1984

Robotic software for the mini-mover 5 robot arm

Alexander Zelinsky
University of Wollongong

THE UNIVERSITY OF WOLLONGONG

DEPARTMENT OF COMPUTING SCIENCE

ROBOTIC SOFTWARE FOR THE MINI-MOVER 5 ROBOT ARM

Alexander Zelinsky

Department of Computing Science
University of Wollongong

Abstract

The aim of the project is to design and implement software on a MOTOROLA M68000 16 bit microprocessor to control a MINI-MOVER 5 robot arm.

Preprint No 84-6 **April 10, 1984**

P.O. Box 1144, WOLLONGONG, N.S.W. AUSTRALIA
telephone (042)-270-859
telex AA29022

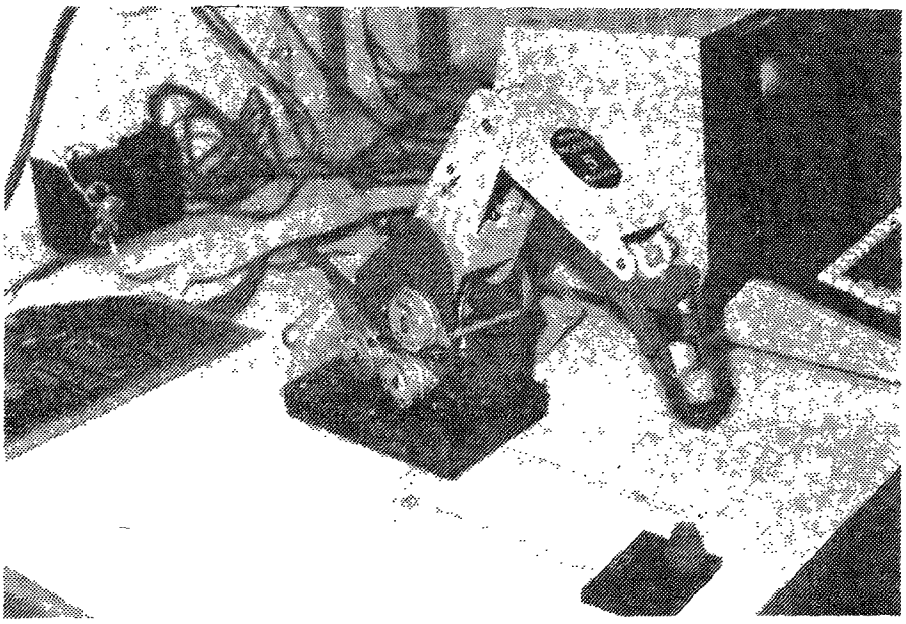
ROBOTIC SOFTWARE FOR THE MINI-MOVER 5 ROBOT ARM

Alexander ZELINSKY

Department of Computing Science

University of Wollongong

29th December 1983



CONTENTS Page

CONTENTS Page

CHAPTER 1
PROJECT COMMENCEMENT

1.1 Project Description

The aim of the project is to design and implement software on a MOTOROLA M68000 16 bit microprocessor to control a MINI-MOVER 5 robot arm.

The software will be designed such that the commands to control the robot are simple and transparent to the user. The commands will be independent and the design will allow extensibility of the software by the users.

The software will become a tool, which will allow users to write application programs to control the robot in performing tasks. Possible areas of application include:

* Education in the control of robots
* Artifical Intelligence and Robotics
* Industrial Automation
* Playing Games
* Robot Art
* Assembly by robots

1.2 Project Stages

The project will be divided into the following major sections:

1. Project Commencement

This will include a brief introduction to the project and a definition of the project stages.

2. Project Investigation

(i) Study of Robotic Software

A study of robot software in use today will be undertaken. Attention will be paid to what features are desirable and undesirable in "good" robotic software.

(ii) Study of the MINI-MOVER 5 robot

A study of the MINI-MOVER 5 robot will be undertaken. This study will examine the physical and software capabilities of the MINI-MOVER 5 robot. Attention will be paid to the limitations of the robot when interfaced to a microprocessor.

3. Project Design

Based upon the project investigation section a project design will be developed. The design will attempt to include the desirable features of robotic software systems without violating the capability and limitation bounds of the MINI-MOVER 5 robot.

4. Detailed Project Design

Basically this section is an expansion of section 3. This stage will provide detailed diagrams of system inputs/outputs and definitions of algorithms, the database and user inputs required. This section will also include the design of the database and a general outline of the programs. It will indicate the division of the programs into logical procedures and the definition of each procedure.

5. Interface Coding and Debugging

The interface code between the MINI-MOVER 5 and the M68000 microprocessor will be written, tested and debugged.

6. Coding and Debugging

The remaining code will be written, tested and debugged.

7. Report and Seminar

A report will be written and a seminar presented in an effort to communicate achievements and results.

CHAPTER 2
PROJECT INVESTIGATION

2.1 Study of Robotic Software

2.1.1 Robot Generations

The increasing level of sophistication of robotic software(1) is reflected in the "generations" of the evolution of robots. The "generations" of robot evolution can be categorised as follows:

Generation 1

The robots are programmable, control is open loop (i.e. no feed back) and motion is memory controlled (i.e. continuous play back of a taught motion path). Such robots have several degrees of freedom. Typical applications include spray painting and welding.

Generation 2

The robots are programmable, sensor controlled (i.e. feedback) and motion is memory controlled. Motion paths are interrupted by sensor feedback, these interrupts are serviced by preprogrammed functions in the controlling computer. Such robots can react to changes in their environment and thus are a powerful tool.

Generation 3

The robots have the same capabilities as generation 2 robots but with added capability of vision. Such robots manipulate objects on the basis of computer vision. These robots can be used in sophisticated industrial applications e.g. complicated assembly tasks.

Footnote

(1) Robotic software can also be referred to as robot programming languages.

Generation 4

The robots have adapative control capabilities. This means that robots exhibit automatic reprogrammability on the basis of sensor and vision inputs. Such robots form a very flexible and powerful tool.

Generation 5

The robots have artifical intelligence.

The current state of technology in robotics stands between generations 3 and 4.

2.1.2 Design of Robot Programming Languages

In the design of robot programming languages there are three basic approaches:

(1) Take an existing programming language and add to it routines to control the robot. This approach permits the full power of a language to be used.

(2) Write a library of routines so that the user program consists of a series of calls to these routines via simple control statements.

(3) Design a language specifically for robot manipulation.

Approach (3) may seem to be an expensive alternative but there are no programming languages available which have these featues: high level, general purpose, real time, support of co-operant and parallel, routines and are able to interface to external sensor and vision devices.

A general comment on the design of robot programming languages can be made: the design is closely related to the application.

2.1.3 Robot Programming Languages Levels

The sophistication level of robot programming languages follows a similar path to the evolution of robots. Robot programming languages have five distinct levels of sophistication. These levels are summarised in figure 2.1.3.1.

Level 1. Microcomputer/Hardware Level

At the lowest level of robot control, the microcomputer/hardware level, commands are highly dependent on the physical structures of the robot. Hence no formal programming languages exist at this level. The emphasis here is on converting joint coordinates to torques and forces on the motors, and on conveying sensor data to higher levels.

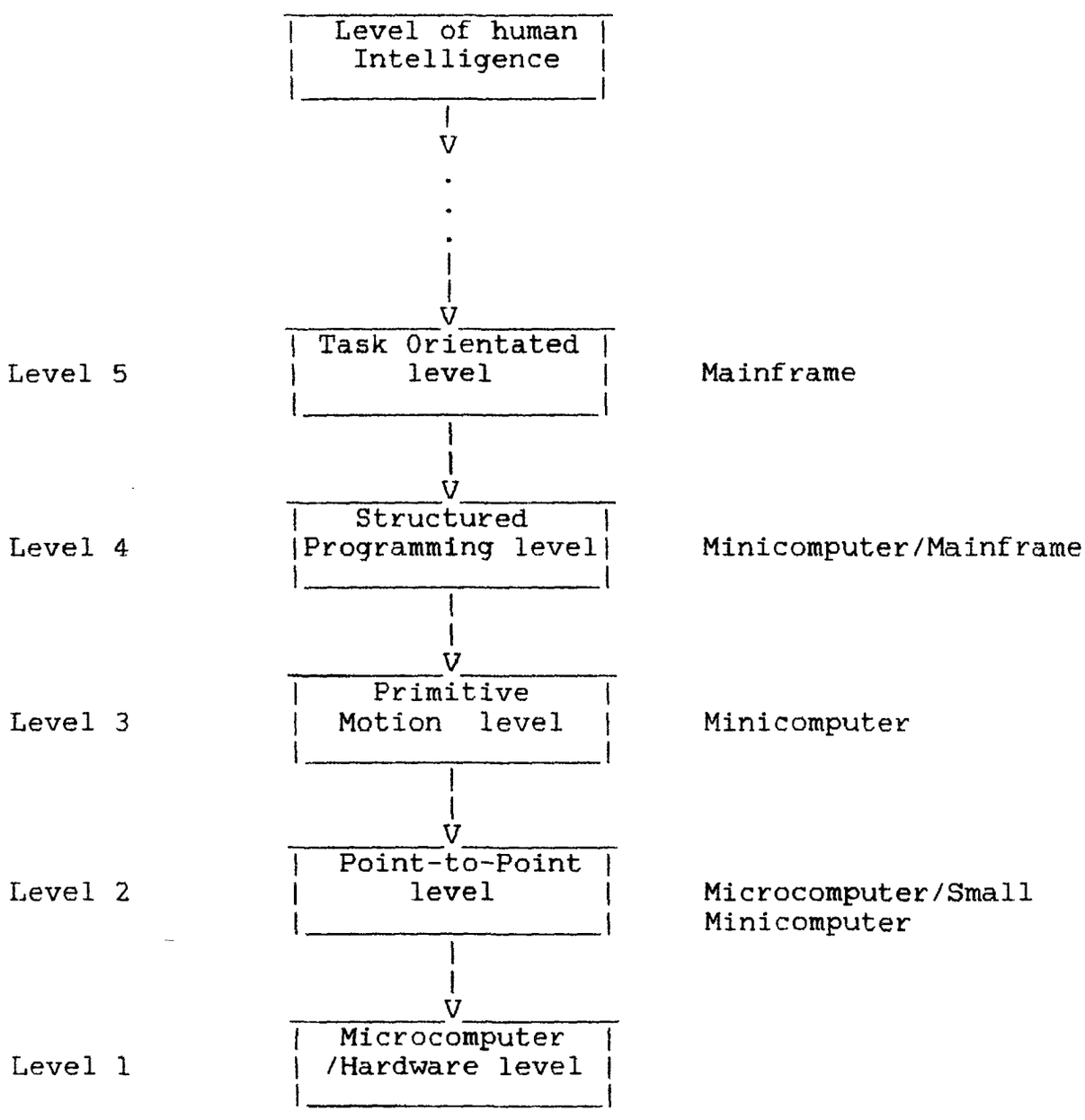

Computing Power Required
Level of human Intelligence
Level 5
Task Orientated level
Mainframe
Level 4
Structured Programming level
Minicomputer/Mainframe
Level 3
Primitive Motion level
Minicomputer
Level 2
Point-to-Point level
Microcomputer/Small Minicomputer
Level 1
Microcomputer /Hardware level

Figure 2.1.3.1

Level 2. Point to Point Level

Point to point robot languages are the most common available at present. They provide robot programming by enabling the user to save a series of positional points. These points are obtained by "manual" guiding of the robot through the required motions.

Usually the guiding is done by using a manual device that activates the joint motors in the robot and records the desired locations. This is known as "walk through teaching". Higher level guiding systems provide specialised function buttons which allow the editing of programs and the interaction of the program with external signals.

The advantage of point to point languages is that they are available and operatable now. Once a task has been programmed it can be repeated any number of times without operator intervention. Programs are easy to debug because testing is constantly being done on the robot itself. An online editing system allows the programming to be refined during an operating cycle until the optimum sequence has been achieved.

The disadvantage of point to point languages is that they include little or no branching and subroutine capability, and limited sensor interaction. The emphasis is on robot motion rather than the task to be performed. There is limited software to handle emergencies and no provision is made for off line programming.

The following languages are point to point languages: FUNKY and T3.

FUNKY is an IBM development, it is an advanced guiding system that produces robot programs through the use of manual guiding and a function keyboard.

T3 is a language provided with the T3 industrial robot manufactured by Cininatti Milacron, T3 is a commercially available system that uses guided teaching and function buttons to program robot tasks.

Level 3. Primitive Motion Level

Primitive motion level languages can be described as point to point motion in language form. These languages help shield the user from some of the cumbersome aspects of the lower level languages. The following aspects characterise primitive motion languages: simple branching, subroutines (generally with parameter passing), more powerful sensing capabilities (including vision), primitive parallel execution and limited coordinate transformation. Most of the primitive motion languages available today are based on interpreters or assemblers.

The disadvantage with primitive motion languages is that the emphasis is on robot motion rather than the task on hand. The languages do not provide complex control structures (while-do etc.) or effective parallel execution capabilities. There is limited software to handle emergencies and no provision is made for off line programming.

The following languages are primitive motion languages: ANAROD, EMILY, RCL, RPL, SIGLA and VAL.

ANAROD is the language provided with the Anomatic II Controller industrial robot manufactured by Anorad Corporation. ANAROD is commercially available and provides a powerful numerical control language with programmable mathematical expressions, variables, jumps and subroutines, and a self configuring capability.

EMILY is an early attempt by IBM to develop a higher level, workhorse robot language with a reasonably simple processor.

RCL is a language which is under development at Rensselaer Polytechnic Institute. RCL is a command orientated motion control language which is used to program a sequence of steps needed to accomplish a robot task.

RPL is a FORTRAN like user language developed by SRI International. RPL is designed to facilitate the writing and debugging of application programs for material handling, inspection, and assembly tasks.

SIGLA is a robot language available from Olivetti with their Super-Sigma robot. Sigla uses only 8K of memory, yet provides features such as parallel task control and variable instruction sets for software tailoring.

VAL is a system and language for programming computer controlled robots. VAL was developed over a period of several years by Unimation.

Level 4. Structured Programming Level

Structured Programming Level represents a major improvement over the primitive motion level because structured programming languages incorporate structured control constructs and permits extensive use of coordinate transforms. Characteristics of structured programming languages include: complex data structures, improvements in sensor interaction, improvements in parallel processing, use of predefined state variables, and user defineable data structures with parameter passing.

The complex data structures allow the definition of points, lines, planes, and frames, thus making transformations easier. State variables refer to reserved words such as arm, tool, leftwrist and gap. The sensing capabilities are similar to primitive motion level capabilities. Parallel processing is expanded by use of semaphore constructs: cobegin and coend. Motion on the structured level is defined in terms of transformations on the frame of the robot hand e.g. move, rotate and goto.

Structured level programs increase program understandability and aid task orientated programming by sophisticated parallel processing and state variable concepts. Off line programming is more feasible, making use of relational transformations.

A major asset and a major drawback of structured programming are coordinate transformations. Transformations allow many advtanages in programming but are hard to learn and use.

The following languages are structured programming languages: AL, HELP, MAPLE, MCL and PAL.

The AL programming system developed at the Stanford Artificial Intelligence Laboratory is a high level robot language with ALGOL like control and block constructs; predefined data types for scalars, vectors, rotations and positions; operations on those data types; local coordinate systems that can be affixed to one another; and the ability to specify motion in terms of objects grasped in the hands.

HELP is a robot language developed by General Electric. It is a high level procedural language that is relatively easy to learn to use, supports structured program design, supports similtaneous arm movement, is sufficiently comprehensive for robot operation, and has a special set of built in functions to support robot operation.

MAPLE is a robot language developed at IBM. Maple has a PL/1 like base language for computation and several extensions for directing a robot to carry out complex tasks.

MCL is an extension of the APT numerical control language developed by McDonnell Douglas Corporation. MCL is a high level language designed for off line programming of industrial robots and associated equipment under a robotic control system.

PAL is a robot programming system developed at Purdue University. Tasks are represented in terms of structured cartesian coordinates, and every motion command is a request to position and orientate the robot to satisfy a position equation.

Level 5. Task Orientated Level

A task orientated language conceals low level details such as sensors and coordinate transforms from the user. The languages make use of high-level commands such as place object 1 on object 2. To accomplish tasks such as this, a world modelling system is required. Such a system would involve tactile sensing, and a sophisticated vision system with which objects can be located and identified.

Task orientated language commands are divided into four classes:

- state change statements e.g. place
- tool statements e.g. operate

- fasten statements e.g. rivet
- miscellaneous statements e.g. verify

High level commands such as the above lead to ambiguities. To alleviate ambiguities the systems are usually designed so that program debugging proceeds interactively with the user. The compiler can question the user about any ambiguities.

Task orientated robot languages as yet are an unachieved dream, although IBM is currently developing a task orientated language called AUTOPASS. AUTOPASS is a high level programming system for computer controlled assembly work. AUTOPASS is orientated toward objects and assembly operations that enable the user to concentrate on the overall assembly sequence and to program with English like statements using familiar name and terminology.

2.1.4 Characteristics of a "Good" Robot Programming Language

(1) Clarity, Simplicity and Unity Concept

At point-to-point level this concept is extremely simple. Hence their success in industry. But point-to-point robot languages do not have the programming power of higher-level languages.

Primitive-motion level languages have a great number of commands, but few control constructs. They are composed of a hodge podge of commands. Thus primitive-motion languages lack unity. This is mainly due to the fact that they have been developed dynamically, thus leading to an overabundance of commmands.

Robot languages at the structured programming and task orientated levels were developed with consistent programming language characteristics in mind. The use of structured programming, and data structures, eases the demand for specific commands. The use of clauses such as MOVE greatly increase generality. These languages are well structured and consistent but their success in industry has yet to be proven.

(2) **Clarity of Program Structure**

Structured programming techniques were developed to make programs easier to comprehend, and debug, by using while-do and if-then-else constructs. Structured robot languages use structured programming constructs which give the program structure readability and understandability. Primitive-motion robot languages make use of goto and if-then constructs. Such constructs reduce the clarity of program structure when compared with structured languages.

(3) **Naturalness of Application**

Point-to-Point robot languages are hardly languages, yet, they have proved quite successful for many applications. The shortcomings of point-to-point languages led to the development of primitive-motion languages. But, lack of readable commands (most two letter), and control structures, make primitive-motion langauges slightly better than point-to-point languages.

Keywords and variables of any length can be used at the structured level. So, commands are understandable and self explanatory. Complex data structures are very useful.

Task-orientated languages are very natural to use, since, commands are ENGLISH like and use of transforms is hidden. These features make programming easy for any type of application. But high-level languages create ambiguity, so the user must understand lower-level languages for debugging.

(4) **Ease of Extension**

It is important that robot languages have a modular expandable structure that can handle present needs and future needs of new applications. One way to do this is through subroutines with parameter passing. Parameter passing avoids the problem of the reuse of global variables.

Point to point languages have no subroutines so they are difficult to extend.

Primitive motion languages have limited expansion since most have subroutine capabilities with no parameter passing.

All structured programming languages have powerful subroutine capabilities which allow parameter passing and several levels of nesting. Structured programming languages can be extended.

Task orientated languages have the same capabilities as structured languages.

(5) Debug and Support Facilities

Efficient debugging tools are of extreme importance in developing robot programs. Off line programming using graphics simulation makes debugging less critical, since program development takes place without robot use. Lower level languages have much more extensive debugging facilities because they are in actual use today and debugging must be done on the robot. Structured programming level languages, which are designed with off line programming in mind, provide few hands on debugging aids.

(6) Efficiency

The efficiency of a robot language depends upon the ease with which programs can be developed. Efficiency can be measured by two factors: programmability and portability. Programmability is the ease with which programs can be developed. Portability is the ease with which the language can be adapted to a new environment.

Programmability is a measure of the ease with which users of the robot language can produce correct executable code. The best basis for measuring programmability is the time taken to write a program.

A transportable, or portable, robot language is one that can easily be adapted for use by any robot on any computing facility. A measure of the portability of a language to a system is the amount of time it takes to complete the interface between the language and the system. Because of the number of computing facilities and robot mechanisms available, an intermediate step between the compiler and the system is one way to ensure widespread portability with minimal effort. This intermediate step is a low level pseudo language known as "p-code" which can

easily translated into the base language of the system.

Robot languages in use today are highly specific to the robot and the computer system on which they are developed. With the exception of the robot language AL, where an effort to make the language portable has been made, very little has been done to develop highly portable robot languages.

(7) Decision Making Capabilites

In order to function most efficiently and safely in real time, the robot system should be able to handle unexpected changes to the environment. This faculty requires a dynamic model of the robot environment, constant monitoring of the model for unexpected changes, and the ability to make intelligent decisions based on the changes. The robot must execute its task concurrently with the monitoring system. When an unexpected change occurs, the monitor interrupts task execution and attempts to return the model to the expected state. Only task orientated robot languages have simple decision making capabilities based on a dynamic world model of the environment.

(8) Interaction with External Devices and Sensors

A robot language must be able to handle interaction between the robot and external devices. Rarely will robots operate as stand alone units. Desirable language features, therefore, include messages that can be sent between devices and commands that allow concurrent activities of different devices.

The languages at the point-to-point level permit only very primitive interaction with external devices.

At the primitive-motion level, interaction ranges from almost none to fairly complex vision system interaction. Parallel processing at this level, if it exists, is restricted to concurrency between more than one robot.

Structured level languages have abilities that range from no interaction at all, to allowing portions of the code to apply to any device in the robot environment. Parallel processing, if provided, is in the form of waiting and signaling events.

The task orientated level provides commands that imply interaction with external devices and concurrent control of devices but do not explicitly send and recieve signals.

(9) Compilers V's Interpreters

In robot applications an interpreter has many advantages over a compiler. An interpreter executes code as it is encountered. A compiler passes through the code more than once before it generates code for the statements. A program that runs on a interpreter is easier to change and allows partial execution of sections of code. This is ideal for on line programming and debugging. However interpreters are generally slower at runtime because of parsing and interpretation. Structured control constructs are difficult to implement because of their complex branching requirements.

A popular solution to this problem is to provide a structured level language that compiles into a lower level primitive language. This compromise allows the programmer to use an interpreter during the debugging stages, but the disadvantage of such an approach is that the original structured code is not always the true source code for the lower level language once it has been debugged. (Little attempt has been made to recreate a structured level program from the lower level code.)

(10) Interaction with World Modelling Systems

The ability to interact effectively with a world modelling system is the key to the development of a truly intelligent robot programming system. Without a world model the robot is essentially blind to it's environment, and the user must specify any changes that the robot's activity makes to the environment. A dynamic world modelling system will not only keep track of the initial state of the robot environment, but will, also, adapt changes made by the robot into the model. Research is currently being undertaken in this area, but currently no functional dynamic world modelling system is available.

Although complex world modelling systems lie in the future, some of the robot systems today do have simple world modelling facilities under development. The most important fact to consider in the current development of a robot programming language is the potential ease of extension of the language to accept information from a world modelling system when a more powerful one becomes available.

2.1.5 Conclusions

On the basis of the previous information some general conclusions can be formulated about the development of robotic software.

Because of the interactive nature of robot programming and the need for rapid debugging techniques, an interpreter based language is favoured over a compiler based one. To alleviate the disadvantage of slow run time performance, use an interpeter to produce correct programs, but let a seperate compiler produce fast efficient code from the programs at run time.

Robot languages should provide as many debugging features as possible. Particularly off line debugging and development facilities which use computer graphics to simulate the robot and its environment.

In order to meet the growing needs of the robot industry, a robot language must be extensible. Extensibility implies a modular language structure with user defined subroutines which have parameter passing capabilities. Global variables should not be allowed, this can be done by parameterisation of variables in subroutine calls. Multiple nesting of subroutines should be allowed.

The robot language should maintain structured programming techniques and present them in as simple a manner as possible.

The use of transformations is important in robotics. So a robot language should allow the definition and use of transformations.

Variable names should be unrestricted in length and recognition should involve as many characters as possible.

Robot languages should use English like syntax for language readability and understandability.

A robot environment usually involves interaction among several devices. A useful robot language would allow easy interaction and parallel execution capabilities with peripheral devices e.g. sensors and vision systems. Without interaction with sensors and peripheral devices the robot will be isolated from the world around it and will never be able to perform with any high degree of proficiency.

2.2 Study of MINI-MOVER 5 Robot

The performance characteristics of the MINI-MOVER 5 robot arm are summarised in Table 2.2.1. The robot arm consists of a stationary base and four movable segments connected in series by the base, shoulder, elbow and wrist joints. The major structural components are shown in Figure 2.2.1. The arm has five degrees of freedom which permits combined motion of the body, shoulder, elbow and wrist to position the arm anywhere from close to the base to 444mm away. The configuration of these joints defines the position and orientation of the robot hand or gripper.

The arm members are hollow, formed from formed aluminium sheet metal. The three outer segments are joined by hinged shafts that define the axis of the joint. Each joint is controlled by a flexible cable that runs from a drive unit mounted on the body to a pulley mounted on the joint shaft. The drive unit for each joint consists of a stepping motor, reduction gearing and a cable drum. From each drum, a tensioned cable goes out over pulleys to the member being driven and returns to the drum. Rotation of the drum causes rotation of each member in proportion to the ratio of the diameter of the drive pulley attached to that member to the diameter of the drum.

2.2.1 Robot Cabling Design

The arrangement of cables used in the MINI-MOVER 5 is shown in Figure 2.2.1.1. Since the cabling design was made to simplify geometry and maintain a low cost, several of the joints interact. Movement of one joint may result in possible unwanted motion in another. Compensation for these interactions can be made in software. Thus, the cabling details must be understood in order to properly control the motion of the arm.

Base Rotation (Joint 1). The base drive cable passes over two idler pulleys, making a 90 degree bend to a drive pulley fixed to the base. The base drive motor causes the entire arm to rotate about the base joint.

Shoulder Bend (Joint 2). The shoulder drive cable passes around the drive pulley fixed to the upper arm segment and rotates the upper arm segment about the shoulder joint.

Table 2.2.1

MINI MOVER 5 Performance Characteristics

GENERAL

Configuration	5 revolute axes and integral gripper
Drive	Electric Stepper Motors, open loop
Controller	Microcomputer provided by user
Interface	8 bit parallel
Power requirement	12 volts, 4 amps DC

PERFORMANCE

Resolution	0.3 mm on all axes
Load Capacity	225 gm at full extension 450 gm at half extension
Gripping Force	13 N maximum
Reach	444 mm
Static Load Force	18 N maximum

PERFORMANCE DETAILS

Motion	Range	Speed (full load)	Speed (no load)
Base	+90,-90 deg	0.37 rad/s	0.42 rad/s
Shoulder	+144,-35 deg	0.15 rad/s	0.36 rad/s
Elbow	+0,-149 deg	0.23 rad/s	0.82 rad/s
Wrist Roll	+90,-90 deg	1.31 rad/s	2.02 rad/s
Wrist Pitch	+180,-180 deg	1.31 rad/s	2.02 rad/s
Hand	0-75 mm	13 N/s	20 mm/s

PHYSICAL CHARACTERISTICS

Arm weight	3 kg
Interface cable length	900 mm

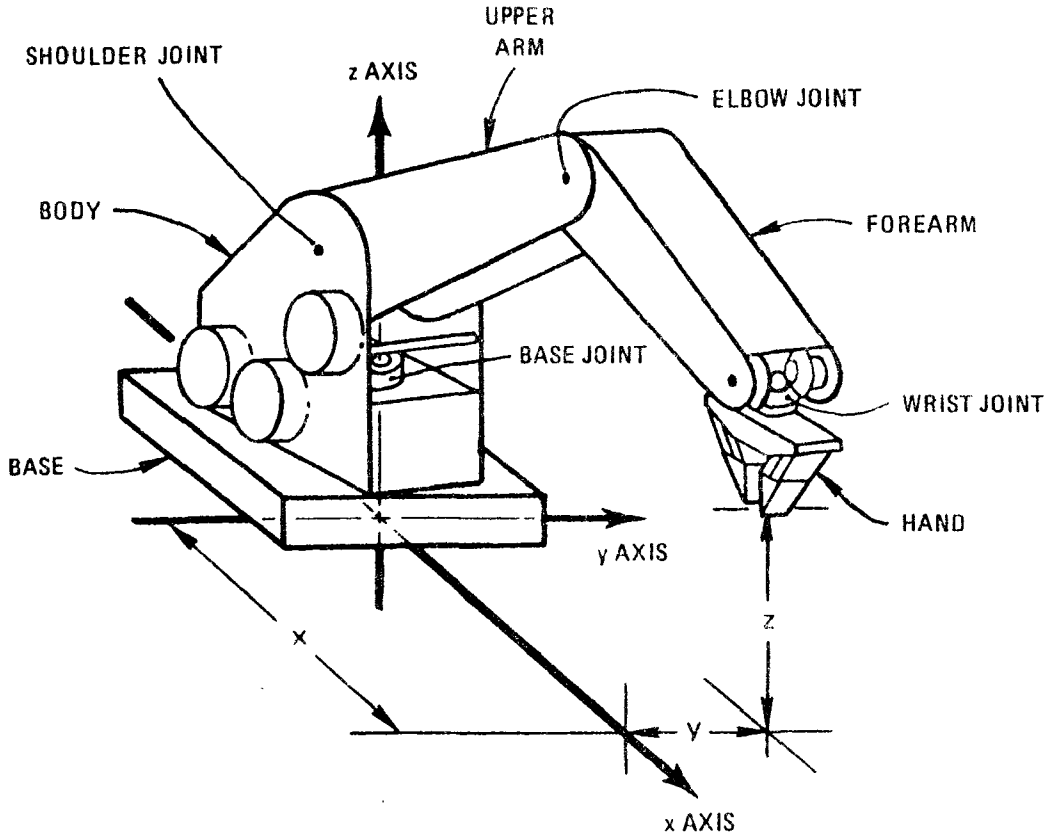


Figure 2.2.1 Major Structural Components

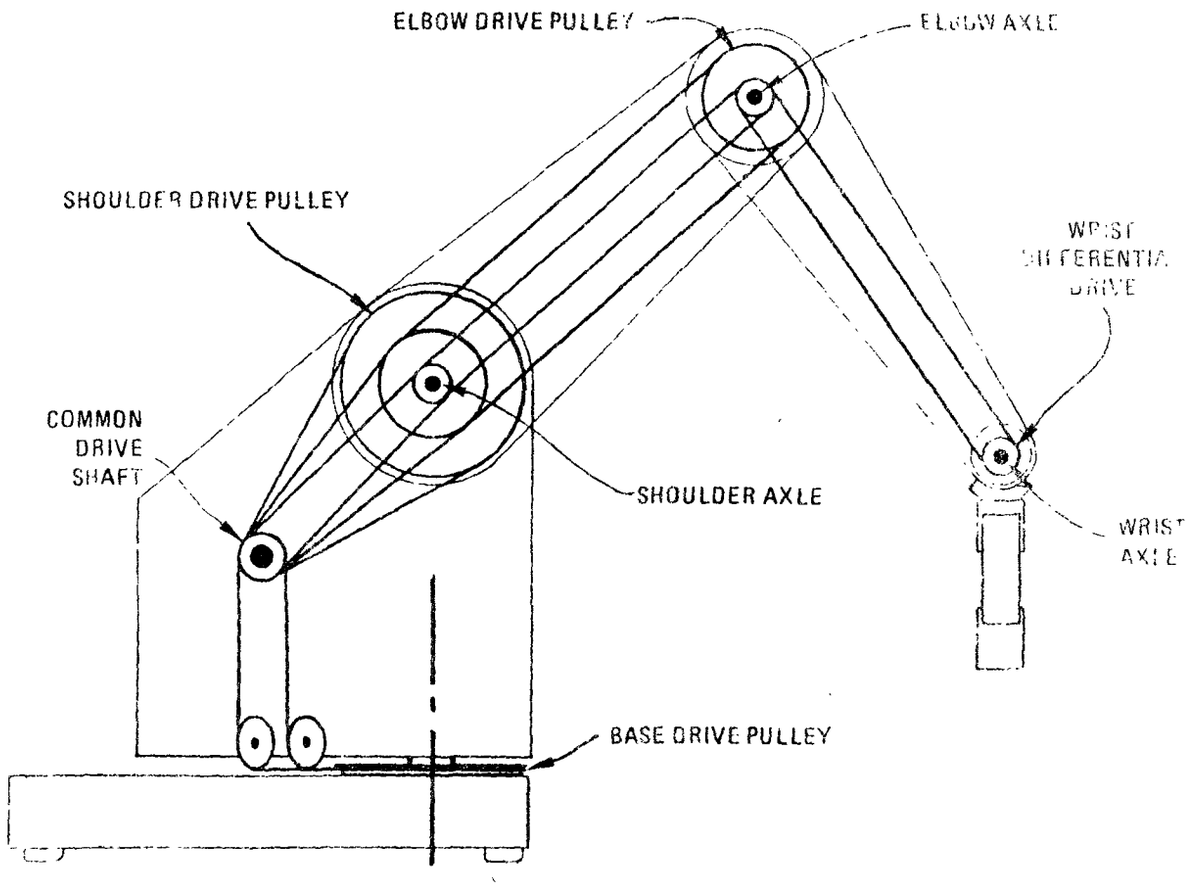


Figure 2.2.1.1 Cabling Diagram

Elbow Bend (Joint 3). The elbow drive cable passes around an idler pulley on the shoulder axis to a drive pulley in the lower arm segment. Joints 2 and 3 interact, so that changing the shoulder angle results in an equal change in the elbow angle. This interaction was designed so that the elbow drive controls the forearm with respect to the horizontal. However the limits of the forearm motion are measured with respect to the upper arm not the horizontal (x-y) plane.

Wrist (Joint 4, left and Joint 5, right). The fourth and fifth drive cables pass around idler pulleys on both the shoulder and elbow joints and terminate on the drive pulleys of the left and right wrist differential gears. The interaction of these joints with joints 2 and 3 has the effect that the wrist drives control the angle of the hand with respect to the horizontal (pitch) and the rotation of the hand around its pointing vector (roll) as shown in Figure 2.2.1.2. Through the action of the differential, the pitch angle P equals the average of the positions of the left and right gears and the roll angle R is their difference. The range of motion is limited to +270,-270 degrees at each of the wrist gears due to cable length limits and to +90,-90 degrees in pitch due to interaction from the lower arm.

Hand (Joint 6). The hand drive cable passes over idler pulleys located on the shoulder and elbow joints, through the center of the wrist differential, and finally terminates at the hand. This drive interacts with joint 3 (elbow) in that the elbow bend will cause the hand to open. This can be compensated in software by closing the hand exactly the same number of steps that the elbow is raised.

The hand housing is attached to the output miter gear of the differential gear set. The hand housing supports the two pairs of links, each pair which terminates in a grip, as shown in Figure 2.2.1.3. The housing, links and grips are attached to each other by small pins. Torsion springs provide the force to open the hand as the hand cable is slackened. As the hand cable is pulled, the hand closes upon an object to be grasped and the cable tension mounts, activating a tension sensitive switch. The state of this switch can be read by the control computer which may stop the drive motor as soon as it closes, in which case the grasping force is limited to about 0.3 N.

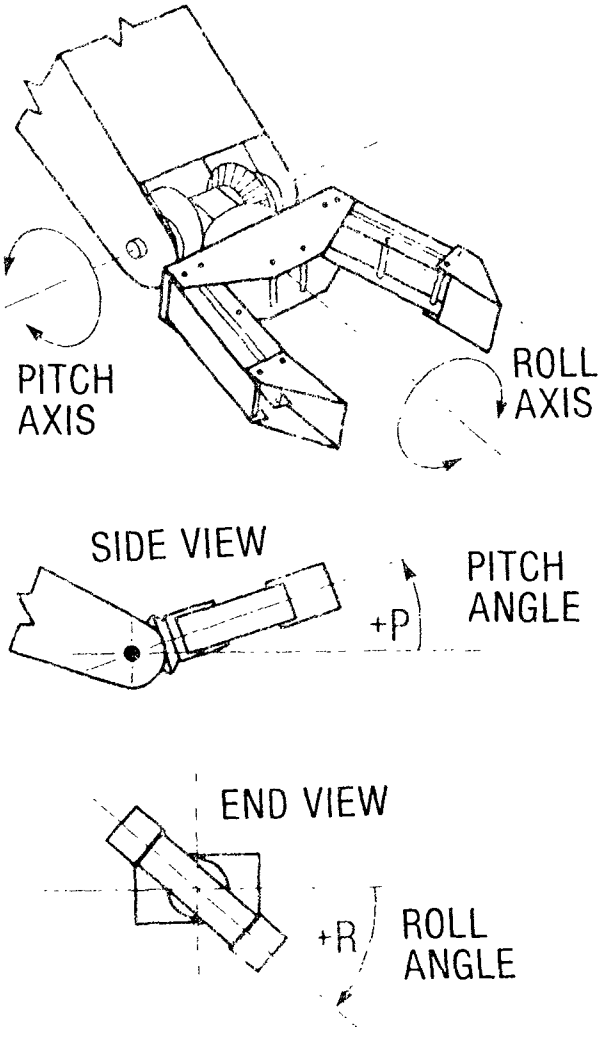


Figure 2.2.1.2 Defintion of Roll and Pitch angles.

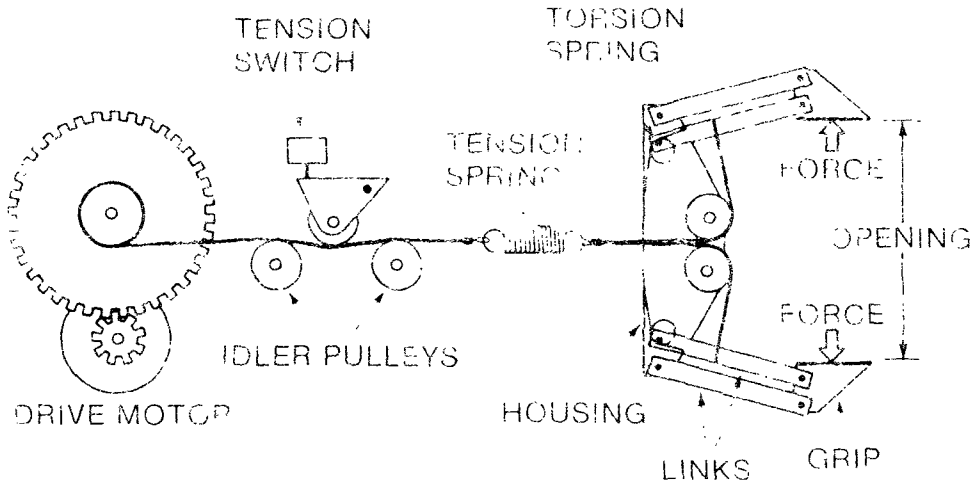


Figure 2.2.1.3 Control system for gripping

Gripping force can be increased by pulling in more cable after the tension switch closure is detected. Additional cable taken up stretches the extension spring which is mounted in series with the hand drive. This additional cable take up is converted into gripping force at the rate of 1 Newton for every 12 steps. Thus gripping force as well as hand opening can be controlled by the same cable drive. Figure 2.2.1.4 gives a graph of the gripper force v's motor steps.

2.2.2 Stepping Motor Control

Each of the cable drives is controlled by a stepper motor. The motors used have four seperate windings, each driven by a power transistor. The drive is digital, with the transistors switched on or off to obtain a desired pattern of currents in the motor. By appropriately changing the pattern of currents a rotating magnetic field is obtained inside the motor, causing it to rotate in small increments or steps. In the MINI-MOVER 5 each of the four coils is individually controlled from the computer. This allows the pattern of motor currents to be controlled by the computer software.

In order to step a motor the sequence of binary patterns shown in Figure 2.2.2.1 is output to the desired motor. The pattern on each row of the table, when sent to the corresponding drive transistors of the motor, will cause it to move one step. To step the motor clockwise the patterns are output sequentially from top to bottom. When the end of the table is reached, it is necessary to wrap around to the other end of the table and continue sequentially. To move the motor in the opposite direction, the sequence is reversed.

2.2.3 Computer Interface

The MINI-MOVER 5 can be interfaced to a microcompuer via a 8-bit parallel I/O port. The interface is organised as seven 4-bit parallel output and a single 4-bit input as shown in Figure 2.2.3.1. Information on the address lines is used to channel the four bits of output data to the appropriate motor drive through individual 4-bit latches. In this way the computer can control the 4-bit phase patterns on each motor with a minimum of hardware. After a phase pattern is sent to a 4-bit latch, it is held by the latches until the next pattern is sent. Outputs of the latches control the power drivers and their respective motor coils.

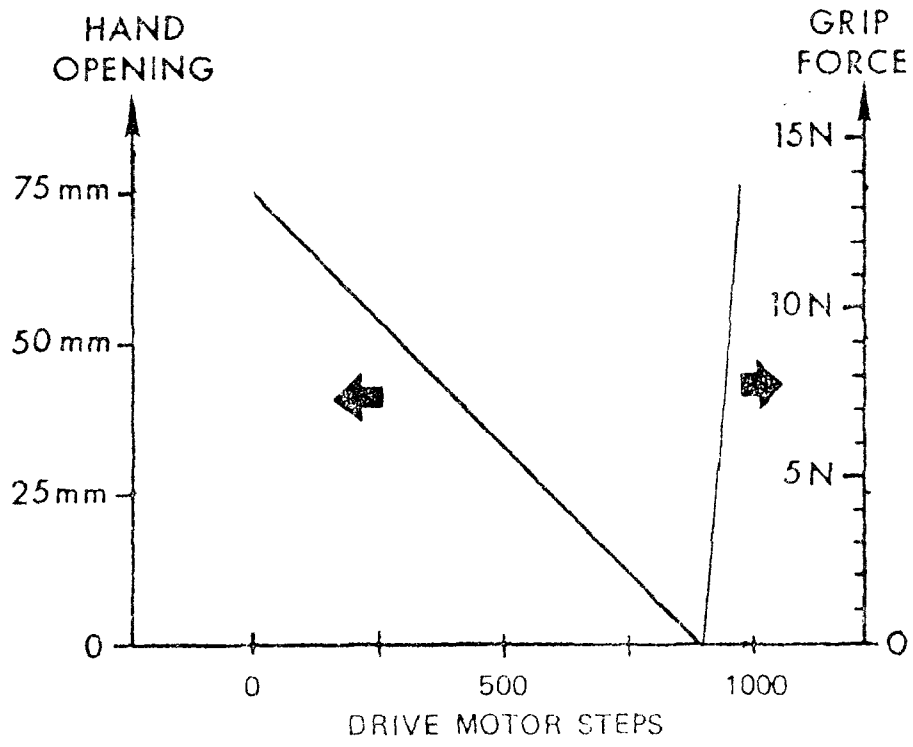


Figure 2.2.1.4 Gripper Force Graph

	$\phi 1$	$\phi 2$	$\phi 3$	$\phi 4$	
CLOCKWISE ↓	0	1	0	1	↑ COUNTERCLOCKWISE
	0	1	0	0	
	0	1	1	0	
	0	0	1	0	
	1	0	1	0	
	1	0	0	0	
	1	0	0	1	
	0	0	0	1	

Figure 2.2.2.1 Table of drive patterns for stepper motor coils.

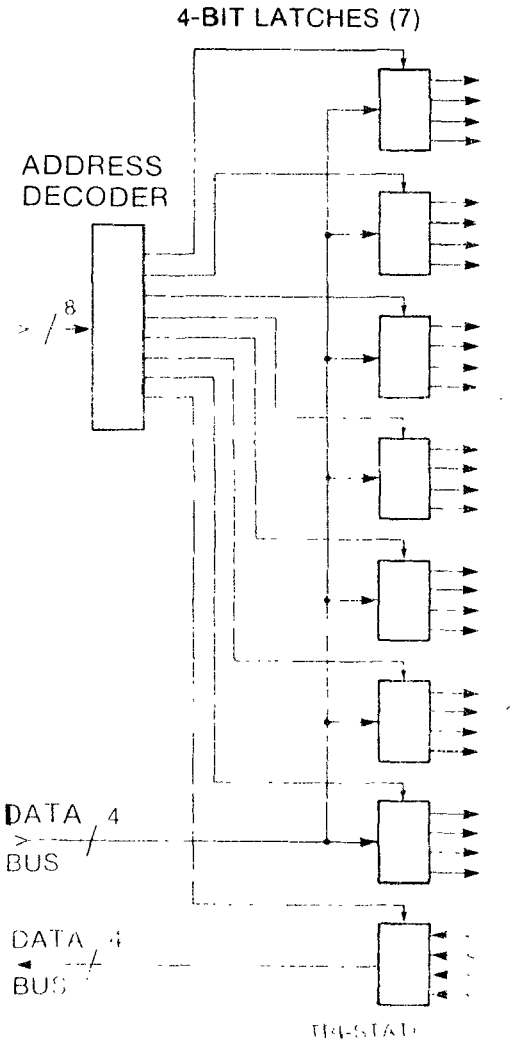


Figure 2.2.3.1 Block diagram of computer interface

The grip switch is connected to one of the inputs of the auxillary port, permitting the computer to close the hand until the gripping force builds up. The other inputs and outputs of the auxillary port are available to the user for experimentation with other sensors or controls. These inputs may be used for tactile, proximity, force and/or position sensors which can be mounted on the hand, arm extremities, and or the table top. Note also that an auxillary output port is available. This port may be used to control a work turntable, for example, to control external parts feeders, or synchronise other equipment with arm operations.

2.2.4 Conclusions

The fact that the electric pulse stepper motors are open loop control (i.e. no feedback) is major disadvantage. This means that all positional information about the MINI-MOVER 5 must be maintained in software. The software positional information approach could lead to errors if stepper motor slippage or an overrun of the end points occurred. End point overrun could be avoided by placing software limits on the endpoints. Such software limits would involve the use of joint transformations due to the interaction of the joints. The effectiveness of software limits would depend on the robot user initialising the positional information system to the correct values.

The robotic software written to drive the MINI-MOVER must compensate for joint interaction.

CHAPTER 3
PROJECT DESIGN

3.1 Definition of Design Approach

Based upon the study of robotic software and the study of the MINI-MOVER 5 capabilities it was decided that most of the "good" robot programming features could be implemented on the MINI-MOVER 5 if the following approach was adopted:

(1) Define a concise set of commands with which the MINI-MOVER 5 can be controlled. The commands in this instruction set will be transparent to be user and independent of each other. Any MINI-MOVER 5 robot action can be programmed by this instruction set, without the need for definition of a new command. The commands will be designed and written in a structured manner. The MINI-MOVER 5 can only be controlled by the defined set of commands. Thus this command set will form a robot control language for the MINI-MOVER 5 robot. The control language will be stored in PROM (programmable read only memory) on the M68000 microprocessor board.

(2) A high level structured language which has "good" extensibility features will be extended to include the defined robot control language. The high level application programs for the MINI-MOVER 5 which include calls to the robot control language will be developed on a host computer (Perkin Elmer 3220). Upon completion of the development of the high level application program, the program will be cross-assembled into M68000 assembler code and down-line loaded into the M68000 microprocessor, see Figure 3.1.1. For this project the programming language C was chosen as the high level language. The reason for the choice of C as the high level language was the availability of a M68000 cross-assembler.

This project design is one that extends a higher level language to include desirable robot language features. The effectiveness of such a solution depends on the design of the robot control language. A poor design will impair the flexibility and effectiveness of the robotic software.

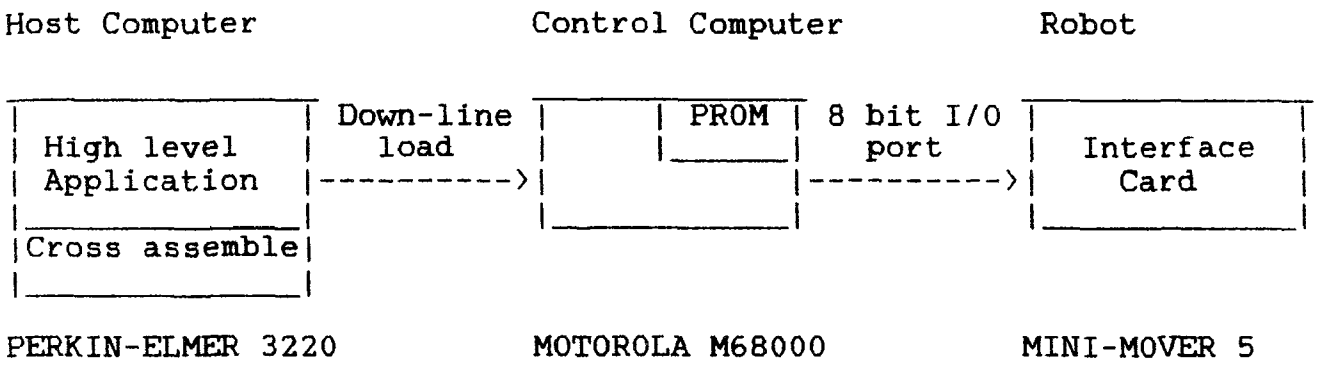

Host Computer
Control Computer
Robot
High level Application
Cross assemble
Down-line load
PROM
8 bit I/O port
Interface Card
PERKIN-ELMER 3220
MOTOROLA M68000
MINI-MOVER 5

Figure 3.1.1.

CHAPTER 4
DETAILED PROJECT DESIGN

4.1 Detailed Design Approach

Expanding upon the project design scope it is necessary to define and develop the robot control language for the MINI-MOVER 5 robot. Before defining a set of commands to be included in the robot control language it would be a good idea to look at what kind of actions the MINI-MOVER 5 could do. Typical examples of such actions include:

(1) Move the shoulder 20 motor steps.
(2) Turn the base 58 degrees clockwise.
(3) Close the hand (gripper) on an object.
(4) Open the gripper to an opening of 50mm.
(5) Move the shoulder 10 degrees up and elbow 34 degrees down.
(6) Move the robot from cartesian point (X1,Y1,Z1) to cartesian point (X2,Y2,Z2).
(7) Teach the robot a task.

From the above expectations of the capabilities of the MINI-MOVER 5 a number of observations can be made. These being:

(1) There are three distinct types of robot motion control, associated with each type of motion control is a positional reference frame to describe the position of the robot. The three classes of robot motion control are as follows:

(i) Robot Motor Control

At this level of control emphasis is upon control of the robot stepper motors. Since the stepper motors are open loop, positional information must be maintained in software by way of positional registers. Each of the six stepper motors has a corresponding positional register. These positional registers form a reference frame which describes the position of the MINI-MOVER 5 in terms of the number of steps each stepper motor has been turned. Such positional information provides valuable data but does not convey to the user any information regarding the position of the robots joints or the position of the robot in the cartesian workspace. Thus there is a need for the

ability to translate the positional register reference frame into joint coordinate and cartesian coordinate reference frames.

(ii) Robot Joint Control

At this level of control the emphasis is upon control of the robot joints. To control the robot by joint control there is a need for positional information to be maintained in a reference frame which describes the position of the robot in terms of joint angles for each moveable segment. To move the robot by joint control the joint coordinate reference frame must be translated into positional register values and passed as parameters to the Robot Motor Control level. Joint control is the most convenient way to control the MINI-MOVER 5. But there is still the need to convey information to the user in terms of cartesian coordinates. This can be done by translation of the joint coordinate reference frame into a cartesian reference frame.

(iii) Robot Cartesian Control

At this level of control the emphasis is upon control of the robot in cartesian coordinates. Such motion control requires positional information to be maintained in a reference frame which describes the location of the robot in terms of specifying the position and the orientation of the end point of the robot arm in 3 dimensional space. The end point refers to the centre point between the two finger tips of the gripper. To move the robot by cartesian control the cartesian coordinate reference frame must be translated into joint coordinate values and passed as parameters to the Robot Joint Control level. The Robot Joint Control level will handle the translation of the joint coordinates into positional register values. The positional register values will be passed as parameters to the Robot Motor Control level. The Robot Motor Control level will move the robot to the position described by the supplied arguments. Cartesian coordinate control is useful particularly for assembly tasks. This level of control is high enough to obscure low level detail and makes its workings transparent to the user.

The three types of robot control modes involve different translations. These translations can be divided into two classes: Forward and Backward translations. Forward translation is the translation of positional register information into joint coordinate information, and the translation of joint coordinate information into cartesian coordinate information. Backward translation is the converse of forward translation, that is, the translation of cartesian coordinate information into joint coordinate information, and the translation of joint coordinate information into positional register information.

(2) The MINI-MOVER 5 robot must have coordinated motion capabilities. For example if a motion of the robot requires movement of the shoulder and elbow, and this action is undertaken by moving the elbow first and the shoulder second, a rather awkward and unpredictable path results. However if the motion was undertaken by interleaving the motions of the shoulder and the elbow during the duration of the motion a coordinated motion results. Coordinated motion produces a straighter path from A to B and is also faster.

(3) To fulfil the requirement of being able to close the gripper on a object, the software must have the ability to monitor the sensor switch located in the robot arm.

(4) To fulfil the requirement of being able to teach the MINI-MOVER 5 tasks there is a need for a database which stores the positional information about the robot. A database will enable the capability to "remember" a variable number of points or positions associated with a taught task.

4.2 Robot Control Language Design

Based upon the the previous observations a set of commands to control the MINI-MOVER 5 can be drafted.

4.2.1 Initialise (INIT) Command

The INIT command sets up the interface between the M68000 microprocessor and the the MINI-MOVER 5 robot. The interface is a Peripheral Interface Adapter (PIA) located on the M68000 board which is set up to allow an 8 bit output into the MINI-MOVER 5 and a 4 bit input from the MINI-MOVER 5. Upon completion of the computer interface setup the MINI-MOVER 5 and the system database are reset by issuing the RESET command.

Algorithm

```
INIT
        begin
        Set up PIA
        RESET
        end
```

4.2.2 Reset MINI-MOVER 5 and Database (RESET) Command

The RESET command resets the electric currents in all six stepper motors in the MINI-MOVER 5 robot to zero. Such a feature is useful as it allows the robot to be backdriven (i.e. the robot can be moved by grabbing it and moving it around). The RESET command also resets the system database to its initial value. Since all positional information is maintained in software there is a need for the robot positional software to be initialised at a known position. A initial position is required so that the physical position of the robot matches the sofware position of the robot. When positional initialisation is required the arm is either moved manually or by using the SET commands to a fixed position. The initial position is shown in Figure 4.2.2.1. A calibration grid is suppied in Appendix A to simplify initialisation.

The cartesian coordinates of the initial position are:

X = 200 mm
Y = 0 mm
Z = 0 mm

The corresponding joint coordinates associated with the above cartesian coordinates are:

Base = 0 degrees
Shoulder = 24.95500113 degrees
Elbow = -77.39549286 degrees
Pitch = -90 degrees
Roll = 0 degrees
Gripper = 0 mm (closed)

The six positional registers P1,P2,P3,P4,P5 and P6 are initialised to zero.

Algorithm

RESET

```
begin
Set motor currents to zero.
P1 = P2 = P3 = P4 = P5 = P6 = 0
BASE = 0
SHOULDER = 24.95500113
ELBOW = -77.39549286
PITCH = 90
ROLL = 0
GRIPPER = 0
X = 200
Y = 0
Z = 0
end
```

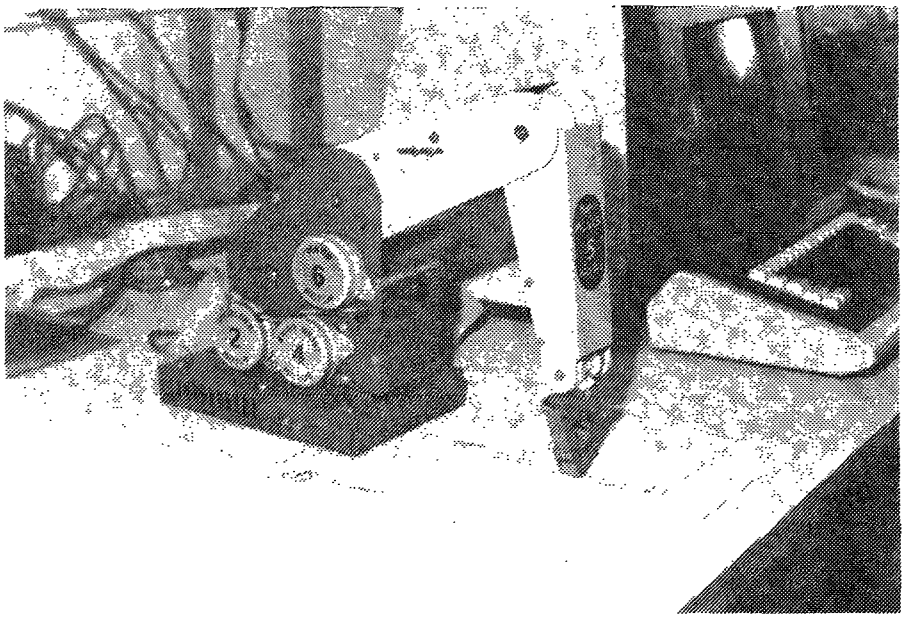


Figure 4.2.2.1 Initialisation Position

4.2.3 Robot Motor Driver (STEP) Command

One of the complicated aspects of controlling the MINI-MOVER 5 by computer is to control the motion of all six stepper motors similtaneously. The STEP command does this task. To do this the STEP command has seven parameters passed to it when it is called i.e. STEP(D,J1,J2,J3,J4,J5,J6).

D is a positive integer which represents the delay between the pulses to the stepper motors and J1,J2,....,J6 are signed integers which represent the number of steps that each of the six motors is to be turned.

The delay D determines the speed of the motion.

The integers J1,J2,....,J6 determine the motion of each joint. The sign of each number indicates the direction each motor should be driven, the magnitude indicates the number of steps. The J1,J2,....,J6 integers represent the following stepper motors:

J1 - Base swivel
J2 - Shoulder bend
J3 - Elbow bend
J4 - Right Wrist
J5 - Left Wrist
J6 - Gripper

Roll and Pitch movements are achieved by combined movements of the left and right wrist motors. Pitch is obtained by moving both left and right wrists in the same direction. Roll is obtained by motion of both wrists in opposite directions.

The STEP command uncouples the elbow (J3) and the gripper (J6) interaction, such that a command to move the elbow will not affect the gripper opening.

Additionally the STEP command linearly coordinates unequal joint (J1,J2,..,J6) integers to obtain coordinated motion. For example if the base motor is to move 21 steps and the shoulder motor is to move 3 steps the resultant timing is shown in Figure 4.2.3.1.

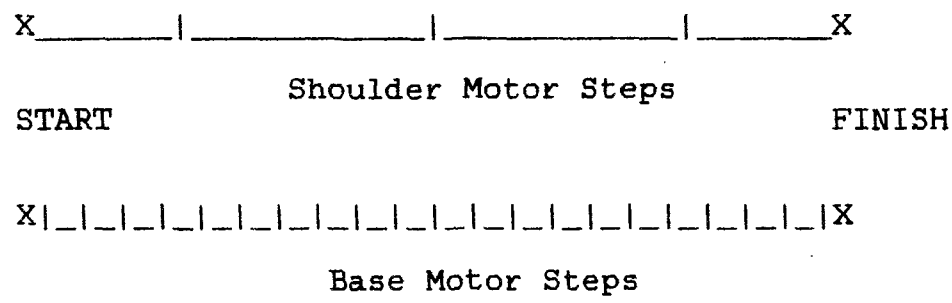


Figure 4.2.3.1 Step Timing Diagram

The STEP command also monitors a STOP motion flag. This flag is used if an event occurs elsewhere in the system and there is a need to abandon the current robot motion.

Finally the STEP command maintains robot positional information by updating the positional registers P1,P2,P3,P4,P5,P6 which correspond to the base, shoulder, elbow, right wrist, left wrist and gripper motors. A positional register is updated every time its corresponding stepper motor is turned.

The algorithm to control the stepping is a modified version of one developed by T.A. Siem presented in a paper called "Applying Microprocessors in Tool Design".

Algorithm

```
STEP(D,J1,J2,J3,J4,J5,J6)
      begin
      M = max(J1,J2,J3,J4,J5,J6)
      for i = 1 to 6 do
              begin
              direction[i] = sign(J[i])
              J[i] = abs(J[i])
              sum[i] = M/2
              end
      N = M
      while N <> 0 and not STOP do
              begin
                      for i = 1 to 6 do
                      begin
                      sum[i] = sum[i] - J[i]
                      if sum[i] < 0 then
                                begin
                                sum[i] = sum[i] + M
                                stepmotor(i)
                                P[i] = P[i] + direction[i]
                                end
                      end
              wait(D)
              N = N - 1
              end
      end
```

4.2.4 CLOSE Command

The CLOSE commmand causes the gripper (hand) to close until the grip switch indicates that the gripping force has built up to at least 1.5N. This occurs either when the gripper fingers close on an object, or when the gripper fingers close and touch one another.

The CLOSE command is implemented by using the STEP comand to close the gripper one step, subtracting one from the positional gripper register (P6) and checking the grip switch. This sequence is repeated until the grip switch closes.

Algorithm

```
CLOSE
        begin
        while Grip Switch not CLOSED do
                begin
                STEP(D,0,0,0,0,0,-1)
                P6 = P6 - 1
                end
        end
```

4.2.5 READ Command

The READ command is used to access the system database and depending on the supplied parameter returns the values of one of the following:

- Robot Motor Positional Register Values.
- Robot Joint Coordinate Values.
- Robot Cartesian Coordinate Values.

Algorithm

```
READ(i)
      begin
      if i = 1 then
              return(P1,P2,P3,P4,P5,P6)
      else
              if i = 2 then
                      return(BASE,SHOULDER,ELBOW,
                              PITCH,ROLL,GRIPPER)
              else
                      return(X,Y,Z)
      end
```

4.2.6 STATUS Command

The STATUS command is used to display the positional information about the MINI-MOVER 5 robot on the terminal console.

The STATUS command is implemented by using the READ command to access the system database and then displaying the returned information on the console. The format of the status display is given in Figure 4.2.6.1.

Algorithm

```
STATUS
      begin
      read(1)
      display(P1,P2,P3,P4,P5,P6)
      read(2)
      display(BASE,SHOULDER,ELBOW,PITCH,ROLL,GRIPPER)
      read(2)
      display(X,Y,Z)
      end
```

DATE _____

Video Display Terminal Format

EXHIBIT NO. _____

PROJECT _____ SYSTEM _____ PROGRAM _____ PAGE _____ of _____

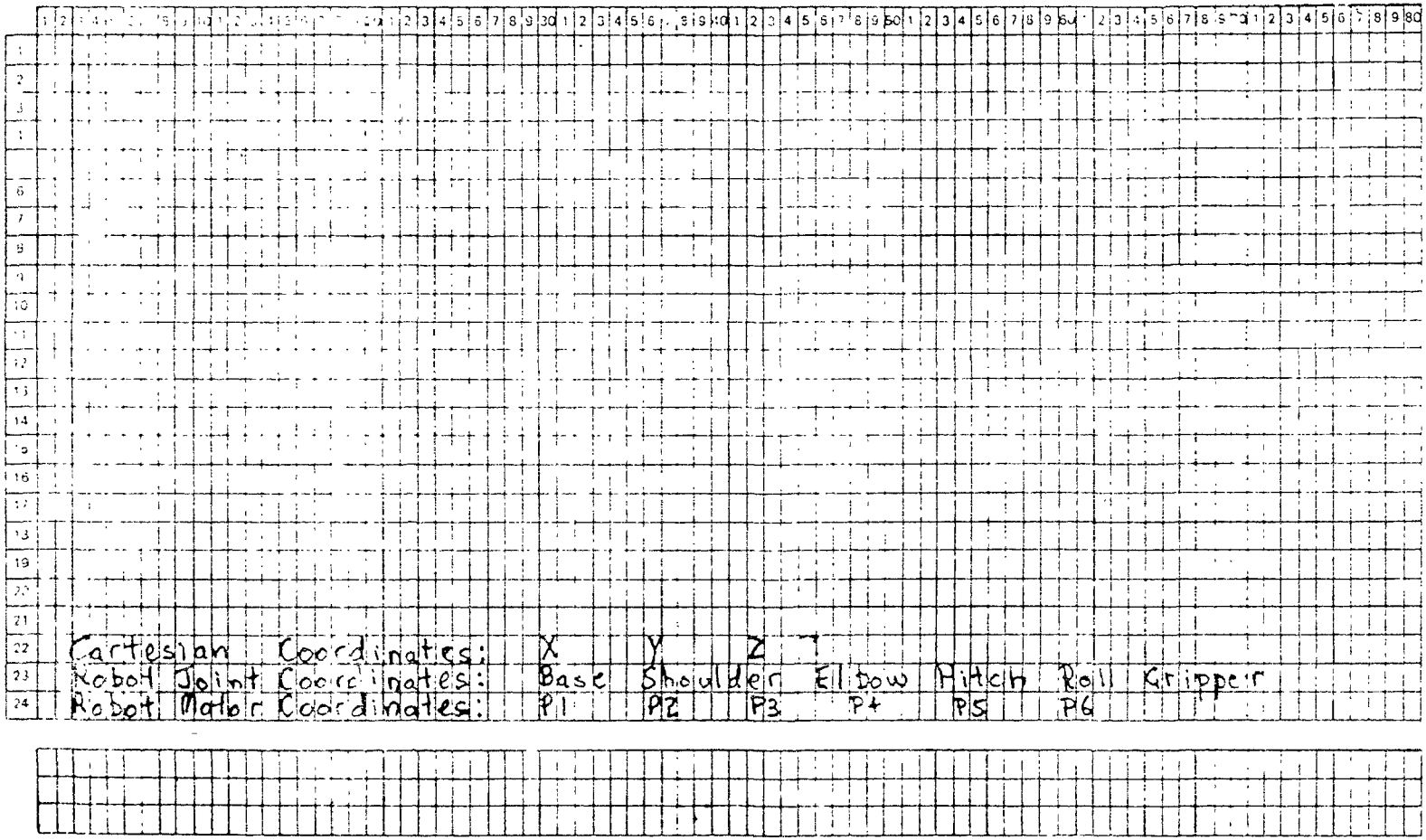


FIGURE 4.2.6.1 STATUS COMMAND SCREEN DISPLAY

REMARKS _____

4.2.7 BOUNDS Command

The BOUNDS command is used to check if a proposed robot motion will violate the physical bounds of the MINI-MOVER 5 i.e. cause overrun of the robots end points. The bounds checking has to be implemented at joint coordinate level due to the interaction of joints. The bounds violation checking can be summarised by the following set of rules. The rules are given in terms of joint angles and are expressed in degrees, except the gripper opening which is expressed in millimetres.

```
 -90 <= BASE <= 90
 -22 <= SHOULDER <= 139
   0 <= SHOULDER - ELBOW <= 149
        (Handles shoulder/elbow interaction)
-180 <= ROLL <= 180
 -90 <= PITCH <= 90
 -90 <= PITCH - ELBOW <= 90
        (Handles hand/elbow interaction)
   0 <= GRIPPER <= 75
        (Handles gripper opening)
```

4.2.8 HOME Command

The HOME command is used to return the MINI-MOVER 5 robot to its initialisation point. This is done by accessing the system database and reading the positional registers P1,P2,.....,P6. The read values are negated and supplied as parameters to the STEP command, which moves the robot to its initial position.

Algorithm

```
HOME
        begin
        read(1)
        STEP(D,-P1,-P2,-P3,-P4,-P5,-P6)
        end
```

4.2.9 Forward Translation Commands

The forward translation commands consist of two commands; the Forward Joint (FWDJOINT) translation command and the Forward Cartesian (FWDCART) translation command.

4.2.9.1 Forward Joint (FWDJOINT) Command

The FWDJOINT command accesses the system database to read the positional register values; P1,P2,...,P6. It then translates the positional register values into joint coordinates using conversion factors and updates the system database joint coordinates; BASE, SHOULDER, ELBOW, PITCH, ROLL and GRIPPER.

The conversion factors to translate the positional register values into joint coordinate values are given in Table 4.2.9.1.1. Note: The joint angles are stored as radians except GRIPPER which is stored in millimetres. The PITCH angle (P) and ROLL angle (R) are calculated for the right wrist angle (θ_4) and the left wrist angle (θ_5) by the following formulae:

$$P = \frac{1}{2} (\theta_5 + \theta_4) \qquad (1)$$

$$R = \frac{1}{2} (\theta_5 - \theta_4) \qquad (2)$$

Table 4.2.9.1.1

Conversion Factors Between Motor Steps and Joint Angles

Motor	Joint	Steps/Radian
1	Base	1125
2	Shoulder	1125
3	Elbow	672
4	Right Wrist	241
5	Left Wrist	241
6	Gripper	340

Algorithm

```
FWDJOINT
        begin
        for i = 1 to 6 do
                joint[i] = P[i]/table[i]
        PITCH = (joint[5] + joint[4])/2
        ROLL  = (joint[5] - joint[4])/2
        end
```

4.2.9.2 Forward Cartesian (FWDCART) Command

The FWDCART command accesses the system database to read the joint coordinates values of; BASE, SHOULDER, ELBOW, PITCH and ROLL. It then translates the joint coordinates into cartesian coordinates using trigonometry and updates the system database cartesian coordinates; X, Y and Z, which specify the position of the end point of the robot. The position of the end point is defined by the following three distances:

X The distance of the desired end point in from of the robot, measured from the base pivot along the X axis.

Y The distance of the desired end point to the left and right of the robot, measured from the base pivot along the Y axis.

Z The vertical height of the desired end point above the table top.

The orientation of the end point is defined by the PITCH angle P and the ROLL angle R.

Thus a unique point in the cartesian reference frame can be defined by specifying the position and orientation of the end point. i.e. (X,Y,Z,P,R).

To translate joint coordinates into cartesian coordinates the relationship between the different parts of the robot arm must be specified. This can be done in terms of the kinematic model shown in Figure 4.2.9.2.1. The kinematic model shows how each joint is articulated, how the joint angles are measured and the distance between joints. The distances between the joints are indicated by the constants H,L and LL. H is the distance from the table top to the shoulder centerline, L is the distance from the shoulder joint to the elbow joint and elbow joint to wrist joint, and LL is the distance from the wrist joint to the centre point between the two fingers. Values for these distances are given in Table 4.2.9.2.1.

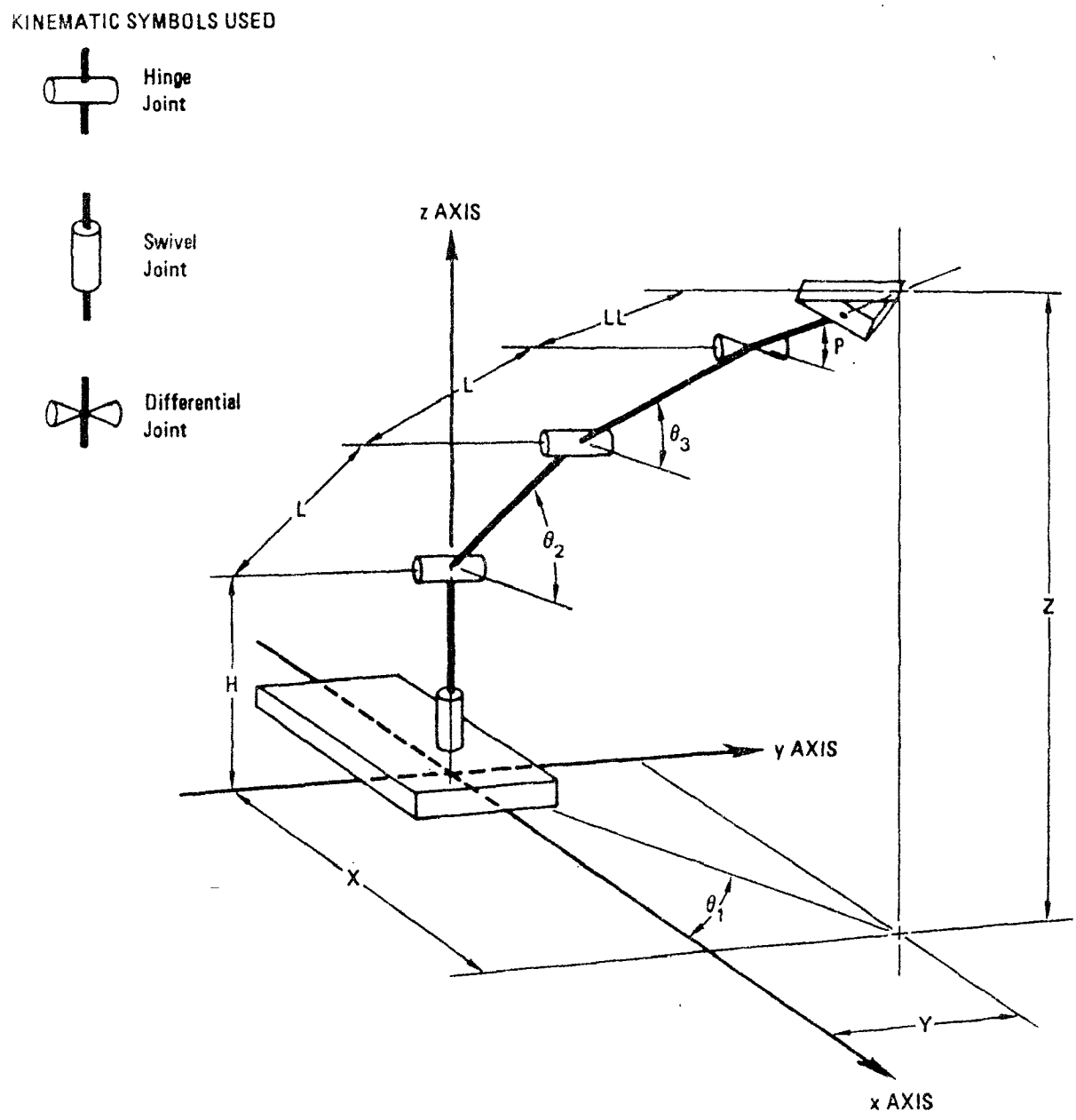


Figure 4.2.9.2.1 Kinematic Model of the MINI-MOVER 5

Table 4.2.9.2.1

Lengths of MINI-MOVER 5 Segments

Segments	Length (mm)
H	195.0
L	177.8
LL	96.5

The first step is to determine Z, the verical height of the end point, and an intermediate variable RR, the horizontal distance from the base pivot to the end point. The situation is summarised in Figure 4.2.9.2.2. Summing the vertical contributions from each link gives the following expression for Z:

$$Z = H + L \sin\theta_2 + L \sin\theta_3 + LL \sin P \quad (3)$$

Summing the horizontal contributions gives:

$$RR = L \cos\theta_2 + L \cos\theta_3 + LL \cos P \quad (4)$$

The second step is to determine the X and Y coordinates of the end point from the intermediate varaible, RR, as shown in Figure 4.2.9.2.3. By inspection, the coordinates are:

$$X = RR \cos\theta_1 \quad (5)$$

$$Y = RR \sin\theta_1 \quad (6)$$

Algorithm

```
FWDCART
      begin
      RR = L cos(SHOULDER) + L cos(ELBOW) + LL cos(PITCH)
      X = RR cos(BASE)
      Y = RR sin(BASE)
      Z = H + L sin(SHOULDER) + L sin(ELBOW) + LL sin(PITCH)
      end
```

where Pitch angle P is given by

$$P = \frac{1}{2}(\theta_5 - \theta_4)$$

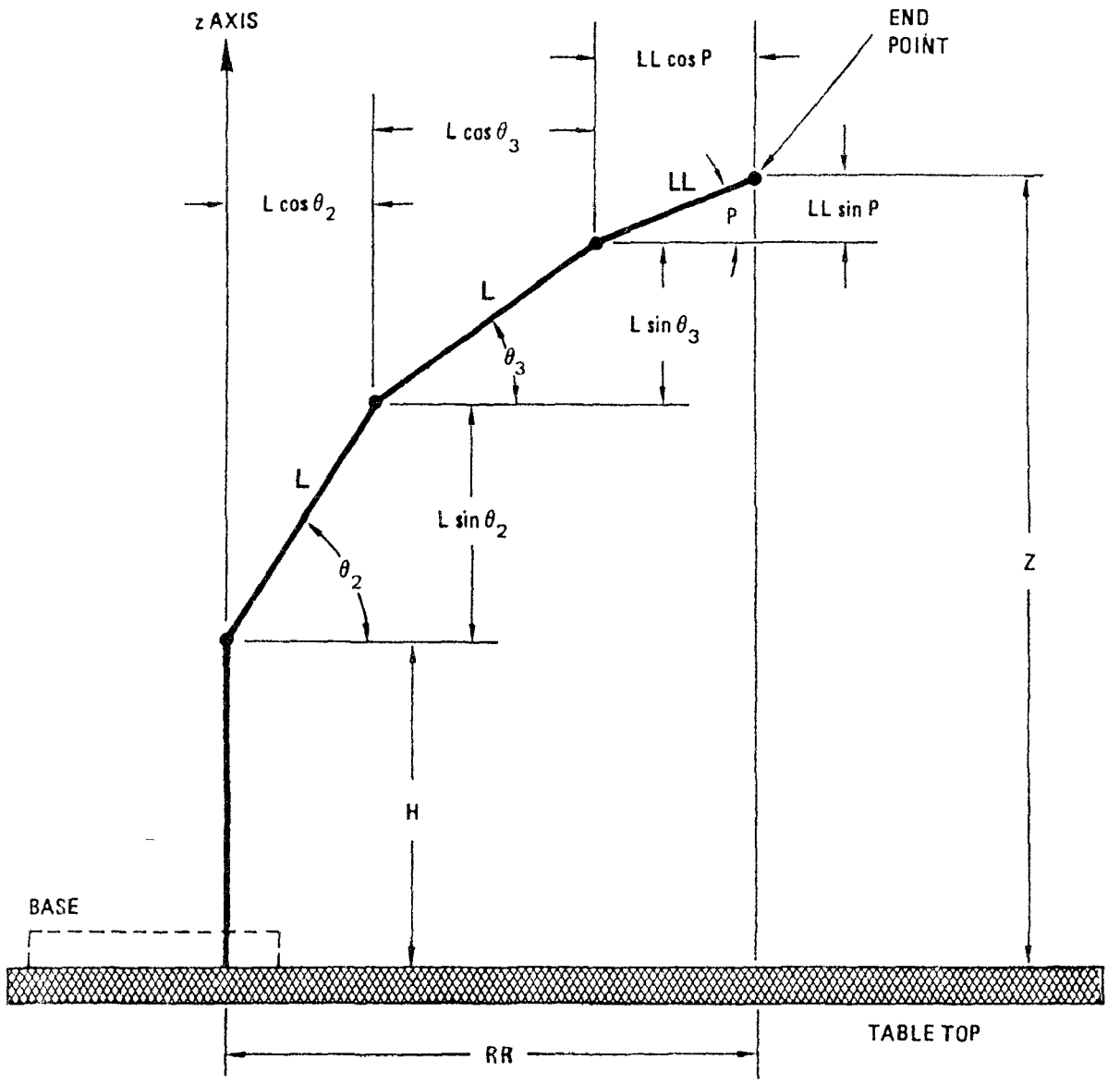


Figure 4.2.9.2.2 Side View of Kinematic Model

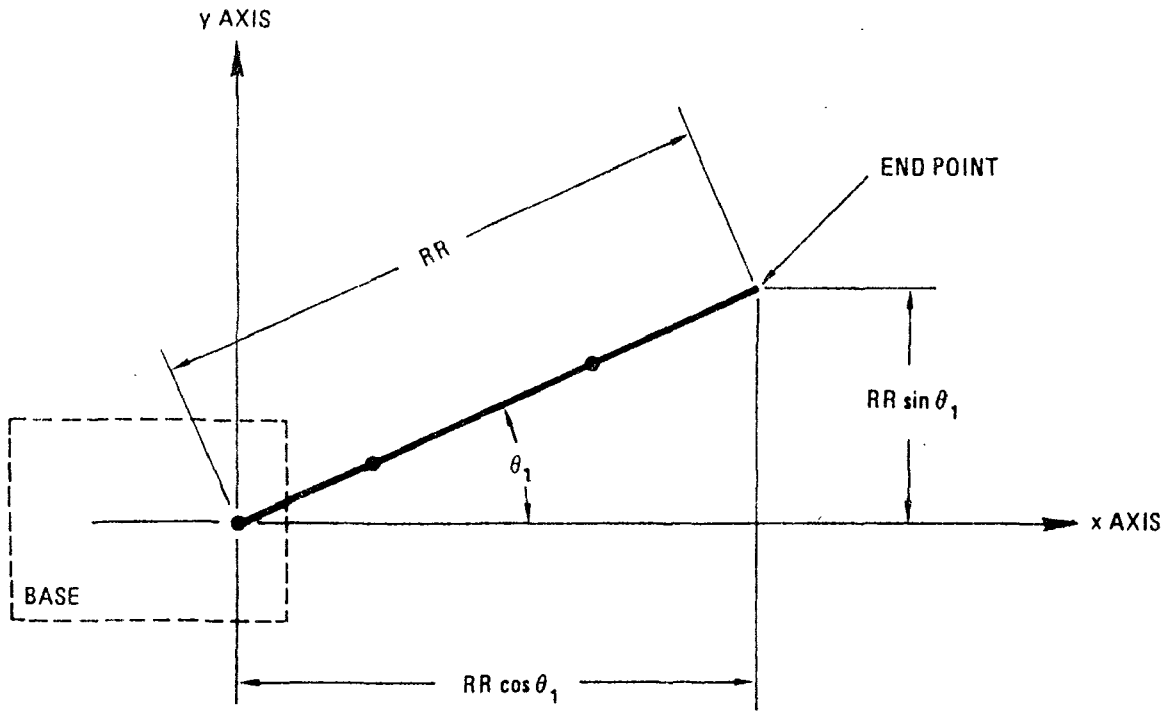


Figure 4.2.9.2.3 Top View of Kinematic Model

4.2.10 Backward Translation Commands

The backward translation commands consist of two commands; the Backward Cartesian (BWDCART) translation command and the Backward Joint (BWDJOINT) translation command.

4.2.10.1 Backward Cartesian (BWDCART) Command

The BWDCART command moves the MINI-MOVER 5 to a point with a specific position, orientation and gripper opening in the cartesian reference frame. The destination point (X,Y,Z,P,R,G) of the move is supplied as a parameter to the BWDCART command. The supplied parameter is translated by trigonometry into joint coordinate values, and these values are supplied as parameters to the BWDJOINT command which translates the joint coordinates into positional register values. The positional register values are then passed as parameters to the STEP command which moves the MINI-MOVER 5 end point to the location described by (X,Y,Z,P,R,G). Note: The gripper opening G argument is passed directly as a parameter to the BWDJOINT command.

The first step of the backward cartesian solution is to determine the base angle (θ_1) and the radius vector RR from the base to the end point as in Figure 4.2.10.1.1. Using the Pythagorean Theorum

$$RR = \sqrt{X^2 + Y^2} \tag{7}$$

$$\theta_1 = \arctan(Y/X) \tag{8}$$

The second step of the solution is the translation of the Roll angle (R) from the cartesian frame to the joint frame i.e. Roll in the cartesian frame is interpreted as the gripper angle to the Y-axis, while Roll in the joint frame is interpreted as the gripper angle to the wrist axis. See Figure 4.2.10.1.2. By subtracting the base angle (θ_1) from the commanded Roll angle, keeps the gripper orientation fixed along the Y-axis as the robot is moved.

$$R = R - \theta_1 \tag{9}$$

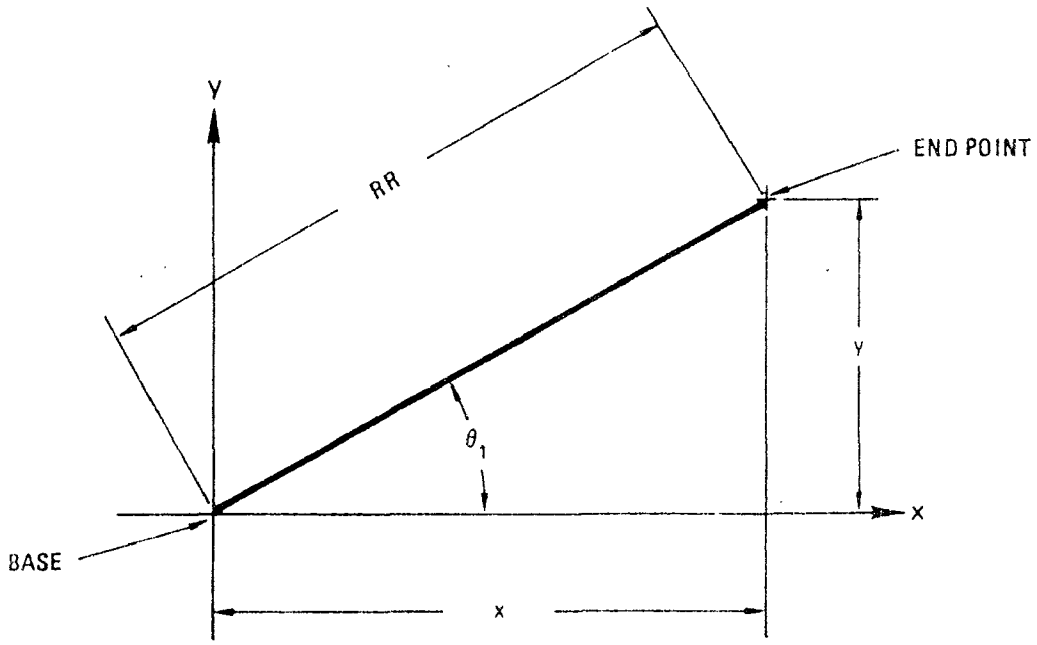
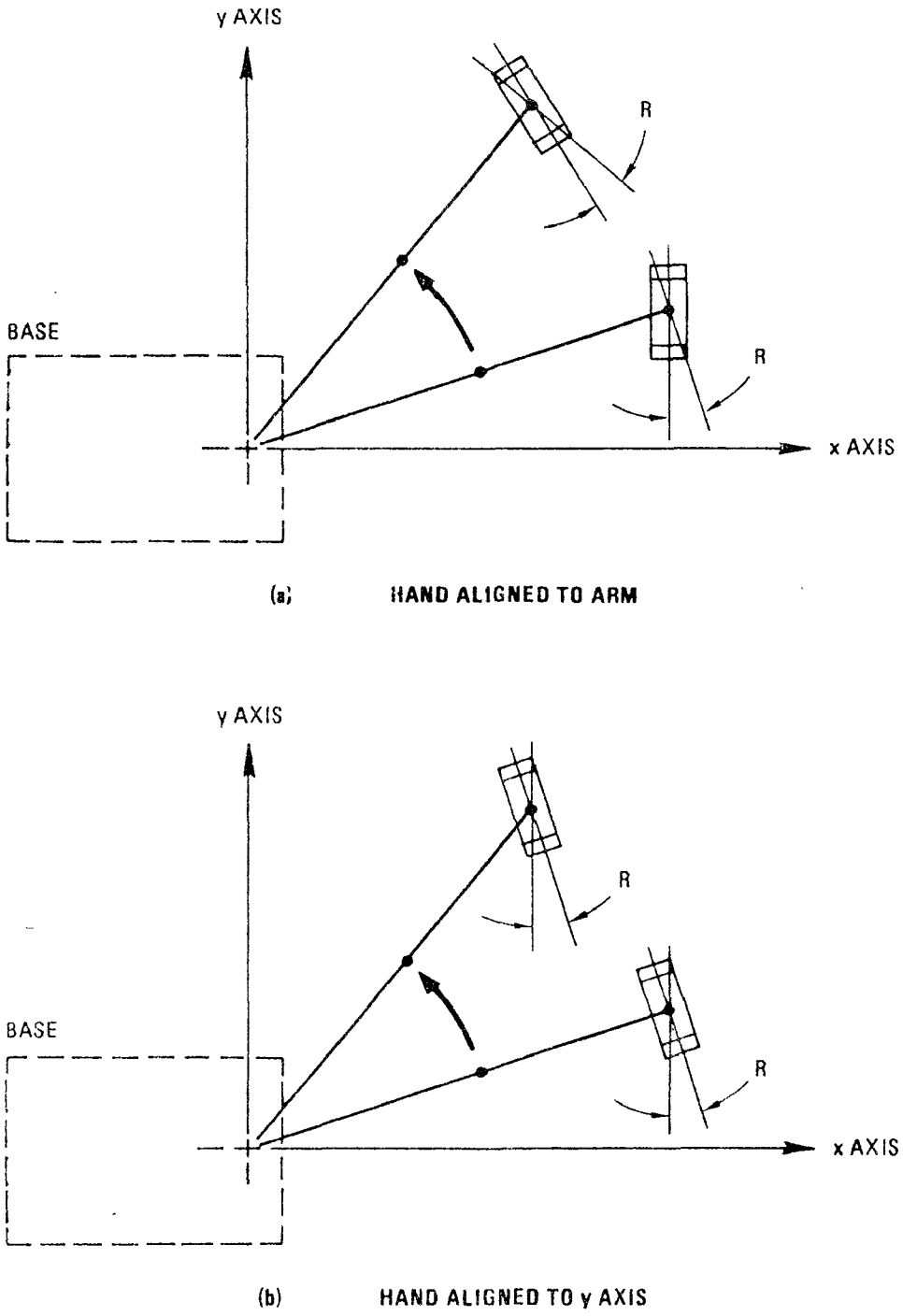


Figure 4.2.10.1.1. Top view of Arm



igure 4.2.10.1.2. Roll in Joint Frame (a) and Cartesian Frame (b)

The third step is to work back from the coordinates of the end point to those of the wrist. As in the forward translation the side view of the kinematic model shown in Figure 4.2.9.2.2 is used. Distances in this view are measured vertically along the Z-axis and horizontally along the radius from the base (R axis). Letting RE and ZE be the coordinates of the end point in this plane, we can calculate the coordinates of the wrist RW and ZW by using triangulation. The coordinates of the wrist are:

$$RW = RE - LL \cos P \quad (10)$$

$$ZW = ZE - LL \sin P \quad (11)$$

The fourth step is to define the shoulder-elbow-wrist triangle so that the shoulder angle θ_2 and the elbow angle θ_3 can be determined. For this purpose the translated coordinate system in Figure 4.2.10.1.3 is used. The origin is (0,0) at the shoulder and the coordinates of the wrist are now (R0,Z0).

The distance from the shoulder to the wrist R0 is the same as RW previously determined in equation (10). This is expressed as:

$$R0 = RE - LL \cos P \quad (12)$$

The height of the wrist above the shoulder Z0 is just the height of the wrist above the table top ZW, less the height of the shoulder H. Thus,

$$Z0 = ZW - H \quad (13)$$

Substituting for ZW using equation (11) gives:

$$Z0 = ZE - LL \sin P - H \quad (14)$$

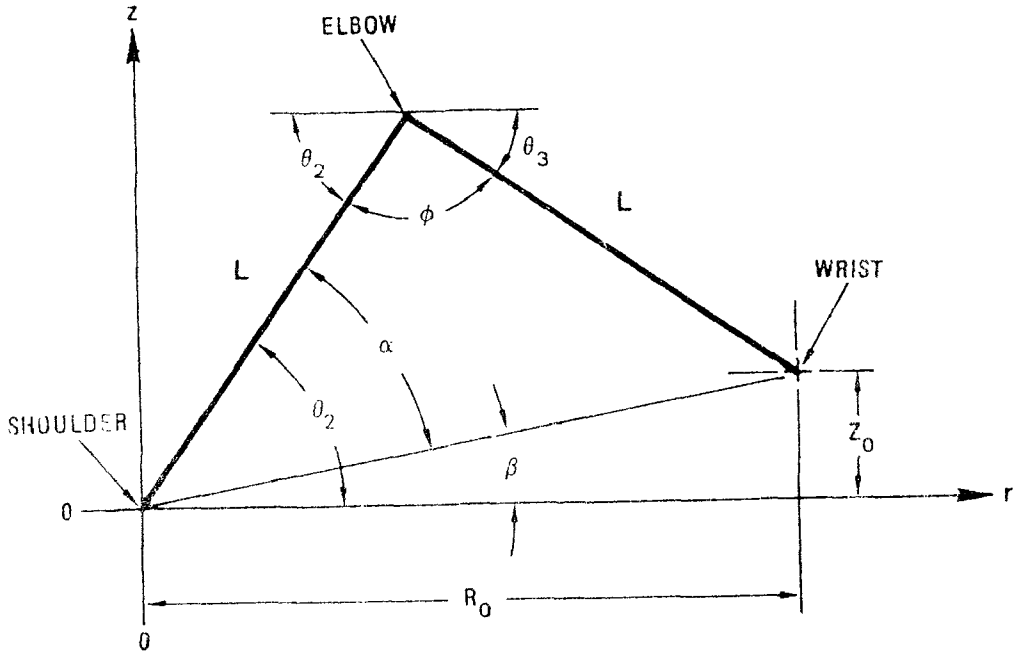


Figure 4.2.10.1.3 Shoulder-Elbow-Wrist Triangle

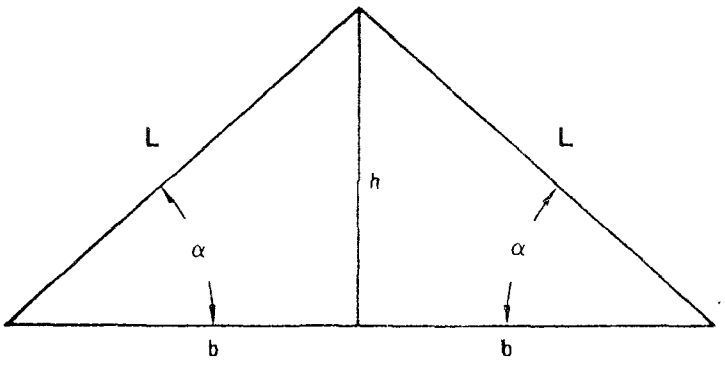


Figure 4.2.10.1.4 Simplified Triangle

The fifth step is to solve the shoulder-elbow-wrist triangle for θ_2 and θ_3. Three new angles α, β, and ϕ, are introduced to simplify the solution. These three angles must be solved first.

Since tan(β) = (Z0/R0), then:

$$\beta = \arctan(Z0/R0) \tag{15}$$

Pivoting the shoulder-elbow-wrist triangle about the shoulder by β gives the simplified triangle shown in Figure 4.2.10.1.4. The simplified triangles can be partitioned into two right angle triangles. The length of each base base b, is half that of the total base, or:

$$b = \frac{1}{2}\sqrt{Z_0^2 + R_0^2} \tag{16}$$

The height h is

$$h = \sqrt{L^2 - b^2} \tag{17}$$

Since the tangent of A is h/b,

$$h = \arctan h/b \tag{18}$$

Substituting for h in equation (18) using equation (22) gives

$$\alpha = \arctan \frac{\sqrt{L^2 - b^2}}{b} \tag{19}$$

Substituting for b in equation (19) using equation (16) gives

$$\alpha = \arctan \sqrt{\frac{4\ L^2}{R_0^2 + Z_0^2} - 1} \tag{20}$$

The sixth step is to use α and β to determine θ_2 and θ_3. The following three relations are set up and solved. At the shoulder (see Figure 4.2.10.1.3)

$$\theta_2 = \alpha + \beta \qquad (21)$$

At the elbow apex (see Figure 4.2.10.1.3)

$$\theta_2 + \phi + \theta_3 = 180 \text{ degrees} \qquad (22)$$

Summing the internal angles of the simplified triangle (see Figure 4.2.10.1.4) gives

$$\phi = 180 - 2\alpha \qquad (23)$$

Substituting the value of θ_2 from equation (21) and the value of ϕ from equation (23) gives

$$\theta_3 = \alpha - \beta$$

thus completing the backward cartesian translation solution.

Algorithm

```
BWDCART(X,Y,Z,PITCH,ROLL,GRIPPER)
      begin
      BASE = arctan(Y/X)
      RR = sqrt(sqr(X) + sqr(Y))
      ROLL = ROLL - BASE
      R0 = RR - LL cos(PITCH)
      Z0 = Z - LL sin(PITCH) - H
      beta = arctan(Z0/R0)
      alpha = arctan(sqrt(4 sqr(L)/(sqr(R0) + sqr(Z0)) - 1))
      SHOULDER = alpha + beta
      ELBOW = beta - alpha
      BWDJOINT(BASE,SHOULDER,ELBOW,PITCH,ROLL,GRIPPER)
      end
```

4.2.10.2 Backward Joint (BWDJOINT) Command

The BWDJOINT command moves the MINI-MOVER 5 to a point with specific joint orientations and gripper opening in the joint frame. The destination location (BASE,SHOULDER,ELBOW,PITCH,ROLL,GRIPPER) of the move is supplied as a parameter to the BWDJOINT command. The supplied parameters are check for bounds violation . If the supplied parameters do not violate bounds they are then translated by conversion factors into position register values which are passed as parameters to the STEP command which moves the MINI-MOVER 5 end point to the location specified by (BASE,SHOULDER,ELBOW,PITCH,ROLL,GRIPPER).

The first step of the backward joint solution is to find the right wrist angle (θ_4) and the left wrist angle (θ_5) from the PITCH and ROLL angles. This can be done by using equations (1) and (2) giving

$$\theta_4 = P + R \qquad (24)$$

$$\theta_5 = P - R \qquad (25)$$

The second step is to check if any of the supplied parameter arguments will violate the bounds of the robot i.e. cause overrun of the end points. This is done by issuing a BOUNDS command. Should any parameter violate a bound then no motion is undertaken and an error status is returned to the issuer of the BWDJOINT command.

The final step is to translate the joint coordinates and the gripper opening into positional register values, to be passed as arguments to the STEP command. This is done by multiplying the joint coordinates and the gripper opening by the conversion factors in Table 4.2.9.1.1.

Upon completion of the translation the STEP command is issued to move the robot to the desired location.

Algorithm

```
BWDJOINT(BASE,SHOULDER,ELBOW,PITCH,ROLL,GRIPPER)
        begin
        J[4] = PITCH + ROLL
        J[5] = PITCH - ROLL
        for i = 1 to 6 do
                J[i] = joint[i] x table[i]
        STEP(D,J1,J2,J3,J4,J5,J6)
        end
```

4.2.11 Manual Set Commands

The manual Set commands display a menu on the terminal console and place the MINI-MOVER 5 (depending on the Set command selected) into one of three manual control modes:

(1) Robot Control Mode - SET command.
(2) Joint Control Mode - JSET command.
(3) Cartesian Control Mode - CSET command.

The Set commands allow the user to invoke functions which move the robot and manipulate the system database, by pressing certain keys on the terminal console keyboard. If a function invocation results in motion of the robot, the FWDJOINT and FWDCART commands are issued. These commands update the system database with the new positional information of the robot. The new positional information is also displayed on the terminal console, this is done by issuing the STATUS command.

The interpretation of keystrokes is implemented by a finite state machine.

The Set commands have the same basic algorithm which is as follows:

Algorithm

SET

```
begin
display(menu)
repeat
        begin
        FWDJOINT
        FWDCART
        STATUS
        end
until QUIT
end
```

Keyboard Interrupt Routine

```
begin
interprete(key)
if valid then
        begin
          function(key)
        if key = Q then
                  QUIT = true
          else
                  QUIT = false
        end
  else
          display('Invalid Input')
end
```

4.2.11.1 SET Command

The SET command places the MINI-MOVER 5 into "robot control mode". Robot motion is done by calling the STEP command with the appropriate parameters. The functions that can be invoked by a keystroke under the SET command are as follows:

I This key invokes the MINI-MOVER 5 and system database initialisation command called INIT. But before the command is invoked the user is asked whether initialisation is required or not. A reply of keystoke Y is taken as yes, all other input is ignored. The inclusion of the question is a precaution against accidental initialisation.

H This key invokes the HOME command which returns the MINI-MOVER 5 to the initialistion position.

V This key invokes the function to change the number of steps the MINI-MOVER 5 moves every time the STEP command is issued. The next keystroke is taken as the stepping rate. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent a step rate of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

D This key invokes the function to change the delay between motor steps of the MINI-MOVER 5 in the STEP command. The next keystroke is take as the delay. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent delay functions of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

+ This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be positive i.e. UP, CLOCKWISE etc. when the STEP command is invoked.

- This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be negative i.e. DOWN, COUNTER-CLOCKWISE etc. when the STEP command is invoked.

B This key invokes the function to issue the STEP command to turn the base stepper motor with the selected stepping rate, direction and delay.

S This key invokes the function to issue the STEP command to turn the shoulder stepper motor with the selected stepping rate, direction and delay.

E This key invokes the function to issue the STEP command to turn the elbow stepper motor with the selected stepping rate, direction and delay.

P This key invokes the function to issue the STEP command to pitch the gripper with the selected stepping rate, direction and delay.

R This key invokes the function to issue the STEP command to roll the gripper with the selected stepping rate, direction and delay.

G This key invokes the function to issue the STEP command to move the fingers of the gripper with the selected stepping rate, direction and delay.

C This key invokes the function to issue the CLOSE command which closes the fingers of the gripper with the selected delay.

Q This key invokes the function to terminate the SET command and to return control to the calling routine.

The menu displayed by the SET command is shown is Figure 4.2.11.1.1.

DATE _____ EXHIBIT NO. _____

PROJECT _____ SYSTEM _____ PROGRAM _____ PAGE _____ of _____

Video Display Terminal Format

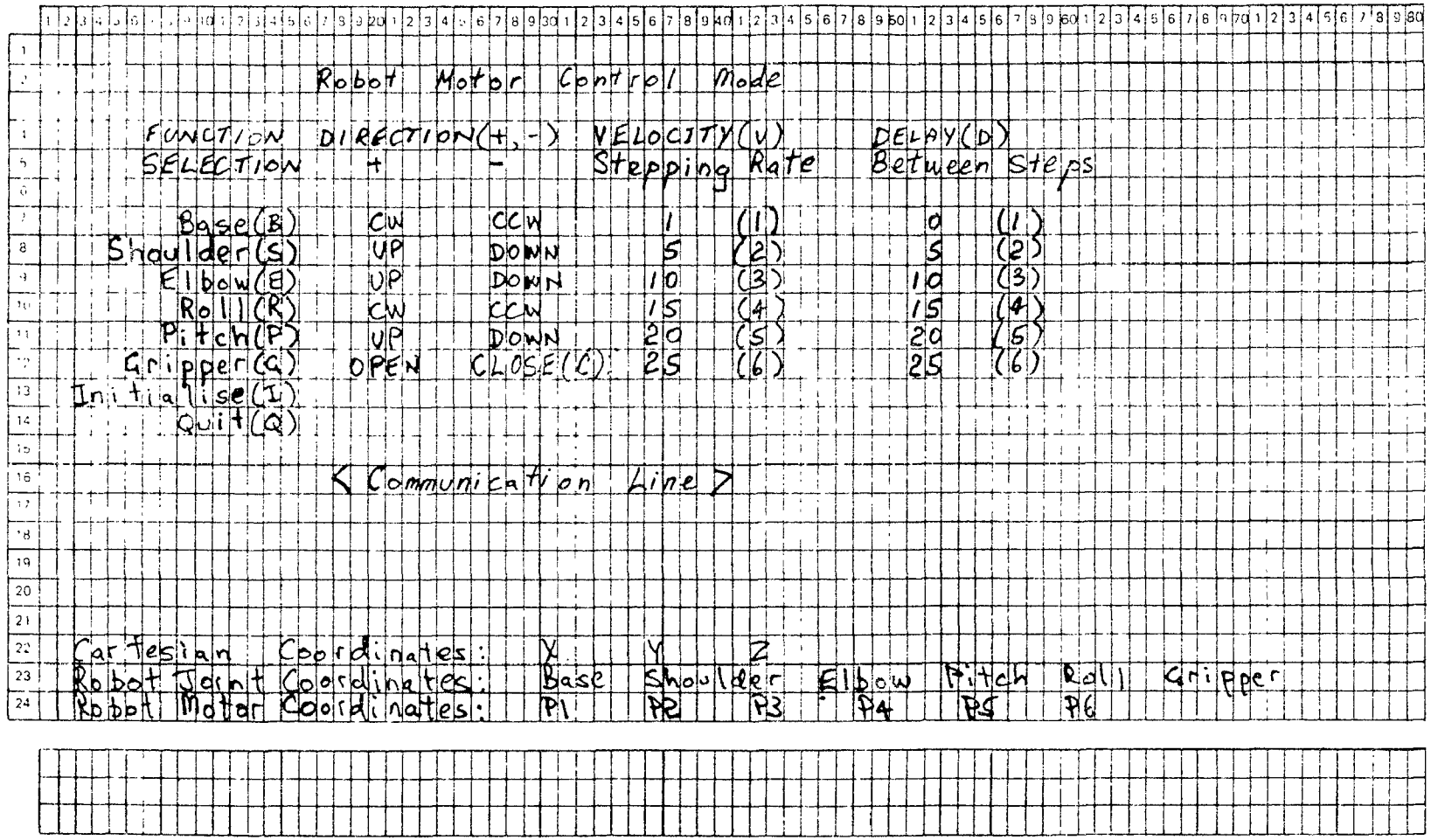


FIGURE 4.2.11.1.1 SET COMMAND SCREEN DISPLAY

REMARKS _____

4.2.11.2 JSET Command

The JSET command places the MINI-MOVER 5 into "joint control mode". Robot motion is done by calling the BWDJOINT command with the appropriate parameters. The functions that can be invoked by a keystroke under the JSET command are as follows:

I This key invokes the MINI-MOVER 5 and system database initialisation command called INIT. But before the command is invoked the user is asked whether initialisation is required or not. A reply of keystoke Y is taken as yes, all other input is ignored. The inclusion of the question is a precaution against accidental initialisation.

H This key invokes the HOME command which returns the MINI-MOVER 5 to the initialistion position.

V This key invokes the function to change:
- The number of degrees each joint of the robot moves
- The number of millimetres the gripper moves
every time the BWDJOINT command is issued. The next keystroke is taken as the degree/millimetre rate. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent a degree/millimetre rate of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

D This key invokes the function to change the delay between motor steps of the MINI-MOVER 5 in the BWDJOINT command. The next keystroke is take as the delay. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent delay functions of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

+ This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be positive i.e. UP, CLOCKWISE etc. when the BWDJOINT command is invoked.

- This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be negative i.e. DOWN, COUNTER-CLOCKWISE etc. when the BWDJOINT command is invoked.

B This key invokes the function to issue the BWDJOINT command to turn the base joint with the selected degree rate, direction and delay.

S This key invokes the function to issue the BWDJOINT command to turn the shoulder joint with the selected degree rate, direction and delay.

E This key invokes the function to issue the BWDJOINT command to turn the elbow joint motor with the selected degree rate, direction and delay.

P This key invokes the function to issue the BWDJOINT command to pitch the gripper with the selected degree rate, direction and delay.

R This key invokes the function to issue the BWDJOINT command to roll the gripper with the selected degree rate, direction and delay.

G This key invokes the function to issue the BWDJOINT command to move the fingers of the gripper with the selected millimetre rate, direction and delay.

C This key invokes the function to issue the CLOSE command which closes the fingers of the gripper with the selected delay.

Q This key invokes the function to terminate the JSET command and to return control to the calling routine.

The menu displayed by the JSET command is shown is Figure 4.2.11.2.1.

Video Display Terminal Format

DATE _____ EXHIBIT NO _____

PROJECT _____ SYSTEM _____ PROGRAM _____ PAGE _____ of _____

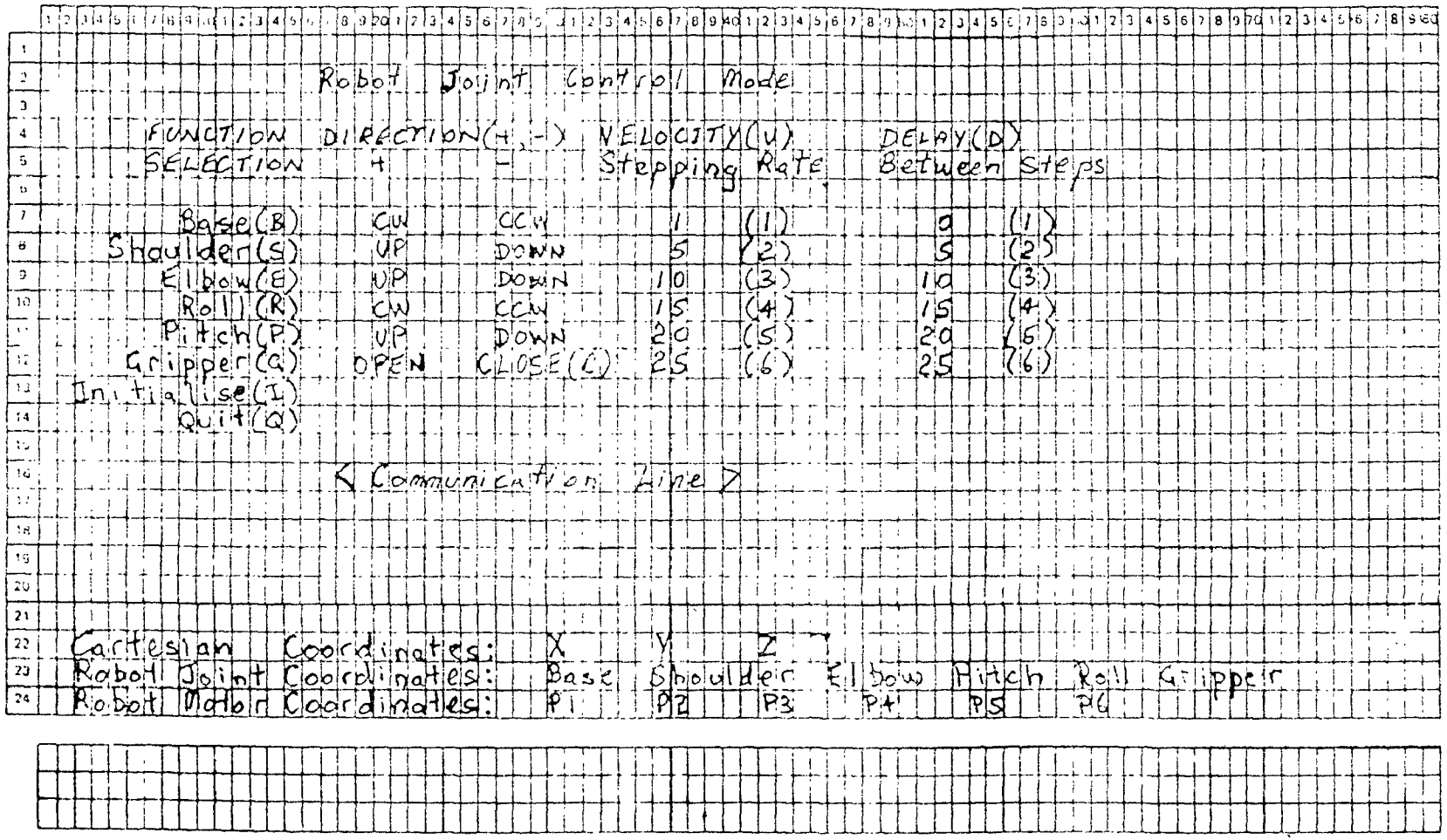


FIGURE 4.2.11.2.1 JSET COMMAND SCREEN DISPLAY

REMARKS _____

4.2.11.3 CSET Command

The CSET command places the MINI-MOVER 5 into "cartesian control mode". Robot motion is done by calling the BWDCART command with the appropriate parameters. The functions that can be invoked by a keystroke under the CSET command are as follows:

I This key invokes the MINI-MOVER 5 and system database initialisation command called INIT. But before the command is invoked the user is asked whether initialisation is required or not. A reply of keystoke Y is taken as yes, all other input is ignored. The inclusion of the question is a precaution against accidental initialisation.

H This key invokes the HOME command which returns the MINI-MOVER 5 to the initialistion position.

V This key invokes the function to change:
- The number of degrees the pitch and roll angles of the gripper move
- The number of millimetres the gripper moves
- The number of millimetres the robot moves in the X, Y and Z planes

every time the BWDCART command is issued. The next keystroke is taken as the degree/millimetre rate. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent a degree/millimetre rate of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

D This key invokes the function to change the delay between motor steps of the MINI-MOVER 5 in the BWDCART command. The next keystroke is take as the delay. Only keys 1,2,3,4,5, and 6 are valid keys. These keys represent delay functions of 1,5,10,15,20 and 25 respectively. Any other input generates a INVALID INPUT message on the terminal console.

+ This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be positive i.e. UP, CLOCKWISE etc. when the BWDCART command is invoked.

- This key invokes the function to set the direction of motion of the stepper motors in the MINI-MOVER 5 to be negative i.e. DOWN, COUNTER-CLOCKWISE etc. when the BWDCART command is invoked.

X This key invokes the function to issue the BWDCART command to move the robot in the X-plane with the selected millimetre rate, direction and delay.

Y This key invokes the function to issue the BWDCART command to move the robot in the Y-plane with the selected millimetre rate, direction and delay.

Z This key invokes the function to issue the BWDCART command to move the robot in the Z-plane with the selected millimetre rate, direction and delay.

P This key invokes the function to issue the BWDJOINT command to pitch the gripper with the selected degree rate, direction and delay.

R This key invokes the function to issue the BWDJOINT command to roll the gripper with the selected degree rate, direction and delay.

G This key invokes the function to issue the BWDJOINT command to move the fingers of the gripper with the selected millimetre rate, direction and delay.

C This key invokes the function to issue the CLOSE command which closes the fingers of the gripper with the selected delay.

Q This key invokes the function to terminate the CSET command and to return control to the calling routine.

The menu displayed by the CSET command is shown is Figure 4.2.11.3.1.

DATE _____ EXHIBIT NO. _____

Video Display Terminal Format

PROJECT _____ SYSTEM _____ PROGRAM _____ PAGE _____ of _____

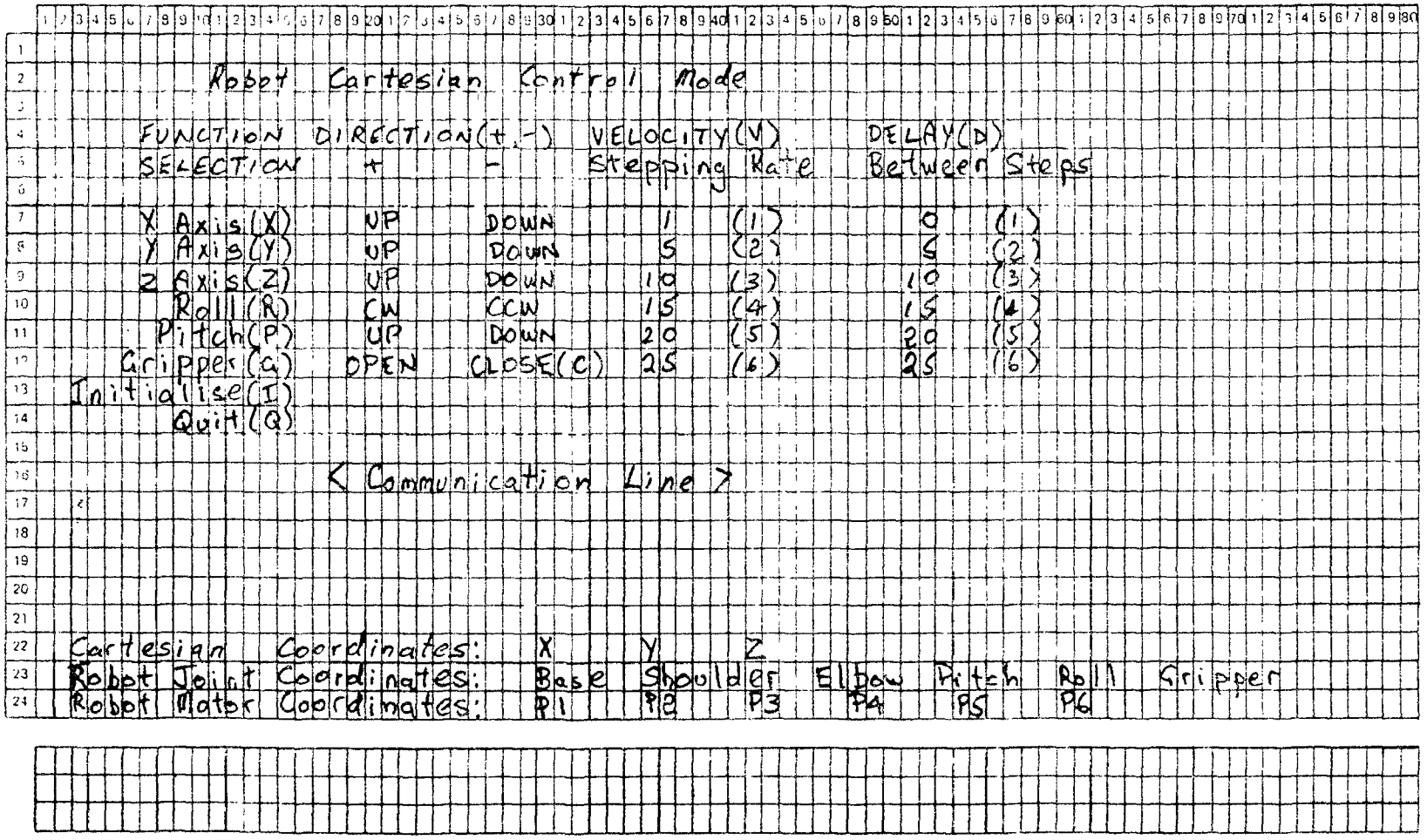
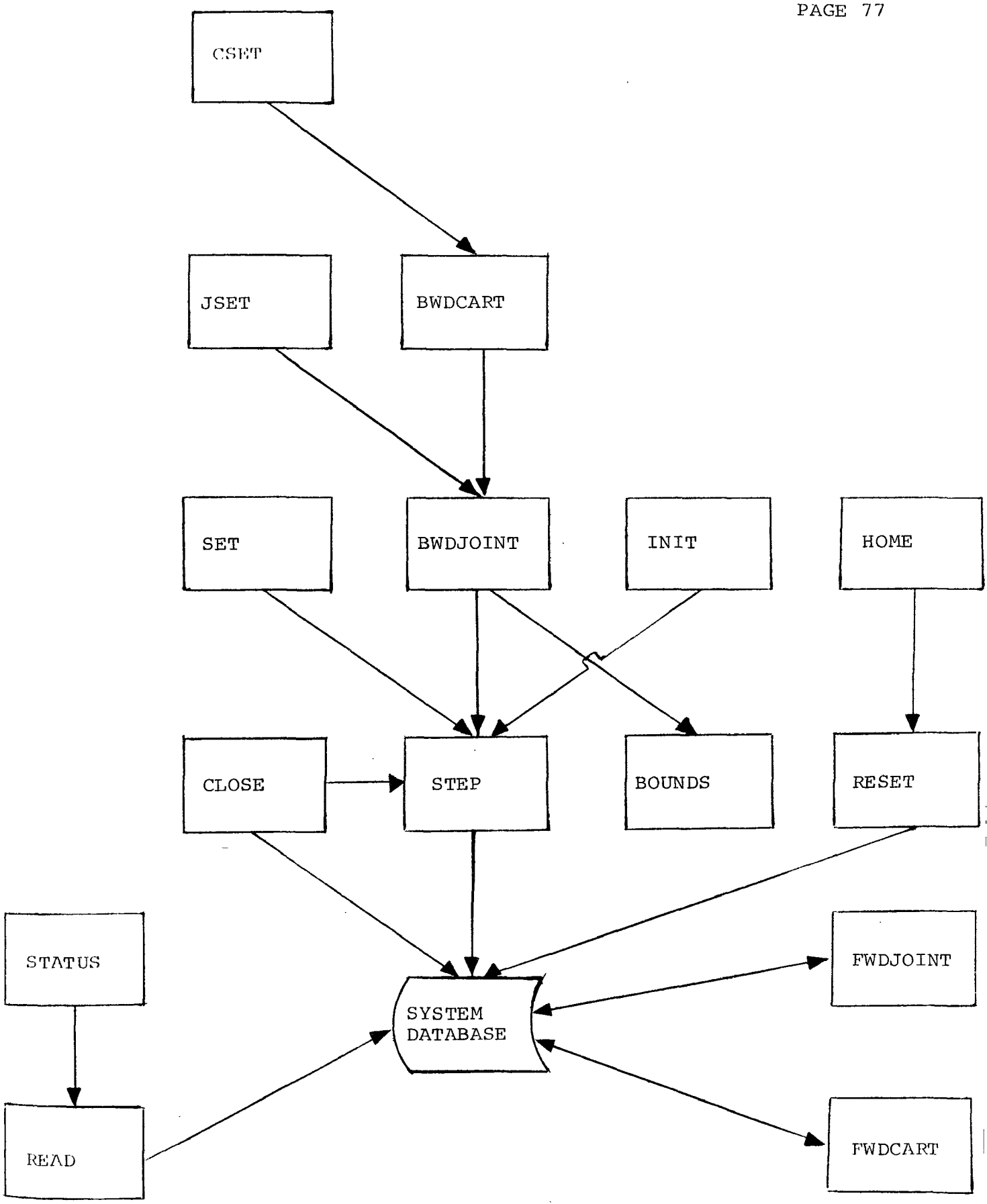


FIGURE 4.2.11.3.1 CSET COMMAND SCREEN DISPLAY

REMARKS _____

4.2.12 Interconnection of Robot Control Language Commands
4.2.12.1 Structure Chart

A structure chart has been drawn to show the interconnection of the robot control language commands. The structure chart appears on the following page.


CSET
JSET
BWDCART
SET
BWDJOINT
INIT
HOME
CLOSE
STEP
BOUNDS
RESET
STATUS
FWDJOINT
SYSTEM
DATABASE
READ
FWDCART

CHAPTER 5
PROJECT IMPLEMENTATION

5.1 Implementation Approach

The implementation approach was as follows:

(1) The interface code between the MINI-MOVER 5 and the M68000 would be written, tested and debugged. This code would be used as a test bed for the remainder of the software. The interface code involved the INIT and STEP commands. It was decided that this code would be implemented in M68000 assembler language, since a requirement of the project scope was that the robotic software had real time capabilities i.e. acceptable speed.

(2) The remaining software would be written, tested and debugged. It was decided that this code would be implemented in the C programming language.

5.2 Implementation Problems

In the course of implementation is was discovered that the M68000 cross-assembler was unusable for the following reasons:

(1) The cross-assembler did not give diagnostics for the C program code. Instead if a source error was detected the cross-assembler would crash, often with a core dump.

(2) The cross-assembler had no support library.

As a consequence of this problem it was decided that all the software would be written in M68000 assembler language. This decision required the provision of a support library, which also had to be written in M68000 assembler language. The support library would have to provide the following:

- Multiple precision arithmetic (Multiply, Divide, Add and Subtract).
- Trigonometric functions (Cosine, Sine and Arctangent).
- Mathematical functions (Square and Square root).

The project implementation was hampered by several problems, these were:

(1) To interface the MINI-MOVER 5 to a 8-bit I/O port on the M68000 microprocessor required some hardware modifications to the MINI-MOVER 5. The modifications were set out in the "MINI-MOVER 5 Reference Manual" which accompanied the robot. It was discovered that the numbering of the pins in the manual for the modifications was wrong. The reference manual has since been updated with the corrections to the pin numbering.

(2) During the course of implementation of the interface code it was discovered that the output timing signals provided in the reference manual were wrong. It was found that the output strobe (OUT) could not be sent low at the same time as the selection of the data/address lines occurred. The problem was remedied by sending the output strobe low after the data/addressing lines had been selected. The reference manual has since been updated with the new timing signals.

(3) During the course of implementation of the interface code it was also discovered that the stepper motor addresses provided in the reference manual were incorrect. The motor addresses instead of being (001)-base, (010)-shoulder, (011)-elbow, (100)-right wrist, (101)-left wrist and (110)-gripper they were all offset by one i.e (000)-base, (001)-shoulder,...,(101)-gripper. The reference manual has since been updated with the new timing signals.

(4) It was also found that the reference manual had other discrepancies, particularly in regard to MINI-MOVER 5 specifications. It was quite common to find the robot dimension information incorrect.

Note

The algorithms used in the implememention of the support library routines are provided in Appendix B. The source code for the project software is provided in Appendix C.

CHAPTER 6
RECOMMENDATIONS

6.1 Possible Future Enhancements

Listed below is a suggested set of possible enhancements to the robotic software developed for this project.

(1) The MINI-MOVER 5 has provision for the addition of four sensors. This facility could be used to add positional, or limits sensors on the MINI-MOVER 5. Such sensors would remove much of the load from the positional information software routines. The addition of tactile sensors could be benifical since they would provide information about the robot's environment. The software required to monitor the sensor inputs from the robot would be minimal, the software would be very similar to the software developed for the CLOSE command, which controls the closing of the gripper by monitoring the gripper sensor switch.

(2) To make the MINI-MOVER 5 a powerful robot which can monitor and react to changes in its environment, a robot vision system could be implemented. Although the system would be simple and crude, because of the computational restrictions of the M68000 microprocessor, a simple template matching system could be implemented. The vision system would be independent of the robot control language and as such require no changes to the existing software.

(3) In the study of robotic software in use today it was noted that debugging tools, particularly off-line debugging tools are a very important and desirable feature in "good" robotic software. The robotic software written for this project provides few debugging tools apart from the debugging aids associated with assembly languages. A useful enhancement would be provision of a debugging aid, paricularly a off-line debugging aid which uses computer graphics to simulate the robot and its environment. Such a debugging system could be implemented on the host computer (Perkin-Elmer 3220) where the applications are developed. Before the applcation code is down-line loaded, the code could be debugged using the aid. An off-line debugging tool would not require any changes to existing software.

(4) A useful enhancement to the robotic software would be the provision of up-line loading facilities to the host computer (Perkin-Elmer 3220). Such a feature would be useful for programs in which the the robot is taught a task, and it is desired to save the information associated with the control of this task. This information must be passed back to the host because the M68000 microprocessor does not have non-volatile memory. An up-line loading enhancement would require additional software, which reads the system database and communicates with a task running in the host computer.

CONCLUSION

The aim of the project was to design and implement software on a MOTOROLA M68000 16 bit microprocessor to control a MINI-MOVER 5 robot arm.

This required software which controlled the robot via a robot control language, which was easy to use and transparent to the user. A design requirement of the software was to allow extensibility of the software by the user. The software was to be a tool for writing robot application programs.

The software has been written and debugged. The project design proved to be a success. The project as presented fulfills the given specifications, and provides a successful method of computer control of a robot. The software implemented in this project enables the programming of the MINI-MOVER 5 robot to do meaningful tasks.

ACKNOWLEDGEMENTS

Sincere thanks must be extended to a number of people who assisted in the development of this project:

* Mr P.J. McKerrow, project supervisor, for his valuable thoughts and for his willingness to discuss any problems.

* Messers J. Fulcher and M. Millway, professional officers, whose hardware expertise solved many problems.

* Mrs G. Zelinsky, my wife, for her continued support and encouragement.

BIBLIOGRAPHY

1. K. Hivany, "Computer Arithemtic, Principles, Architecture and Design".

2. W. Barden Jnr, "How to Program Microcomputer".

3. N. Graham, "Microcomputer Programming for Computer Hobbyists".

4. L.A. Leventhal, "6809 Assembly Language Programming".

5. F.G. Duncan, "Microprocessor Programming and Software Development".

6. R. Zaks and W. Labiak "Programming the 6809".

7. Motorola, "MC68000 16-bit Microprocessor User's Manual"

8. R.G. Dromey, "How to Solve it by Computer".

9. K. Takase, R.P. Paul and E.J. Berg, "A Structured Approach to Robot Programming and Teaching", IEEE Transactions on Systems, Man and Cybernetics April 1981.

10. H.R. Holt, "Robot Decision Making", Volume 1 "Introduction to Robotics".

11. G. Gini and M. Gini, "ADA a Language for Robot Programming ?", Computers in Industry No. 3 1982.

12. R.H. Taylor, P.D. Summers and J.M. Meyer, "AML a Manufacturing Language", International Journal of Robotics Research Vol. 1 No. 3 1982.

13. M.A. Lavin, L.I. Lieberman, "AML/V an Industrial Machine Vision Programming System", International Journal of Robotics Research Vol. 1 No. 3 1982.

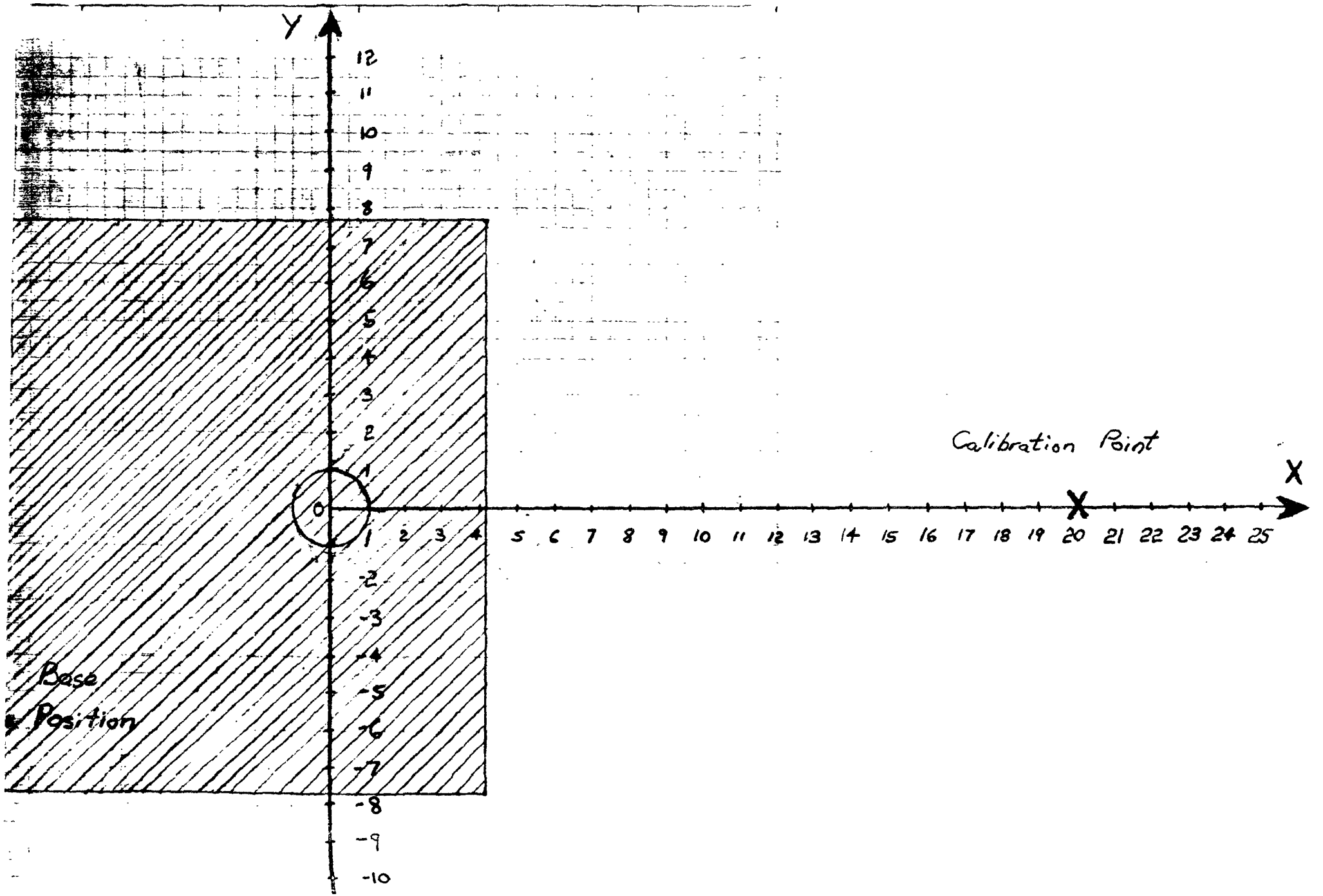
14. "Robots", Compressed Air, April 1982.

15. H.M. Morris, "Where Do Robots Fit in Industrial Control ?", Control Engineering, February 1982.

16. Microbot, "MINI-MOVER 5 Reference Manual"

APPENDIX A
Calibration Grid

The figure presented on the following page is a calibration grid which can be copied or removed from this report to facilitate calibration and initialisation of the MINI-MOVER 5. The robot should be placed on this grid such that the base is in the position indicated. The robot should then be moved so that the gripper is closed and the finger tips are aligned with the calibration mark shown in the figure.


Y
12
11
10
9
8
7
6
5
4
3
2
1
0
-2
-3
-4
-5
-6
-7
-8
-9
-10
X
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Calibration Point
Base Position

APPENDIX B
Support Library Algorithms

The algorithms set out below, are the ones used in the implementation of the support library for the robotic software project. The algorithms can be categorised into three classes:

- Multiple Precision Arithmetic Functions
- Trigonometric Functions
- Mathematical Functions

B.1 Multiple Precision Arithmetic Algorithms

It was determined that to give accurate results in the translation commands of the project software, required 6 decimal places of precision in the arithmetic functions. It was decided to use a fixed point representation for the floating point numbers. It was calculated that using 32-bit words to store the floating point numbers, required a 8.24 representation (i.e. 8-bit integer and 24-bit fraction) to attain the required precision of numbers. Such a representation was acceptable since the numbers stored using this format were angles using a radian representation.

The multiplication function consists of multiplying two 32-bit numbers together and returning a 64 bit result. This function can be implemented by using the Motorola M68000 MUL assembler instruction which multiplies two 16-bit numbers and returns a 64-bit result. The 32-bit multiplication requires four 16-bit multiplications in order to obtain the correct result. This process is developed by using the rules of factoring and associativity.

Algorithm

```
MULTIPLY(N1,N2)
        begin
        x = LOWWORD(N1)
        y = LOWWORD(N2)
        result = MUL(x,y)
        x = HIGHWORD(N1)
        y = LOWWORD(N2)
        partialsum = MUL(x,y)
        result = result + partialsum
        x = LOWWORD(N1)
        y = HIGHWORD(N2)
        partialsum = MUL(x,y)
        result = result + partialsum
        x = HIGHWORD(N1)
        y = HIGHWORD(N2)
        partialsum = MUL(x,y)
        result = result + partialsum
        return(result)
        end
```

Divison is performed by subtracting the largest possible multiple of the divisor from the left most digits of the dividend. The new dividend is the result of the subtraction of a multiple of the divisor from the the dividend. The multiplier of the of the divisor becomes the next digit of the quotient. The remainder is the result of the last subtraction. Binary division is performed exactly the same way as is decimal division.

Algorithm

```
DIVIDE(dividend,divisor)
        quotient = 0
        counter = 32
        repeat
                begin
                if left(dividend) > divisor then
                        begin
                        left(dividend) = left(dividend)
                                         - divisor
                        quotient = quotient + 1
                        end
                counter = counter -1
                end
        until counter = 0
        return(quotient,dividend)
        { remainder in dividend }
        end
```

B.2 Trigonometric Algorithms

It was determined that the project software, required the sine, cosine and arctangent trigonometric functions.

The most common method used by many systems to implement trigonometric functions is to use a fitted polynomial to the desired function. Calculating polynomials is straight forward. One simply supplies the coefficients for the required function. The calculation for the polynomial is as follows:

```
P(z) = a0 + a1*z + a2*z*z + a3*z*z*z + ...
     ..... + an*z*....*z (z to the n)
```

this can be broken down to:

```
P(z) = a0 + z*(a1 + z*(a2 + z*(a3 + ....z*(an)...)))
```

which reduces the number of multiplies and which can be encoded as:

Algorithm

```
TRIG(z)
        begin
        p = an
        for i = n-1 to 0 do
                begin
                p = p*z + a(i)
                end
        return(p)
        { p = desired polynomial on exit }
        end
```

The sine and cosine functions are implemented using fitted polynomials. The table of coefficients for these functions is given below. Note: All coefficients not listed are zero.

Table of Coefficients

function	(1/X)*SIN(X)	COS(X)
argument range	0 <=X <= pi/2	0 <=X <= pi/2
a0	1	1
a2	-0.1666666664	-0.4999999963
a4	0.0083333315	0.0416666418
a6	-0.0001984090	-0.0013888397
a8	0.0000027562	0.0000247609
a10	-0.0000000239	-0.0000002605

The arctangent function can not be implemented efficiently using the fitted polynomial technique, since it is not a repeating polynomial. Instead another method is used, which is the Cordic technique. The Cordic technique was developed by Henry Briggs in 1624. This algorithm is easily implemented of any computer which has access to "bit fiddling". The algorithm consists basically of splitting the supplied argument into values for which the corresponding arctangent function value is known. The values are split into powers of two, since they can easily be manipulated on a computer. The algorithm is as follows:

Algorithm

```
ARCTANGENT(X)
        begin
        Y = 1.0
        Z = 0.0
        J = 0
        repeat
                begin
                Xold = X
                X = X - Y*power(2,-J)  { Shift op. }
                if X >= 0.0 then
                        begin
                        Y = Y + Xold*power(2,-J)
                        Z = Z + atan(2,-J)
                        end
                  else
                        begin
                        X = Xold
                        J = J + 1
                        end
                end
        until J > 32   { 32-bits of precision }
        end
```

Note: power(x,n) is a function which raises x to the power n.
atan(x) is a lookup table of arctangents of powers of 2.

The lookup table of arctangents is given below.

Arctangent Table for Cordic Technique

i	power(2,-i)	arctan(power(2,-i))
0	1	0.7853981633
1	0.5	0.4636476090
2	0.25	0.2449786631
3	0.125	0.1243549945
4	0.0625	0.0624188100
5	0.03125	0.0312398334
6	0.015625	0.0156237286
7	0.0078125	0.0078123410
8	0.00390625	0.0039062301

Note: By examing the series it can bee seen that it is rapidly converging. For i > 8 arctanangent an be approximated by power(2,-j).

B.3 Mathmetical Algorithms

It was determined that the project software, required the square and squareroot mathematical functions.

The square function is straight forward and its algorithm is not listed. The square root function was implemented using a algorithm from "How to Solve it by Computer" by R.G. Dromey. The algorithm for this function is as follows:

Algorithm

```
SQUAREROOT(m)
        begin
        g2 = m/2
        repeat
                begin
                g1 = g2
                g2 = (g1 + m/g1)/2
                end
        until abs(g1 - g2) < error
        return(g2)
        end
```

APPENDIX C
Project Source Code

The source code listings for the software written to implement this project, appear on the following pages.