE FUNCTIONS FOR THE SECOND UP TO THE
C     LAST SECOND SEGMENTS FOR EACH JOINT
      N1=N-1
105   ISEG=ISEG+1
      IF(ISEG.EQ.N1)GOTO 115
      DO 110 J=1,3
         B1=F(ISEG,J)
         B2=FDASH(ISEG-1,J)
         B3=(3*(F(ISEG+1,J)-F(ISEG,J))/T(ISEG+1)-2*FDASH(ISEG-1,J)-
     *        FDASH(ISEG,J))/T(ISEG+1)
         B4=(2*(F(ISEG,J)-F(ISEG+1,J))/T(ISEG+1)+FDASH(ISEG-1,J)+
     *        FDASH(ISEG,J))/(T(ISEG+1)**2)
         B5=0
         B6=0
         IF(ABS(B4).LT.1E-30)THEN
            TVMAX=T(ISEG+1)
         ELSE
            TVMAX=-B3/(3*B4)
         ENDIF
         TAMAX=T(ISEG+1)
         CALL SORT(ISEG,J,TVMAX,TAMAX)
         WRITE(30,'(4E12.5)')B1,B2,B3,B4
110   CONTINUE
      GOTO 105

C     SETTING UP THE SPLINE FUNCTION FOR THE LAST SEGMENT FOR
C     EACH JOINT
115   DO 130 J=1,3
         B1=F(N-1,J)
         B2=FDASH(N-2,J)
         B3=(6*F(N,J)-6*F(N-1,J)-3*FDASH(N-2,J)*T(N))/(T(N)**2)
         B4=(-8*F(N,J)+8*F(N-1,J)+3*FDASH(N-2,J)*T(N))/(T(N)**3)
         B5=(3*F(N,J)-3*F(N-1,J)-FDASH(N-2,J)*T(N))/(T(N)**4)
         B6=0
         IF(ABS(B5).LT.1E-30)THEN
            TVMAX=T(N)
            TAMAX=T(N)
         ELSE
            TVMAX=B3/(6*B5*T(N))
            TAMAX=-B4/(4*B5)
         ENDIF
         CALL SORT(ISEG,J,TVMAX,TAMAX)
         WRITE(30,'(5E12.5)')B1,B2,B3,B4,B5
130   CONTINUE
      GOTO 300

C     CALCULATING THE SPLINE FUNCTION WHEN ONLY TWO DATA ARE GIVEN.
200   ISEG=1
      DO 220 J=1,3
         B1=F(1,J)
         B2=0
         B3=0
         B4=10*(F(2,J)-F(1,J))/(T(2)**3)
```

```
         B5=-15*(F(2,J)-F(1,J))/(T(2)**4)
         B6=6*(F(2,J)-F(1,J))/(T(2)**5)
      TVMAX=T(2)/2
      TAMAX=0.788675134*T(2)
      CALL SORT(ISEG,J,TVMAX,TAMAX)
      WRITE(30,'(6E12.5)')B1,B2,B3,B4,B5,B6
 220  CONTINUE

C     EVALUATING TIME SCALE FACTOR TO CONVERT TO REAL TIME.

C     EVALUATING SCALE FACTOR, SVA, DUE TO VELOCITY & ACCELERATION
C     CONSTRAINS.
 300  SVA=0
      DO 310 J=1,3
         SVMAX=VMX(J)/VCMX(J)
         SVMIN=VMN(J)/VCMN(J)
         IF(AMX(J).GT.0)THEN
            SAMAX=SQRT(AMX(J)/ACMX(J))
         ELSE
            SAMAX=0
         ENDIF
         IF(AMN(J).LT.0)THEN
            SAMIN=SQRT(AMN(J)/ACMN(J))
         ELSE
            SAMIN=0
         ENDIF
         SVA=MAX(SVMAX,SVMIN,SAMAX,SAMIN,SVA)
 310  CONTINUE

C     EVALUATING SCALE FACTOR, ST, DUE TO TIME CONSTRAINTS.
      IF(FAST)THEN
         TC=(REAL(WPITCH(N)+WROLL(N)-WPITCH(1)-WROLL(1)))/20
         ST=ABS(TC/TSUM)
         WRITE(20,'(2I10)')WPITCH(N),WROLL(N)
      ELSE
         ST=0
         DO 315 K=1,N-1
            TC=(REAL(WPITCH(K+1)+WROLL(K+1)-WPITCH(K)-WROLL(K)))/20
            STC=ABS(TC/T(K+1))
            ST=MAX(STC,ST)
            WRITE(20,'(2I10)')WPITCH(K+1),WROLL(K+1)
 315     CONTINUE
      ENDIF

C     EVALUATING THE ULTIMATE SCALE FACTOR, S.
      S=MAX(SVA,ST)

C     CONVERTING THE SCALE FACTOR IN THE UNIT OF MSEC.
      S=1000*S

C     RENEWING COEFFICIENTS OF SPLINE TO REAL TIME FUNCTION.
      REWIND(30)
      IF(N.EQ.2)GOTO 380

C     RENEWING FOR THE 1ST SEGMENT.
```



```
      K=1
      DO 330 J=1,3
         READ(30,'(5E12.5)')B1,B2,B3,B4,B5
         CALL RENEW(S)
         WRITE(20,'(3E12.5)')B1,B4,B5
 330  CONTINUE

C     RENEWING FOR THE INTERMEDIATE SEGMENTS.
 340  K=K+1
      IF(K.EQ.N1)GOTO 360
      DO 350 J=1,3
         READ(30,'(4E12.5)')B1,B2,B3,B4
         CALL RENEW(S)
         WRITE(20,'(4E12.5)')B1,B2,B3,B4
 350  CONTINUE
      GOTO 340

C     RENEWING FOR THE LAST SEGMENT.
 360  DO 370 J=1,3
         READ(30,'(5E12.5)')B1,B2,B3,B4,B5
         CALL RENEW(S)
         WRITE(20,'(5E12.5)')B1,B2,B3,B4,B5
 370  CONTINUE
      GOTO 400

C     RENEWING WHEN ONLY 2 POINTS RECORDED.
 380  DO 390 J=1,3
         READ(30,'(6E12.5)')B1,B2,B3,B4,B5,B6
         CALL RENEW(S)
         WRITE(20,'(4E12.5)')B1,B4,B5,B6
 390  CONTINUE

 400  CLOSE(30,STATUS='DELETE')

C     CALCULATING THE REAL TIME IN MSEC FOR EACH SEGMENT.
      TSUM=S*TSUM
      DO 410 K=1,N-1
         T(K+1)=S*T(K+1)
         WRITE(20,'(E12.5)')T(K+1)
 410  CONTINUE

      RETURN

C     TERMINATING THE PROGRAM WHEN DUMMY DATA DETECTED
 666  STOP 'DUMMY DATA EXIST. PLEASE REPROGRAM THE ROBOT.'

C     TERMINATING THE PROGRAM WHEN SECTION NO. NOT MATCH.
 999  STOP 'BAD DATA. PLEASE REPROGRAM THE ROBOT.'

      END


C     THIS SUBROUTINE IS TO EVALUATE THE MAXIMUN VELOCITIES AND
C     ACCELERATIONS ON EACH JOINT.
      SUBROUTINE SORT(ISEG,J,TVMAX,TAMAX)
```

```
      DIMENSION T(50),VMX(3),VMN(3),AMX(3),AMN(3)

      COMMON /BLK1/B1,B2,B3,B4,B5,B6
      COMMON /BLK2/T,VMX,VMN,AMX,AMN

C     SETTING FINCTIONS FOR EVALUATING VELOCITY & ACCELERATION.
      VEL(TIME)=B2+2*B3*TIME+3*B4*(TIME**2)+4*B5*(TIME**3)+
     *          5*B6*(TIME**4)
      ACC(TIME)=2*B3+6*B4*TIME+12*B5*(TIME**2)+20*B6*(TIME**3)

C     CALCULATING MAX. & MIN. VELOCITIES & ACCELERATIONS.
      K=ISEG+1
      IF(TVMAX.GT.T(K))TVMAX=T(K)
      VMAX=VEL(TVMAX)
      IF(TAMAX.GT.T(K))TAMAX=T(K)
      AMAX=ACC(TAMAX)

C     CHOOSING THE GLOBAL MAX. & MIN. OF VELOCITIES & ACCELERATIONS.
      VMX(J)=MAX(VMAX,VMX(J))
      VMN(J)=MIN(VMAX,VMN(J))
      AMX(J)=MAX(AMAX,AMX(J))
      AMN(J)=MIN(AMAX,AMN(J))

      RETURN
      END



C     THIS SUBROUTINE IS TO RENEW THE COEFFICIENTS OF THE SPLINE
C     FUNCTION INTO REAL TIME FUNCTION.
      SUBROUTINE RENEW(S)

      COMMON /BLK1/B1,B2,B3,B4,B5,B6

      B2=B2/S
      B3=B3/(S**2)
      B4=B4/(S**3)
      B5=B5/(S**4)
      B6=B6/(S**5)

      RETURN
      END
```



```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM GO

      REAL B1(50,3),B2(50,3),B3(50,3),B4(50,3),B5(2,3),B6(1,3)
      REAL TOUT(50)
      INTEGER W(2,50)
      DIMENSION IDATA(5)
      LOGICAL OLDJOB,FAST,NFAST,CYCLIC
      CHARACTER FNAME*11,KEY*1

C     INITIALIZING LAB-PACK SUBROUTINES.
      CALL INIT
      CALL INTROFF
      CALL STPORT

C     OPENING FILE-BUFFER.JOB .
      INQUIRE(FILE='BUFFER.JOB',EXIST=OLDJOB)
      IF(.NOT.OLDJOB)GOTO 24
      OPEN(7,FILE='BUFFER.JOB')

C     READING THE FILE NAME OF THE DATA FROM WHICH BUFFER.JOB IS
C     CREATED.
 10   READ(7,'(20X,A)',END=420,ERR=440)FNAME

C     DISPLAYING DATA FILE NAME.
      WRITE(*,20)FNAME
 20   FORMAT(//,' JOB DATA IS FROM:- ',A,/,' OK ? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')GOTO 44
      CLOSE(7)
 24   WRITE(*,28)
 28   FORMAT(//,' PLEASE SPECIFY ANOTHER JOB FILE NAME.',//)
      READ(*,'(A)')FNAME
      INQUIRE(FILE=FNAME,EXIST=OLDJOB)
      IF(OLDJOB)GOTO 36
      WRITE(*,32)
 32   FORMAT(//,' JOB FILE NOT FOUND.',//)
      GOTO 24

 36   OPEN(7,FILE=FNAME)
      GOTO 10

 44   WRITE(*,48)
 48   FORMAT(//,' DO YOU WANT TO RE-ALIGN THE WRIST ? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.EQ.'N')GOTO 52

C     RESETTING REFERENCE OF THE WRIST POSITIONS.
      WRITE(*,40)
 40   FORMAT(//,' PLEASE FOLLOW THE INSTRUCTIONS BELOW TO RESET THE',
     */,' WRIST POSITIONS:- ',
     *//,5X,'1. TURN THE POWER SUPPLY OF THE ARM OFF.',
     *//,5X,'2. POSITION THE WRIST TO THE REFERENCE LINES.',
```

```
     *//,5X,'3. PUT ALIGNMENT SWITCH TO ON POSITION.',
     *//,5X,'4. TURN THE POWER ON OF THE ARM ON.',
     *//,5X,'4. PRESS ANY KEY WHEN ALL THE ABOVE DONE.',//)
      READ(*,'(A)')KEY
      IDATA(4)=100
      IDATA(5)=150
      CALL WRIST(IDATA(4))
      WRITE(*,50)
 50   FORMAT(//,' WRIST REFERENCE POSITION SET.',
     *//,' PLEASE SWITCH ALIGNMENT SWITCH BACK TO OP. POSITION.',
     *//,' PRESS ANY KEY WHEN READY.',//)
      READ(*,'(A)')KEY

C     READING THE APPROPRIATE OPERATION MODE.
 52   READ(7,'(2L5)',ERR=440)FAST,CYCLIC
      NFAST=.NOT.FAST

C     REQUESTING NO. OF TIMES THE PROCESS WANTED TO BE REPEATED.
      IF(CYCLIC)THEN
 54      WRITE(*,56)
 56      FORMAT(//,' PLEASE SPECIFY NO. OF TIMES THAT THE PROCESS',
     *   /,' TO BE REPEATED.',
     *   /,' [NOTE: ENTER 0 FOR UNLIMITED NO. OF TIMES.  ]',//)
         READ(*,*,ERR=54)NTIMES
         IF(NTIMES.LT.0)GOTO 54
      ELSE
         NTIMES=1
      ENDIF

C     POSITION THE ARM TO STARTING POSITION.
      READ(7,'(5I10)',ERR=440)(IDATA(J),J=1,5)
      CALL SELFADJ(IDATA)

      PAUSE ' THE ROBOT IS READY TO GO.'
      WRITE(*,57)
 57   FORMAT(//////////,14X,' ROBOT IS IN OPERATION.',
     *//,10X,'PRESS ESC-BUTTON FOR EMERGENCY STOP.',//////////////)

C     READING REQUIRED GRIPPER STATUS AND VALUE ON GRIPPER POT.
      IGSTAT=0
      MTIMES=1
 58   READ(7,'(I10)',ERR=440)IGRIP
      READ(7,'(I10)',ERR=440)IGPOT
      IF(IGSTAT.EQ.0)THEN
         IF(IGRIP.EQ.1)THEN
            CALL GOPEN(IGPOT)
            IGSTAT=1
         ENDIF
      ELSE
         IF(IGRIP.EQ.0)THEN
            CALL GCLOSE
            IGSTAT=0
         ENDIF
      ENDIF
```



```
C     READING NO. OF POINTS FOR THE SECTION.
 60   READ(7,'(I10)',ERR=440)N
      NN1=N-1
      IF(FAST)NN1=1
      N2=N-2
      N1=N-1

C     READING WRIST POSITIONS.
      DO 65 K=1,NN1
         READ(7,'(2I10)',ERR=440)(W(J,K),J=1,2)
 65   CONTINUE

      IF(N.EQ.2)GOTO 300

C     READING COEFFICIENTS OF 5TH ORDER SPLINE FOR THE 1ST SEGMENT.
      DO 70 J=1,3
         READ(7,'(3E12.5)',ERR=440)B1(1,J),B4(1,J),B5(1,J)
 70   CONTINUE

C     READING COEFFICIENTS OF 4TH ORDER SPLINE FOR THE INTERMEDIATE
C     SEGMENTS.
      DO 80 ISEG=2,N2
         DO 75 J=1,3
            READ(7,'(4E12.5)',ERR=440)B1(ISEG,J),B2(ISEG,J),B3(ISEG,J),
     *      B4(ISEG,J)
 75      CONTINUE
 80   CONTINUE

C     READING COEFFICIENTS OF 5TH ORDER SPLINE FOR THE LAST SEGMENT.
 90   DO 100 J=1,3
         READ(7,'(5E12.5)',ERR=440)B1(N1,J),B2(N1,J),B3(N1,J),B4(N1,J),
     *   B5(2,J)
 100  CONTINUE

C     READING REAL TIME LIMIT FOR EACH SEGMENT.
      DO 110 K=2,N
         READ(7,'(E12.5)',ERR=440)TOUT(K)
 110  CONTINUE

C     EVALUATING & OUTPUTING DATA FOR THE 1ST SEGMENT.
      CALL WRIST(W(1,1))
      CALL TIMST(0)
 120  CALL TIMRR(0,TMSEC)
      CALL M2ISQQ(TMSEC,T)
      IF(T.GT.TOUT(2))GOTO 140
      T3=T**3
      T4=T*T3
      DO 130 J=1,3
         IDATA(J)=NINT(B1(1,J)+B4(1,J)*T3+B5(1,J)*T4)
 130  CONTINUE
      CALL POSITN(IDATA)
      GOTO 120

C     EVALUATING & OUTPUTING DATA FOR THE INTERMEDIATE SEGMENTS.
 140  DO 170 K=2,N2
```

```
         CALL ESC(ISTOP)
         IF(ISTOP.EQ.1)THEN
            PAUSE
         ENDIF
         IF(NFAST)THEN
            CALL WRIST(W(1,K))
         ENDIF
         CALL TIMST(0)
 150     CALL TIMRR(0,TMSEC)
         CALL M2ISQQ(TMSEC,T)
         IF(T.GT.TOUT(K+1))GOTO 170
         T2=T*T
         T3=T2*T
         DO 160 J=1,3
            IDATA(J)=NINT(B1(K,J)+B2(K,J)*T+B3(K,J)*T2+B4(K,J)*T3)
 160     CONTINUE
         CALL POSITN(IDATA)
         GOTO 150
 170   CONTINUE

C      EVALUATING & OUTPUTING DATA FOR THE LAST SEGMENT.
       IF(NFAST)THEN
          CALL WRIST(W(1,N1))
       ENDIF
       CALL TIMST(0)
 180   CALL TIMRR(0,TMSEC)
       CALL M2ISQQ(TMSEC,T)
       IF(T.GT.TOUT(N))GOTO 350
       T2=T*T
       T3=T2*T
       T4=T3*T
       DO 190 J=1,3
          IDATA(J)=NINT(B1(N1,J)+B2(N1,J)*T+B3(N1,J)*T2+B4(N1,J)*T3+
      *                 B5(2,J)*T4)
 190   CONTINUE
       CALL POSITN(IDATA)
       GOTO 180

C      READING COEFFICIENTS OF 6TH ORDER SPLINE FOR 2-POINT SECTION.
 300   DO 310 J=1,3
          READ(7,'(4E12.5)',ERR=440)B1(1,J),B4(1,J),B5(1,J),B6(1,J)
 310   CONTINUE
       READ(7,'(E12.5)',ERR=440)TOUT(2)

C      EVALUATING & OUTPUTING DATA FOR THE 2-POINT SECTION.
       IF(NFAST)THEN
          CALL WRIST(W(1,1))
       ENDIF
       CALL TIMST(0)
 320   CALL TIMRR(0,TMSEC)
       CALL M2ISQQ(TMSEC,T)
       IF(T.GT.TOUT(2))GOTO 350
       T3=T**3
       T4=T3*T
       T5=T4*T
```



```
       DO 330 J=1,3
          IDATA(J)=NINT(B1(1,J)+B4(1,J)*T3+B5(1,J)*T4+B6(1,J)*T5)
 330   CONTINUE
       CALL POSITN(IDATA)
       GOTO 320

C      READING & PREPARING GRIPPER STATUS FOR NEXT SECTION.
 350   READ(7,'(I10)',END=360,ERR=440)IGPOT

C      CLOSING/OPENING GRIPPER IF THE GRIPPER IS OPENED/CLOSED.
       IF(IGSTAT.EQ.1)THEN
          CALL GCLOSE
          IGSTAT=0
       ELSE
          CALL GOPEN(IGPOT)
          IGSTAT=1
       ENDIF

       GOTO 60

C      REPEATING THE PROCESS IF REQUIRED.
 360   IF(MTIMES.EQ.NTIMES)GOTO 600
       MTIMES=MTIMES+1
C      REWINDING THE FILE TO APPROPRIATE POSITION TO REPEAT THE PROCESS.
       REWIND(7)
       READ(7,'(A)')FNAME
       READ(7,'(2L5)')FAST,CYCLIC
       READ(7,'(5I10)')(IDATA(J),J=1,5)
       GOTO 58

C      DISPLAYING ERROR MESSAGES IF ANY.
 400   WRITE(*,410)
 410   FORMAT(//,' NO BUFFER.JOB FILE CREATED.',//)
       GOTO 610
 420   WRITE(*,430)
 430   FORMAT(//,' NO DATA IN FILE:-BUFFER.JOB .',//)
       GOTO 610
 440   WRITE(*,450)
 450   FORMAT(//,' DATA IN BUFFER.JOB NOT IN APPROPRIATE FORMAT.',//)
       GOTO 610

C      CLOSING THE GRIPPER IF IT IS OPEN BEFORE EXITING THE PROGRAM.
 600   CALL ADIN(5,IPOT5)
       IF(IPOT5.GT.0)THEN
          CALL GCLOSE
       ENDIF

 610   CONTINUE

       END


C      THIS SUBROUTINE IS TO CLOSE THE GRIPPER.
       SUBROUTINE GCLOSE
```

```
      CALL DIGOUT(2,5)
 50   CALL ADIN(5,IPOT5)
      IF(IPOT5.GT.0)GOTO 50
      CALL DIGOUT(0,5)
      RETURN
      END


C     THIS SUBROUTINE IS TO OPEN THE GRIPPER.
      SUBROUTINE GOPEN(IGPOT)

      CALL DIGOUT(1,5)
 50   CALL ADIN(5,IPOT5)
      IF(IGPOT.GT.IPOT5)GOTO 50
      CALL DIGOUT(0,5)
      RETURN
      END
```



```
$STORAGE:2

      PROGRAM SETFLAGS

      CHARACTER FLAG*11

C     DISPLAYING THE OPTIONS
 30   WRITE(*,40)
 40   FORMAT(//,
     *' PLEASE CHOOSE ONE OF THE FOLLOWING OPTIONS TO CONTINUE:-',
     *//,5X,' 1 = REPEAT THE JOB JUST TAUGHT.',
     *//,5X,' 2 = START THE ROBOT AGAIN WITH ALIGNMENT PROCEDURE.',
     *//,5X,' 3 = TEACH THE ROBOT A NEW JOB.',
     *//,5X,' 4 = ASK THE ROBOT TO DO AN OLD JOB.',
     *//,5X,' 5 = PUT THE ROBOT TO SLEEP MODE.',
     *//,5X,' 6 = RETURN TO THE DOS SYSTEM.',//)
      READ(*,*,ERR=50)IFLAG
      IF((IFLAG.LE.0).OR.(IFLAG.GT.6))GOTO 50
      GOTO(100,200,300,400,500,600)IFLAG

 50   WRITE(*,60)
 60   FORMAT(//,' ONLY NUMBER FROM 1 TO 6 IS ACCEPTED.',
     */,' PLEASE TRY AGAIN.',//)
      GOTO 30

C     SETTING APPROPRIATE FLAG.
 100  FLAG='FLAG1.SET'
      GOTO 550
 200  FLAG='FLAG2.SET'
      GOTO 550
 300  FLAG='FLAG3.SET'
      GOTO 550
 400  FLAG='FLAG4.SET'
      GOTO 550
 500  FLAG='FLAG5.SET'

 550  OPEN(10,FILE=FLAG,STATUS='NEW')
      ENDFILE(10)
      CLOSE(10)

 600  CONTINUE

      END
```

```
$STORAGE:2
$NOFLOATCALLS

      PROGRAM SLEEP

      IMPLICIT INTEGER(A-Z)

C     INITIALIZING THE LAB-PACK SUBROUTINES.
      CALL INIT
      CALL INTROFF

      WRITE(*,'(//////////)')
      WRITE(*,50)
 50   FORMAT(//,17X,' ROBOT IS NOW SLEEPING.',
     *//,15X,' PLEASE DO NOT DISTURB HIM.',//,
     */,' [WARNING: DO NOT SWITCH OFF THE POWER SUPPLY OF THE ROBOT.]',
     *//,10X,' PRESS ESC-BUTTON IF YOU WANT TO WAKE HIM UP.',
     *//////)

C     DETECTING THE ESC-BUTTON.
 60   CALL ESC(PRESSED)
      IF(PRESSED.EQ.1)GOTO 80

C     DETECTING THE POWER SUPPLY OF THE ROBOT.
      CALL DISTAT(0,7,POWER,0)
      IF(POWER.EQ.1)GOTO 60

      OPEN(10,FILE='FLAG6.SET',STATUS='NEW')
      ENDFILE(10)
      CLOSE(10)

      WRITE(*,70)
 70   FORMAT(//////////,20X,' POWER IS CUT !',
     *//,17X,' ROBOT DE-ENERGIZED.',//////////)

      CALL PAUSE(10000)
      GOTO 110

 80   WRITE(*,90)
 90   FORMAT(//////////,22X,' ROBOT IS READY.',//////////)
 100  CALL PAUSE(5000)

 110  CONTINUE

      END
```



```
REM ************************
REM THE TASROBOT BATCH FILE
REM ************************

ECHO OFF
CLS
BREAK ON
ECHO THE ROBOT IS NOW BEING ENERGIZED.
IF EXIST ALIGN.EXE GOTO PASS1
ECHO PROGRAM ALIGN.EXE NOT FOUND.
GOTO ERROR

:PASS1
IF EXIST OPFILE.EXE GOTO PASS2
ECHO PROGRAM OPFILE.EXE NOT FOUND.
GOTO ERROR

:PASS2
IF EXIST LEARN.EXE GOTO PASS3
ECHO PROGRAM LEARN.EXE NOT FOUND.
GOTO ERROR

:PASS3
IF EXIST TRIM.EXE GOTO PASS4
ECHO PROGRAM TRIM.EXE NOT FOUND.
GOTO ERROR

:PASS4
IF EXIST SMOOTH.EXE GOTO PASS5
ECHO PROGRAM SMOOTH.EXE NOT FOUND.
GOTO ERROR

:PASS5
IF EXIST GO.EXE GOTO PASS6
ECHO PROGRAM GO.EXE NOT FOUND.
GOTO ERROR

:PASS6
IF EXIST SETFLAGS.EXE GOTO PASS7
ECHO PROGRAM SETFLAGS.EXE NOT FOUND.
GOTO ERROR

:PASS7
IF EXIST SLEEP.EXE GOTO PASS8
ECHO PROGRAM SLEEP.EXE NOT FOUND.

:ERROR
ECHO ROBOT CANNOT BE INITIALIZED.
GOTO DOS

:PASS8
IF NOT EXIST *.SET GOTO START
DEL *.SET

:START
```

```
ALIGN

:TEACH
CLS
OPFILE
CLS
ECHO TEACH PROCESS IS BEING INITIALIZED.
LEARN
TRIM

:DO
CLS
ECHO                    DATA IS BEING FUTHER PROCESSED.
SMOOTH
PAUSE

:REPEAT
CLS
GO

:SETFLAGS
CLS
SETFLAGS
IF NOT EXIST FLAG1.SET GOTO FLAG2
DEL FLAG1.SET
GOTO REPEAT

:FLAG2
IF NOT EXIST FLAG2.SET GOTO FLAG3
DEL NAME.JOB
DEL FLAG2.SET
GOTO START

:FLAG3
IF NOT EXIST FLAG3.SET GOTO FLAG4
DEL NAME.JOB
DEL FLAG3.SET
GOTO TEACH

:FLAG4
IF NOT EXIST FLAG4.SET GOTO FLAG5
DEL NAME.JOB
DEL FLAG4.SET
GOTO DO

:FLAG5
IF NOT EXIST FLAG5.SET GOTO DOS
DEL FLAG5.SET
CLS
SLEEP
GOTO SETFLAGSS

:DOS
DEL NAME.JOB
CLS
```



```
ECHO RETURN TO DOS
```

APPENDIX C

CONVERSION FROM DISCRETE-TIME TRANSFER FUNCTION

TO CONTINUOUS-TIME FOR A THIRD-ORDER SYSTEM

APPENDIX C

CONVERSION FROM DISCRETE-TIME TRANSFER FUNCTION TO CONTINUOUS-TIME FOR A THIRD-ORDER SYSTEM

The discrete-time transfer function is in the form:

$$G(z) = \frac{B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3}}{1 - A_1 z^{-1} - A_2 z^{-2} - A_3 z^{-3}} \qquad \text{(C-1)}$$

For the rest of this Appendix, T will be used to denote the sampling period.

CASE 1

The denominator polynomial can be factorized into three real roots c_1, c_2 and c_3, where c_1, c_2 or c_3 does not equal to unity. Then, equation (C-1) can be expressed as:

$$G(z) = (1-z^{-1})\left[\frac{F_0 z}{z-1} + \frac{F_1 z}{z-c_1} + \frac{F_2 z}{z-c_2} + \frac{F_3 z}{z-c_3}\right] \qquad \text{(C-2)}$$

where
$$F_0 = \frac{B_1 + B_2 + B_3}{(1-c_1)(1-c_2)(1-c_3)}$$

$$F_1 = \frac{B_1 c_1^2 + B_2 c_1 + B_3}{(c_1-1)(c_1-c_2)(c_1-c_3)}$$

$$F_2 = \frac{B_1 c_2^2 + B_2 c_2 + B_3}{(c_2-1)(c_2-c_1)(c_2-c_3)} \qquad \text{(C-3)}$$

$$F_3 = \frac{B_1 c_3^2 + B_2 c_3 + B_3}{(c_3-1)(c_3-c_2)(c_3-c_1)}$$

Taking inverse Z-transform on equation (C-2) gives:

$$G(s) = \frac{1-e^{-sT}}{s}\left[\frac{b_0 s^2 + b_1 s + b_0}{(s-a_1)(s-a_2)(s-a_3)}\right] \qquad \text{(C-4)}$$

where $b_0 = a_1 a_2 a_3 F_0$

$$b_1 = (a_1 a_2 + a_2 a_3 + a_3 a_1) F_0 + a_2 a_3 F_1 + a_1 a_3 F_2 + a_1 a_2 F_3$$

$$b_2 = -(a_1 + a_2 + a_3) F_0 - (a_2 + a_3) F_1 - (a_1 + a_3) F_2 - (a_1 + a_2) F_3$$

$$a_1 = -\ln[c_1]/T \qquad \text{(C-5)}$$

$$a_2 = -\ln[c_2]/T$$

$$a_3 = -\ln[c_3]/T$$

CASE 2

This is a special case of CASE 1 when one of the roots of the denorminator polynomial is unity. This occurs when the identifying system contains an integrating element. Without loss of generality, the root c_1 will be assumed to be unity. Therefore, the discrete-time transfer function can be factorized as:

$$G(z) = (1-z^{-1}) \left[\frac{F_0 z}{z-1} + \frac{F_1 z}{(z-1)^2} + \frac{F_2 z}{z-c_2} + \frac{F_3 z}{z-c_3} \right] \qquad \text{(C-6)}$$

where $F_0 = -F_2 - F_3$

$$F_1 = \frac{B_1 + B_2 + B_3}{(1-c_2)(1-c_3)}$$

$$F_2 = \frac{B_1 c_2^2 + B_2 c_2 + B_3}{(c_2-1)^2 (c_2-c_3)} \qquad \text{(C-7)}$$

$$F_3 = \frac{B_1 c_3^2 + B_2 c_3 + B_3}{(c_3-1)^2 (c_3-c_2)}$$

Taking inverse Z-transform on equation (C-6) gives:

$$G(s) = \frac{1-e^{-sT}}{s} \left[\frac{b_0 s^2 + b_1 s + b_0}{s(s-a_2)(s-a_3)} \right] \qquad \text{(C-8)}$$

where $b_0 = a_2 a_3 F_1 / T$

$b_1 = (a_2 + a_3) F_1 / T - a_2 a_3 F_2 - a_2 a_3 F_3$

$b_2 = F_1 / T - a_3 F_2 - a_2 F_3$ (C-9)

$a_2 = -\ln[c_2] / T$

$a_3 = -\ln[c_3] / T$

CASE 3

If two of the three roots of the denorminator polynomial are imaginary pairs, the denorminator polynomial of equation (C-1) can thus be expressed in the form:

$$1 - A_1 z^{-1} - A_2 z^{-2} - A_3 z^{-3} = (1 - C_1 z^{-1} - C_2 z^{-2})(1 - C_3 z^{-1}) \quad \text{(C-10)}$$

Using partial technique, equation (C-1) can be factorized as:

$$G(z) = (1 - z^{-1}) \left[\frac{F_0 z}{z-1} + \frac{F_1 z}{(z - C_3)} + \frac{F_2 z + F_3 z}{z^2 - C_1 z + C_3} \right] \quad \text{(C-11)}$$

where $F_0 = \dfrac{B_1 + B_2 + B_3}{(1 - C_3)(1 - C_1 + C_2)}$

$F_1 = \dfrac{B_1 C_3^2 + B_2 C_3 + B_3}{(C_3 - 1)(C_3^2 - C_1 C_3 + C_2)}$ (C-12)

$F_2 = -F_0 - F_1$

$F_3 = \dfrac{C_2 C_3 F_0 + C_2 F_1}{C_3}$

Taking inverse Z-transform on equation (C-11) gives:

$$G(s) = \frac{1 - e^{-sT}}{s} \left[\frac{b_0 s^2 + b_1 s + b_0}{(s - a_3)(s^2 + 2as + a^2 + w^2)} \right] \quad \text{(C-13)}$$

where $b_0 = a_3(a^2+w^2)F_0$

$$b_1 = (a^2+w^2+2aa_3)F_0 + (a^2+w^2)F_1 + aa_3F_2 + a_3wK$$

$$b_2 = (2a+a_3)F_0 + 2aF_1 + (a+a_3)F_2 + wK$$

$$a_3 = -\ln[C_3]/T \qquad \text{(C-14)}$$

$$a = -\ln[C_2]/2T$$

$$w = \frac{1}{T}\arccos\left[\frac{C_1}{2\sqrt{C_2}}\right]$$

$$K = \frac{F_3+F_2e^{-aT}\cos wT}{e^{-aT}\sin wT}$$

CASE 4

Again, this is a special case of CASE 3 when the real root of the denominator polynomial is unity, i.e. $C_3=1$. Hence, equation (C-1) can be expressed as:

$$G(z) = (1-z^{-1})\left[\frac{F_0z}{z-1} + \frac{F_1z}{(z-1)^2} + \frac{F_2z+F_3z}{z^2-C_1z+C_2}\right] \qquad \text{(C-15)}$$

where $F_0 = -F_2$

$$F_1 = \frac{B_1+B_2+B_3}{1-C_1+C_2}$$

$$F_2 = \frac{B_1(C_1-2C_2)+B_2(1-C_2)+B_3(2-C_1)}{(1-C_1+C_2)^2} \qquad \text{(C-16)}$$

$$F_3 = \frac{B_1(C_2-1)C_2+B_2(C_1-2)C_2+B_3(1+C_1^2-2C_1-C_2)}{(1-C_1+C_2)^2}$$

Taking inverse Z-transform on equation (C-15) gives:

$$G(s) = \frac{1-e^{-sT}}{s}\left[\frac{b_0s^2+b_1s+b_0}{s(s^2+2as+a^2+w^2)}\right] \qquad \text{(C-17)}$$

where $b_0 = (a^2+w^2)F_1/T$

$b_1 = 2aF_1/T - (a^2+w^2)F_2$

$b_2 = F_1/T - aF_2 + wK$

$a = -\ln[C_2]/2T$ (C-18)

$$w = \frac{1}{T}\arccos\left[\frac{C_1}{2\sqrt{C_2}}\right]$$

$$K = \frac{F_3+F_2 e^{-aT}\cos wT}{e^{-aT}\sin wT}$$

Therefore, by using the results of these four cases, almost any discrete-time transfer function can be converted into its corresponding continuous-time transfer function, provided that the discretized system is approximated by a zero-order-hold data extrapolator, which is the most commonly encountered case, since discrete signals, in many cases, are sent to analog systems through DACs which can be approximated by a zero-order-hold extrapolator.

APPENDIX D

SYSTEM IDENTIFICATION PROGRAMS

```
C     ****************************************************************
C     MAIN PROGRAM: THIS IS THE FIRST PROGRAM OF THE PROGRAM SERIES
C                   WRITTEN FOR SYSTEM IDENTIFICATION.
C                   THIS PROGRAM IS TO GENERATE A PSEUDO-RANDOM
C                   BINARY SEQUENCE EQUIVALENT TO ONE GENERATED
C                   USING 12-STAGE SHIFT REGISTER.
C                   THE TWO STATES OF THE BINARY SEQUENCE ARE
C                   REPRESENTED BY VOLTAGE LEVELS WHICH CAN THAN
C                   BE OUTPUT TO A 12-BIT D/A CONVERTER.

C              BY: LAM H.Y.R.

C            DATE: MAY,1986.

C     ****************************************************************

$STORAGE:2
      PROGRAM SYSID1
      INTEGER PRBS
      LOGICAL BIT(12),TEMP(3)

C     REQUESTING PRBS VOLTAGE LEVEL EQUIVALENT.
 30   WRITE(*,40)
 40   FORMAT(//,' PLEASE INPUT THE VOLTAGE LEVEL, V1, EQUIVALENT',
     */,' TO THE TWO STATES OF THE BINARY SEQUENCE, SUCH THAT:-',
     *//,'               V1 IS EQUIVALENT TO 1, AND',
     */,'              -V1 IS EQUIVALENT TO 0.',
     *//,' [NOTE: V1 MUST BE POSITIVE & LESS THAN OR EQUAL TO 10.]',//)
      READ(*,*)V1
      IF((V1.GT.0).AND.(V1.LE.10))GOTO 45
      GOTO 30

C     CONVERTING THE VOLTAGE LEVEL INTO EQUIVALENT D/A DATA.
 45   IVHIGH=NINT(2047*V1/10)
      IVLOW=NINT(-2048*V1/10)

C     RESETTING THE REGISTERS.
      DO 50 I=1,12
         BIT(I)=.TRUE.
 50   CONTINUE

C     OPENING FILES TO STORE PRBS DATA & PARAMETERS.
      OPEN(5,FILE='PRBS.DAT',STATUS='NEW')
      OPEN(6,FILE='SYSID.DAT',STATUS='NEW')
      WRITE(6,'(F10.5)')V1

C     GENERATING THE PRBS DATA.
      DO 150 K=1,4095
         PRBS=IVLOW
         IF(BIT(12))PRBS=IVHIGH
         WRITE(5,'(I10)')PRBS
         TEMP(1)=.FALSE.
         IF(BIT(2).NEQV.BIT(10))TEMP(1)=.TRUE.
         TEMP(2)=.FALSE.
         IF(BIT(11).NEQV.BIT(12))TEMP(2)=.TRUE.
```



```
         TEMP(3)=.FALSE.
         IF(TEMP(1).NEQV.TEMP(2))TEMP(3)=.TRUE.
         DO 100 I=1,11
            BIT(13-I)=BIT(12-I)
100      CONTINUE
         BIT(1)=TEMP(3)
150   CONTINUE

      CLOSE(5)
      CLOSE(6)

      WRITE(*,160)
160   FORMAT(//,' PRBS IS STORED IN FILE :- PRBS.DAT .',
     *//,' PLEASE USE THE PROGRAM-SYSID2-OF THE SAME SERIES',
     */,' (IF AVAILABLE) FOR OUTPUTING THE PRBS DATA TO ',
     */,' THE SYSTEM AND AT THE SAME TIME SAMPLING THE SYSTEM',
     */,' OUTPUT TO CONTINUE THE SYSTEM IDENTIFICATION PROCESS.',//)

      END
```

```
C     ***************************************************************
C      MAIN PROGRAM: THIS IS THE SECOND PROGRAM OF THE PROGRAM SERIES
C                    WRITTEN FOR SYSTEM IDENTIFICATION.
C                    THIS PROGRAM IS TO OUTPUT A  PRBS DATA TO A
C                    SYSTEM AND AT THE SAME TIME SAMPLE THE OUTPUT
C                    OF THE ENERGIZED SYSTEM IN ORDER THAT THE
C                    PARAMETERS OF THE SYSTEM CAN BE IDENTIFIED
C                    USING RECURSIVE LEAST SQUARE METHOD.
C                    THE PROGRAM CAN FUNCTION UP TO 12-BIT PRBS DATA.
C                    [NOTE: THE  PRBS DATA MUST BE AVAILABLE
C                           IN THE FILE:-'PRBS.DAT' PRIOR TO THE
C                           EXECUTION OF THE PROGRAM. ]
C
C        WRITTEN BY: LAM H.Y.R.
C
C              DATE: AUGUST,1986
C     ***************************************************************


$STORAGE:2
      PROGRAM SYSID2
      IMPLICIT INTEGER(A-Z)
      REAL DUMMY1,DUMMY2
      DIMENSION PRBS(4095),Y(4095),CHANEL(2),X(4095)
      LOGICAL OLDFILE
      CHARACTER KEY*1

C     INITIALIZING THE LAB-PACK SUBROUTINES.
      CALL INIT
      CALL INTROFF

C     CHECKING FILES TO SEE IF EXISTS.
      INQUIRE(FILE='PRBS.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 300
      OPEN(11,FILE='PRBS.DAT')
      OPEN(22,FILE='SYSIO.DAT',STATUS='NEW')
      INQUIRE(FILE='SYSID.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 370
      OPEN(33,FILE='SYSID.DAT')
      READ(33,'(F10.5)',ERR=370)DUMMY1

      N=4095

C     READING DATA FROM FILE-'PRBS.DAT'.
      DO 50 I=1,N
         READ(11,'(I10)',ERR=320)PRBS(I)
         X(I)=PRBS(I)
 50   CONTINUE

C     REQESTING PARAMETERS FROM USER.
 70   WRITE(*,80)
 80   FORMAT(//,' PLEASE INPUT THE D/A CHANNEL NO. TO WHICH',
     */,' THE  PRBS ARE TO BE OUTPUT.',
     *//,' [NOTE: ONLY 0 OR 1 IS AVAILABLE.]',//)
      READ(*,*)DACHAN
```



```
      IF((DACHAN.NE.1).AND.(DACHAN.NE.0))GOTO 70

C     SETTING THE INITIAL CONDITION TO ZERO.
      INITL=0
      CALL DAOUT(DACHAN,INITL)

 90   WRITE(*,100)
 100  FORMAT(//,' PLEASE INPUT THE A/D CHANNEL NO. FROM WHICH',
     */,' SYSTEM OUTPUT IS TO BE SAMPLED.',
     *//,' [NOTE: ONLY NO. FROM 0 TO 15 IS AVAILABLE.]',//)
      READ(*,*)ADCHAN
      IF((ADCHAN.GT.15).OR.(ADCHAN.LT.0))GOTO 90
 110  WRITE(*,120)
 120  FORMAT(//,' PLEASE INPUT CLOCK PERIOD & THE NO. OF CLOCKS',
     */,' BETWEEN WHICH SAMPLING TAKES PLACE.',
     *//,' [NOTE: CLOCK PERIOD IS IN THE ORDER OF MICRO-SECOND.',
     */,7X,' THE MIN. CLOCK PERIOD IS 1,000 MICRO-SEC.',
     */,7X,' THE MAX. CLOCK PERIOD IS 32,767 MICRO-SEC.',
     */,7X,' THE MAX. NO. OF CLOCKS BETWEEN SAMPLES IS 32,767.',
     */,7X,' BOTH DATA ARE TO BE INTEGERS.          ]',//)
      READ(*,*,ERR=110)CLOCK,INTVAL
      IF((CLOCK.LT.1000).OR.(CLOCK.GT.32767))GOTO 110
      IF((INTVAL.LE.0).OR.(INTVAL.GT.32767))GOTO 110

      WRITE(*,130)
 130  FORMAT(//,' DO YOU WANT TO IMPOSE A SAFETY LIMIT ON THE SYSTEM',
     */,' OUTPUT, SO THAT I.D. PROCESS WILL BE TERMINATED WHEN',
     */,' SAMPLED OUTPUT EXCEEDS THE SPECIFIED LIMIT ? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.NE.'N')THEN
 135     WRITE(*,137)
 137     FORMAT(//,' PLEASE SPECIFY THE LIMIT.',
     *   //,' [NOTE:  0 < LIMIT <= 2047 ]',//)
         READ(*,*,ERR=135)LIMIT
         IF(LIMIT.LE.0)GOTO 135
      ELSE
         LIMIT=3000
      ENDIF

C     RECORDING CLOCK PERIOD & SAMPLING INTERVAL USED & DAY & TIME.
      WRITE(33,'(2I20)')CLOCK,INTVAL
      CALL QTIME(IHR,IMIN,ISEC,IHUND)
      CALL QDATE(IYR,IMTH,IDAY)
      WRITE(33,'(5I10)')IHR,IMIN,IYR,IMTH,IDAY

C     SETTING INTERRUPT CLOCK PERIOD.
      CALL STTIMEB(CLOCK)

C     SETTING UP A/D CHANNEL.
      CHANEL(1)=ADCHAN
      CHANEL(2)=999


C     OUTPUTING THE  PRBS & SAMPLING THE SYSTEM OUTPUT.
      CALL DASWST(DACHAN,INTVAL,X,N,1,1,0)
```

```
      CALL ADSWST(CHANEL,INTVAL,Y,N,0)

C     ENABLING INTERRUPT OF THE LAB-MASTER INTERFACING UNIT.
      CALL INTRON

C     WAITING UNTIL SPECIFIED NO. OF SAMPLES OBTAINED; OR SPECIFIED
C     LIMIT ON OUPUT VALUE EXCEEDED.
 140  CALL ADSWAB(0,NBUFER)
      OUTPUT=ABS(Y(NBUFER))
      IF(OUTPUT.GT.LIMIT)GOTO 143
      IF(NBUFER.NE.N)GOTO 140

C     DISABLE INTERRUPT OF THE LAB-MASTER INTERFACING UNIT.
 143  CALL DAOUT(DACHAN,INITL)
      CALL INTROFF

C     DETECTING IF SPECIFIED OUTPUT LIMIT EXCEEDED.
      IF(NBUFER.EQ.N)GOTO 147
      CALL ADSWAB(1,IDUMMY)
      WRITE(*,145)NBUFER
 145  FORMAT(//,' IDENTIFICATION PROCESS TERMINATED DUE TO THE',
     */,' SPECIFIED OUTPUT LIMIT EXCEEDED.',
     *//,' NO. OF SAMPLE RECORDED = ',I5,
     *//,' DO YOU WANT TO TRY AGAIN ? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.EQ.'N')GOTO 146
      WRITE(*,200)
 200  FORMAT(//,' DO YOU WANT TO CHANGE THE STARTING POINT OF PRBS DATA
     *? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.EQ.'N')GOTO 240
 205  WRITE(*,210)
 210  FORMAT(//,' INPUT NO. OF DELAY OF PRBS DATA.',
     *//,' [NOTE: 1 <= NO. OF DELAY <= 4094  .]',//)
      READ(*,*)NSTART
      IF((NSTART.LT.1).OR.(NSTART.GT.4094))GOTO 205
      DO 220 I=1,4095-NSTART
         X(I)=PRBS(I+NSTART)
 220  CONTINUE
      DO 230 I=1,NSTART
         X(4095-NSTART+I)=PRBS(I)
 230  CONTINUE
 240  REWIND(33)
      READ(33,'(F10.5)')DUMMY1
      GOTO 110

 146  N=NBUFER

C     STORING THE  PRBS INPUT & THE SAMPLED SYSTEM OUTPUT DATA
C     TO FILE:-'SYSIO.DAT'.
 147  DO 150 J=1,N
         Y(J)=Y(J)+7
         WRITE(22,'(2I10)')X(J),Y(J)
 150  CONTINUE
```



```
      CLOSE(11)
      CLOSE(22)
      CLOSE(33)

      WRITE(*,160)
 160  FORMAT(//,' SYSTEM INPUT & OUTPUT DATA ARE STORED IN FILE:-',
     */,' SYSIO.DAT .',
     *//,' PLEASE USE THE PROGRAM-SYSID3-OF THE SAME SERIES',
     */,' (IF AVAILABLE) FOR ESTIMATING THE SYSTEM PARAMETERS',
     */,' USING RECURSIVE LEAST SQUARE METHOD.',//)
      GOTO 500

C     DISPLAYING ERROR SIGNALS (IF ANY).
 300  WRITE(*,310)
 310  FORMAT(//,' NO  PRBS DATA AVAILABLE.',//)
      GOTO 350
 320  WRITE(*,330)
 330  FORMAT(//,' UNACCEPTABLE DATA FORMAT IN FILE PRBS.DAT .',//)

 350  WRITE(*,360)
 360  FORMAT(//,' PLEASE CHECK DATA FILE:-PRBS.DAT, OR',
     */,' RE-RUN THE PROGRAM-SYSID1-TO REGENERATE',
     */,'  PRBS DATA.',//)
      GOTO 500

 370  WRITE(*,380)
 380  FORMAT(//,' CONDITIONS FOR GENERATING SYSTEM INPUT DATA LOST.',
     *//,' PLEASE RE-START THE PROGRAM SERIES BY USING THE',
     */,' PROGRAM-SYSID1 .',//)
      GOTO 500


 500  CONTINUE

      END
```

```
$NOFLOATCALLS
$STORAGE:2

C     ***************************************************************
C     MAIN PROGRAM: THIS IS THE THIRD PROGRAM OF THE IDENTIFICATION
C                   PROGRAM SERIES.
C                   THIS PROGRAM IS TO ESTIMATE THE COEFFICIENTS OF
C                   A THIRD ORDER TRANSFER FUNCTION WITH AN
C                   INTEGRATING ELEMENT BY MINIMIZING THE SUM OF
C                   SQUARES OF THE EQUATION ERROR.
C
C       WRITTEN BY: R.H.Y.LAM
C
C             DATE: MAY,1986
C     ***************************************************************
      PROGRAM SYSID3

      REAL X(4095),Y(4095),TEMPP(6,6),P(6,6),THETA(6,1),H(1,6)
      REAL K(6,1),TRANSH(6,1),TEMP1(6,1),TEMP2(1,1),TEMP3(1,6)
      REAL TEMP4(6,6),TEMP5(6,6),TEMP6(1,1),TEMP7(6,1),INVDEL
      REAL C(4095)
      INTEGER DATAX,DATAY,CLOCK
      CHARACTER SNAME*50,KEY*1
      LOGICAL OLDFILE

C     CHECKING TO SEE IF DATA FILES AVALIABLE.
      INQUIRE(FILE='SYSIO.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 220
      INQUIRE(FILE='SYSID.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 280

C     SETTING UP FILES FOR READING & STROING DATA.
      OPEN(1,FILE='PRN')
      OPEN(44,FILE='SYSID.DAT')
      OPEN(55,FILE='SYSIO.DAT')
      OPEN(66,FILE='SYSOUTS.DAT',STATUS='NEW')
      OPEN(77,FILE='SYSINFO.DAT',STATUS='NEW')
      OPEN(88,FILE='SYSERRS.DAT',STATUS='NEW')

C     READING CONDITIONS FOR DATA GENERATION.
      READ(44,'(F10.5)',ERR=280)V1
      READ(44,'(2I20)',ERR=320)CLOCK,INTVAL
      READ(44,'(5I10)',ERR=320)IHR,IMIN,IYR,IMTH,IDAY

      DO 50 I=1,4096
         READ(55,'(2I10)',END=55,ERR=240)DATAX,DATAY
         X(I)=FLOAT(DATAX)
         Y(I)=FLOAT(DATAY)
 50   CONTINUE

 55   N=I-1

C     REQUESTING NECESSARY PARAMETERS FROM THE USER.
 58   WRITE(*,60)
 60   FORMAT(//,' PLEASE INPUT WINDOW FACTOR FOR THE RECURSIVE LEAST',
     */,' SQUARE ALGORITHM.',
     *//,' [NOTE: 0 < WINDOW FACTOR <= 1    ]',//)
      READ(*,*)DELTA
      IF((DELTA.GT.1).OR.(DELTA.LE.0))GOTO 58
      WRITE(*,62)
 62   FORMAT(//,' PLEASE INITIALIZE THE ALGORITHM BY SPECIFYING',
     */,' THE DIAGONAL ELEMENT OF THE INITIAL P MATRIX &',
     */,' THE INITIAL VALUE OF THE SYSTEM PARAMETERS.',//)
      READ(*,*)PDIAGL,THETA0

 63   WRITE(*,65)
 65   FORMAT(//,' PLEASE INPUT A NAME FOR THE IDENTIFIED SYSTEM.',
     *//,' [WARNING: NOT MORE THAN 50 CHARACTERS ARE ACCEPTED.]',//)
      READ(*,'(A)')SNAME

C     INITIALZING P TO A LARGE DIAGONAL MATRIX.
      CALL UNIT(TEMPP,6)
      CALL SCAMUL(PDIAGL,TEMPP,P,6,6)

C     INITIALIZING THETA TO ARBITRARY COLUMN MATRIX.
      DO 70 J=1,6
         THETA(J,1)=THETA0
 70   CONTINUE

C     INITIALIZING SUMS OF THETA & THETA SQUARE.
      CALL NULL(SUM,6,1)
      CALL NULL(SUMSQ,6,1)

C     SET INITIAL ERROR TO ZERO
      ERROR=0
      C(1)=0
      C(2)=0
      C(3)=0

C     PERFORMING RECURSIVE LEAST SQUARE METHOD TO
C     ESTIMATE THETA MATRIX.
      DO 80 M=4,N
C     SETTING UP H MATRIX.
         H(1,1)=X(M-1)
         H(1,2)=X(M-2)
         H(1,3)=X(M-3)
         H(1,4)=Y(M-1)
         H(1,5)=Y(M-2)
         H(1,6)=Y(M-3)
C     CALCULATING K MATRIX.
         CALL TRANSP(H,TRANSH,1,6)
         CALL MULTI(P,TRANSH,TEMP1,6,6,1)
         CALL MULTI(H,TEMP1,TEMP2,1,6,1)
         CONSTA=1/(DELTA+TEMP2(1,1))
         CALL SCAMUL(CONSTA,TEMP1,K,6,1)
C     CALCULATING P MATRIX.
         CALL MULTI(H,P,TEMP3,1,6,6)
         CALL MULTI(K,TEMP3,TEMP4,6,1,6)
         CALL SUBTR(P,TEMP4,TEMP5,6,6)
         INVDEL=1/DELTA
```

```
         CALL SCAMUL(INVDEL,TEMP5,P,6,6)
C      CALCULATING REFINED THETA MATRIX.
         CALL MULTI(H,THETA,TEMP6,1,6,1)
         CONSTB=Y(M)-TEMP6(1,1)
         CALL SCAMUL(CONSTB,K,TEMP7,6,1)
         CALL ADD(THETA,TEMP7,THETA,6,1)
         C(M)=CONSTB
 80    CONTINUE

C      CALCULATING MODELLED OUTPUT USING THE MEAN OF THE ESTIMATED
C      THETA AND STORING THE ACTUAL & THE MODELLED OUTPUTS TO FILE:-
C      'SYSOUTS.DAT' .
 83    Y(1)=0
       Y(2)=THETA(1,1)*X(1)+THETA(4,1)*Y(1)
       Y(3)=THETA(1,1)*X(2)+THETA(2,1)*X(1)+THETA(4,1)*Y(2)+
      *     THETA(5,1)*Y(1)
       DO 85 J=4,N
          Y(J)=THETA(1,1)*X(J-1)+THETA(2,1)*X(J-2)+THETA(3,1)*X(J-3)+
      *        THETA(4,1)*Y(J-1)+THETA(5,1)*Y(J-2)+THETA(6,1)*Y(J-3)
          IF(ABS(Y(J)).GT.1.0E+30)GOTO 86
 85    CONTINUE
       GOTO 88

 86    WRITE(*,87)
 87    FORMAT(//,' WARNING: SIMULATED SYSTEM UNSTABLE.',
      */,9X,' MODELLED OUTPUT EXCEEDING 1E50.',
      *//,' MODELLING PROCESS TERMINATED.',//)

 88    REWIND 55
       DO 89 I=1,J
          READ(55,'(10X,I10)',ERR=91)DATAY
          YORGN=FLOAT(DATAY)
          OUTERR=YORGN-Y(I)
          DATANO=FLOAT(I)
          WRITE(66,'(3E12.5)',ERR=91)DATANO,YORGN,Y(I)
          WRITE(88,'(3E12.5)',ERR=91)DATANO,C(I),OUTERR
 89    CONTINUE

 91    CLOSE(44)
       CLOSE(55)
       CLOSE(66)
       CLOSE(88)

       WRITE(1,90)SNAME,IDAY,IMTH,IYR,IHR,IMIN,V1,V1,CLOCK,INTVAL,N,
      *DELTA
       WRITE(1,'(A1)')'1'
       WRITE(77,90)SNAME,IDAY,IMTH,IYR,IHR,IMIN,V1,V1,CLOCK,INTVAL,N,
      *DELTA
 90    FORMAT(15X,'***** SYSTEM PARAMETER IDENTIFICATION *****',
      *////,' SYSTEM:  ',A,
      *//,' TIME  ',I2,'-',I2,'-',I4,5X,'(',I2,':',I2,')',
      *//,' INPUT: 12-BIT PSEUDO-RANDOM BINARY SEQUENCES',
      *//,' PRBS VOLTAGE LEVEL:  (+)',F8.5,' OR (-)',F8.5,'  VOLT',
      *//,' CLOCK:  ',I6,' MICRO-SEC.',
      *//,' SAMPLING INTERVAL:  ',I6,' (CLOCKS)',
```



```
      *//,' NO. OF I/O PAIRS:  ',I6,
      *////,' IDENTIFICATION METHOD:  RECURSIVE LEAST SQUARE',
      *//,' WINDOW FACTOR:  ',F5.3,
      *////,38X,'B1*Z(-1)+B2*Z(-2)+B3*Z(-3)',
      */,' MODELLED TRANSFER FUNCTION:  G(Z)=----------------------------
      *-----',
      */,36X,' 1-A1*Z(-1)-A2*Z(-2)-A3*Z(-3)')
       WRITE(1,93)THETA(1,1),THETA(2,1),THETA(3,1),
      *THETA(4,1),THETA(5,1),THETA(6,1)
       WRITE(77,93)THETA(1,1),THETA(2,1),THETA(3,1),
      *THETA(4,1),THETA(5,1),THETA(6,1)
 93    FORMAT(////,' ESTIMATED PARAMETERS VALUES:-',
      *//,10X,'B1=',F15.10,5X,
      *//,10X,'B2=',F15.10,5X,
      *//,10X,'B3=',F15.10,5X,
      *//,10X,'A1=',F15.10,5X,
      *//,10X,'A2=',F15.10,5X,
      *//,10X,'A3=',F15.10,5X)

       CLOSE(1)
       CLOSE(77)

       WRITE(*,95)
 95    FORMAT(//,' ALL THE INFORMATION & RESULTS OF THE IDENTIFICATION',
      */,' PROCESS ARE PRINTED AND ALSO STORED IN FILE:-SYSINFO.DAT .',
      *//)

       WRITE(*,100)
 100   FORMAT(//,' ACTUAL OUTPUT & MODELLED OUTPUT ARE STORED IN',
      */,' FILE:- SYSOUTS.DAT',
      */,' EQUATION ERROR & OUTPUT ERROR ARE STORED IN FILE:-',
      */,'   SYSERRS.DAT',
      *//,' PLEASE USE THE PROGRAM SYSID4 OF THE SAME SERIES TO',
      */,' DISPLAY THE OUTPUTS OR THE ERRORS.',//)

       GOTO 500

C      DISPLAYING ERROR MESSAGE (IF ANY).
 220   WRITE(*,230)
 230   FORMAT(//,' SYSTEM INPUT/OUTPUT DATA NOT AVAILABLE.',//)
       GOTO 260

 240   WRITE(*,250)
 250   FORMAT(//,' INPUT/OUTPUT DATA NOT IN ACCEPTABLE FORMAT.',//)

 260   WRITE(*,270)
 270   FORMAT(//,' PLEASE CHECK DATA FILE:-SYSIO.DAT, OR',
      */,' RE-RUN THE PROGRAM-SYSID3 .',//)
       GOTO 500

 280   WRITE(*,290)
 290   FORMAT(//,' CONDITIONS FOR INPUT DATA GENERATION LOST !',
      *//,' PLEASE RE-START THE IDENTIFICATION PROCESS BY ',
      */,' USING THE PROGRAM-SYSID1 .',//)
       GOTO 500
```

```
 320  WRITE(*,330)
 330  FORMAT(//,'  PRBS DATA CHANGED.',
     */,' PLEASE RE-NEW THE SYSTEM INPUT/OUTPUT BY USING',
     */,' THE PROGRAM-SYSID2 .',//)

 500  CONTINUE

      END


C       **********************************
C       SUBROUTINES FOR MATRIX CALCULATION
C       **********************************

C MATRIX ADDITION
        SUBROUTINE ADD(A,B,C,M,N)
        DIMENSION A(M,N),B(M,N),C(M,N)
        DO 1 I=1,M
        DO 1 J=1,N
 1      C(I,J)=A(I,J)+B(I,J)
        RETURN
        END

C MATRIX SUBTRACTION
        SUBROUTINE SUBTR(A,B,C,M,N)
        DIMENSION A(M,N),B(M,N),C(M,N)
        DO 1 I=1,M
        DO 1 J=1,N
 1      C(I,J)=A(I,J)-B(I,J)
        RETURN
        END

C TRANSPOSE OF A MATRIX
        SUBROUTINE TRANSP(A,B,M,N)
        DIMENSION A(M,N),B(N,M)
        DO 1 I=1,M
        DO 1 J=1,N
 1      B(J,I)=A(I,J)
        RETURN
        END

C SET IDENTITY MATRIX
        SUBROUTINE UNIT(A,M)
        DIMENSION A(M,M)
        DO 1 I=1,M
        DO 1 J=1,M
        A(I,J)=1.0
 1      IF (I.NE.J) A(I,J)=0.0
        RETURN
        END

C MULTIPLICATION OF MATRICES
        SUBROUTINE MULTI(A,B,C,L,M,N)
        DIMENSION A(L,M),B(M,N),C(L,N)
```



```
        DO 1 I=1,L
        DO 1 J=1,N
        C(I,J)=0.0
        DO 1 K=1,M
 1      C(I,J)=C(I,J)+A(I,K)*B(K,J)
        RETURN
        END

C MATRIX MULTIPLICATION BY A SCALAR
        SUBROUTINE SCAMUL(S,A,B,M,N)
        DIMENSION A(M,N),B(M,N)
        DO 1 I=1,M
        DO 1 J=1,N
 1      B(I,J)=S*A(I,J)
        RETURN
        END
```

```
$NOFLOATCALLS
$STORAGE:2

C     ***************************************************************
C     MAIN PROGRAM: THIS IS THE FOURTH PROGRAM OF THE IDENTIFCATION
C                   PROGRAM SERIES.
C                   THIS PROGRAM IS TO DISPLAY THE OUTPUTS OR THE
C                   ERRORS VARIATIONS OF THE IDENTIFIED TRANSFER
C                   FUNCTION.
C
C       WRITTEN BY: R.H.Y.LAM
C
C             DATE: MAY 1986
C     ***************************************************************

      PROGRAM SYSID4

      REAL X(1000),Y1(1000),Y2(1000),XTEMP(100),Y1TEMP(100)
      REAL Y2TEMP(100)
      CHARACTER XTEXT*6,YTEXT*6,KEY*1
      LOGICAL OLDFILE,FRAME,ANALOG,EXPAND,FILEND,FSTSEG

   5  WRITE(*,10)
  10  FORMAT(//,' PLEASE CHOOSE THE FOLLOWING OPTIONS:-',
     */,5X,' 1 = PLOT THE ACTUAL AND THE MODELLED OUTPUTS.',
     */,5X,' 2 = PLOT THE EQUATION ERROR AND THE OUTPUT ERROR.',//)
      READ(*,*)IKEY
      IF((IKEY.NE.1).AND.(IKEY.NE.2))GOTO 5
      GOTO(20,30)IKEY

C     CHECKING THE APPROPRIATE FILE.
  20  INQUIRE(FILE='SYSOUTS.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 500
      OPEN(55,FILE='SYSOUTS.DAT')
      GOTO 35

  30  INQUIRE(FILE='SYSERRS.DAT',EXIST=OLDFILE)
      IF(.NOT.OLDFILE)GOTO 500
      OPEN(55,FILE='SYSERRS.DAT')

C     SETTING FLAGS TO APPROPRIATE CONDITIONS.
  35  EXPAND=.FALSE.
      FILEND=.FALSE.
      FSTSEG=.TRUE.

C     READING & SORTING DATA FROM FILE-SYSOUTS.DAT .
      READ(55,'(3E12.5)',END=540,ERR=520)X(1),Y1(1),Y2(1)
      YMIN=MIN(Y1(1),Y2(1))
      YMAX=MAX(Y1(1),Y2(1))
      REWIND(55)
  40  DO 50 I=1,1000
         READ(55,'(3E12.5)',END=60,ERR=520)X(I),Y1(I),Y2(I)
         YMIN=MIN(YMIN,Y1(I),Y2(I))
         YMAX=MAX(YMAX,Y1(I),Y2(I))
```



```
  50  CONTINUE

      N=1000
      GOTO 65

  60  N=I-1
      FILEND=.TRUE.
      CLOSE(55)
      IF(N.LT.10)GOTO 250
  65  NPT=N

C     FORMING X-AXIS & YAXIS PRARMETERS.
      XXMIN=X(1)-1
      XMIN=XXMIN
      XXMAX=X(N)
      XMAX=XXMAX
      XORG=XXMIN
      NDECX=0
      XLENGH=XXMAX-XXMIN
      IF(XLENGH.LE.10)THEN
         XMARK=1
      ELSEIF(XLENGH.LE.100)THEN
         XMARK=10
      ELSE
         XMARK=100
      ENDIF
      XTEXT='SAMPLE'

      IF(.NOT.FSTSEG)GOTO 75

      WRITE(*,73)
  73  FORMAT(//,' PLEASE SPECIFY TEXT FOR LABELLING Y-AXIS.',
     *//,' [WARNING: NOT MORE THAN 6 CHARACTERS WILL BE ACCEPTED.]',//)
      READ(*,'(A)')YTEXT
  75  WRITE(*,80)YMIN,YMAX
  80  FORMAT(//,' MIN. & MAX. VALUES OF Y ARE: ',F7.1,'  &  ',F7.1,
     *//,' PLEASE SPECIFY MIN. & MAX. VALUES ON Y-AXIS.',//)
      READ(*,*)YYMIN,YYMAX
      IF((YYMIN.LE.YMIN).AND.(YYMAX.GE.YMAX))GOTO 84
      WRITE(*,82)
  82  FORMAT(//,' ERROR: BOUNDARIES ON Y-AXIS TOO SMALL.',
     *//,' PLEASE TRY AGAIN.')
      GOTO 75
  84  WRITE(*,86)
  86  FORMAT(//,' PLEASE SPECIFY MARKING INTERVALS ON Y-AXIS.',//)
      READ(*,*)YMARK
      YINTVL=(YYMAX-YYMIN)/YMARK
      IF(YINTVL.LE.20.0)GOTO 90
      WRITE(*,87)
  87  FORMAT(//,' WARNING: MARKING INTERVALS TO CLOSE TOGETHER.',
     *//,' PLEASE TRY AGAIN.')
      GOTO 84
  90  YORG=0
      IF(YYMIN.GT.0)YORG=YYMIN
      IF(YYMAX.LT.0)YORG=YYMAX
```

```
      NDECY=0
      FRAME=.TRUE.

  100 CALL XYFRAM(XXMIN,XXMAX,YYMIN,YYMAX,XORG,YORG,XMARK,YMARK,XTEXT,
     *YTEXT,NDECX,NDECY,FRAME)

C     PLOTTING THE ACTUAL OUTPUT CURVE WITH NO SYMBOL ADDED.
      NMARK=1
      ISYMBL=-2
      ANALOG=.TRUE.
      IF(EXPAND)THEN
         CALL XYPLOT(XTEMP,Y1TEMP,NPT,NMARK,ISYMBL,ANALOG)
      ELSE
         CALL XYPLOT(X,Y1,NPT,NMARK,ISYMBL,ANALOG)
      ENDIF

C     PLOTTING THE MODELLED OUTPUT CURVE WITH CROSSES ADDED.
      NMARK=20
      ISYMBL=-1
      IF(EXPAND)THEN
         CALL XYPLOT(XTEMP,Y2TEMP,NPT,NMARK,ISYMBL,ANALOG)
      ELSE
         CALL XYPLOT(X,Y2,NPT,NMARK,ISYMBL,ANALOG)
      ENDIF

C     GRAPHICS PAUSE UNTIL ANY KEY DEPRESSED.
  105 CALL XYCLS

      IF(EXPAND)GOTO 115

C     OPTION FOR DISPLAYING THE PLOTS AFTER CLEARING SCREEN.
      WRITE(*,110)
  110 FORMAT(//,' DO YOU WANT TO DISPLAY THE PLOT AGAIN ? (Y/N)',//)
      READ(*,'(A)')KEY
      IF(KEY.EQ.'Y')GOTO 100
      IF(N.LE.200)GOTO 200

C     OPTION FOR DISPLAYING PORTION OF THE PLOT.
  115 IF(EXPAND)THEN
         WRITE(*,120)
  120    FORMAT(//,' DO YOU WANT TO DISPLAY ANOTHER PORTION ? (Y/N)',/)
      ELSE
         WRITE(*,130)
      EXPAND=.FALSE.
  130    FORMAT(//,' DO YOU WANT TO DISPLAY AN EXPANDED PORTION OF',
     *   /,' THE PLOT CONTAINING 100 SAMPLES ? (Y/N)',//)
      ENDIF
      READ(*,'(A)')KEY
      IF(KEY.EQ.'N')GOTO 200

C     PLOTTING AN EXPAND PORTION OF THE PLOT CONTAINING 100 SAMPLES.
      IXLOW=INT(XMIN)
      IXUP=INT(XMAX)-100
  135 WRITE(*,140)IXLOW,IXUP
  140 FORMAT(//,' PLEASE SPECIFY THE STARTING SAMPLE NO.',
```



```
     *//,' [NOTE: ',I5,' <= NO. SPECIFIED <= ',I5,' ]',//)
      READ(*,*,ERR=135)NST
      IF((NST.LT.IXLOW).OR.(NST.GT.IXUP))GOTO 135
      YMIN=MIN(Y1(NST+1),Y2(NST+1))
      YMAX=MAX(Y1(NST+1),Y2(NST+1))
      DO 170 I=1,100
         XTEMP(I)=FLOAT(NST+I)
         Y1TEMP(I)=Y1(NST+I-IXLOW)
         Y2TEMP(I)=Y2(NST+I-IXLOW)
         YMIN=MIN(Y1TEMP(I),Y2TEMP(I),YMIN)
         YMAX=MAX(Y1TEMP(I),Y2TEMP(I),YMAX)
  170 CONTINUE
      EXPAND=.TRUE.
      XXMIN=XTEMP(1)-1
      XXMAX=XTEMP(100)
      XMARK=10
      XORG=XXMIN
      NPT=100
      GOTO 75

C     STARTING THE NEXT SEGMENT OF DATA PLOTTING.
  200 IF(FILEND)GOTO 700
      EXPAND=.FALSE.
      FSTSEG=.FALSE.
      WRITE(*,210)
  210 FORMAT(//,' PRESS ANY KEY TO CONTINUE OR <E> TO EXIT PLOTTING.',
     *//)
      READ(*,'(A)')KEY
      IF(KEY.EQ.'E')GOTO 700
      YMIN=MIN(Y1(N),Y2(N))
      YMAX=MAX(Y1(N),Y2(N))
      GOTO 40

  250 WRITE(*,260)
  260 FORMAT(//,' LESS THAN 10 DATA LEFT.',
     *//,' PLOTTING INHIBITED.',//)
      GOTO 700

C     DISPLAYING ERROR MESSAGES (IF ANY).

  500 WRITE(*,510)
  510 FORMAT(//,' DATA FILE (SYSOUTS.DAT/SYSERRS.DAT) NOT FOUND.',//)
      GOTO 600
  520 WRITE(*,530)
  530 FORMAT(//,' DATA NOT IN APPROPRIATE FORMAT.',//)
      GOTO 600

  540 WRITE(*,550)
  550 FORMAT(//,' NO DATA FOUND IN FILE SYSOUTS.DAT/SYSERRS.DAT.',//)
      GOTO 600

  600 WRITE(*,610)
  610 FORMAT(' PLEASE RE-RUN THE PROGRAM-SYSID3 TO RE-GENERATE',
     */,' DATA TO FILE SYSOUTS.DAT/SYSERRS.DAT.',//)
```

```
700   CLOSE(55)
      END
```