EXTERNAL CONTROL OF PUMA 700 SERIES ROBOT BASED ON THE COMMUNICATION PROTOCOLS LUN AND DDCMP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖNDER EMİN GEBİZLİOĞLU

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2003

Approval of the Graduate School of natural and Applied Sciences

_____
Prof Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof Dr. Mübeccel DEMİREKLER
Head of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

_____
Prof Dr. Aydın ERSAK
Supervisor

Examining Committee Members

Prof Dr. Erol KOCAOĞLAN _____

Prof Dr. Aydın ERSAK _____

Doç Dr. Aydın ALATAN _____

Yrd. Doç. Dr. Cüneyt BAZLAMAÇCI _____

Ediz ÇELİK, M.Sc. _____

ABSTRACT

EXTERNAL CONTROL OF PUMA 700 SERIES ROBOT BASED ON THE COMMUNICATION PROTOCOLS LUN AND DDCMP

GEBIZLIOĞLU, Önder Emin

Department of Electrical and Electronics Engineering
Middle East Technical University

Supervisor: Prof. Dr. Aydın ERSAK

December 2003

This thesis analyzes the supervisory control of the PUMA 700 series robot through a remote computer. Supervisory communication carries the control through MARK II controller, common controller for the PUMA robots, to a standard PC, enabling the development of purpose specific control programming without the knowledge of the VAL (Variable Assembly Language), robot-programming language. Using the supervisory communication feature of PUMA, new control software has been developed in which both VAL commands and interactive control commands can be executed simultaneously. The supervisory communication with the control software enables exploitation of third party applications and additional operating system features.

The supervisory communication uses the Digital Data Communications Message Protocol (DDCMP). The frame structure of data messages, which is specific to PUMA robots, is fitted into this protocol. The messages embedded into DDCMP are actually logical units, having different abilities and features. Data messages are formed with the interactive control software according to execution requests of the user.

This thesis explains the implementation of the communication without using the CRC (Cyclic Redundancy Checking) on the remote computer side and the data messages formed with the interactive control software, which also enables the use of sensory inputs (camera, infrared, sound, color information) to be used for the robot motion control.

Keywords: PUMA 700 Series, DDCMP, LUN (Logical Unit Number)

ÖZ

LUN VE DDCMP İLETİŞİM PROTOKOLLARI YOLUYLA PUMA 700 SERİSİ ROBOTUN DIŞARIDAN DENETLENMESİ

GEBIZLIOĞLU, Önder Emin

Elektrik - Elektronik Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi

Tez Danışmanı: Prof. Dr. Aydın ERSAK

Aralık 2003

Bu tez çalışmasında PUMA 700 serisi robotların uzaktan harici bir bilgisayar ve denetleyici haberleşme kullanılarak kontrol edilmesi analiz edilmiştir. Denetleyici ve kontrol amaçlı kullanılan bu haberleşme sayesinde, kontrol PUMA robotları için ortak olarak kullanılmakta olan MARK II kontrol sisteminden standart bilgisayarlara aktarılmıştır. Böylece PUMA robotları için geliştirilmiş olan VAL kontrol yazılımı bilgisine sahip olmadan, işe özel kontrol programlaması yapılabilmektedir. PUMA robotların denetleyici ve kontrol yeteneğine sahip haberleşme olanağına imkan vermesi sayesinde yeni bir kontrol programı geliştirilmiş olup, bu program ile istenirse VAL ve interaktif olarak hazırlanabilen kontrol komutları çalıştırılabilmektedir. Bu sistem aynı zamanda üçüncü taraf uygulamalarının ve işletim sistemi özelliklerinin robot üzerinde kullanılabilmesine olanak sağlamıştır.

Denetleyici kontrol haberleşmesi DDCMP (Data Communications Message Protocol) adlı protokolü kullanmaktadır. PUMA için özel olan data mesajları yapısı, DDCMP protokolünün içine giydirilmiştir. DDCMP içine oturtulan mesajlar esasen farklı özellik ve yeteneklere sahip olan mantıksal birimlerdir. Data mesajları, interaktif olarak, kullanıcı isteklerine göre oluşturulabilmektedir.

Bu çalışmada, harici bilgisayarda CRC (Cyclic Redundancy Checking) kullanılmadan yapılan haberleşme ve interaktif sistemin gerçekleştirilmesi açıklanmaktadır. Bu sayede farklı sensör (kamera, infrared, ses, renk bilgisi) bilgileri kullanılarak robot hareketini kontrol etmek mümkün olmuştur.

Anahtar Kelimeler: PUMA 700 Serisi, DDCMP, LUN (Logical Unit Number)

ACKNOWLEDGEMENTS

I would like to express my special thanks to Prof. Dr. Aydın ERSAK for his powerful guidance, courage, critics and supervision throughout this thesis work.

Also I would like to thank Mr. Benjamin CLERK, from RP automation, who has shared his experience on the robots and shown me the ways to get used to the PUMA robot operation and programming.

Special thanks to my friends in DELTA AEROSPACE who have given me courage and who have shared their ideas about my studies.

Finally I would like to thank my family for their great support that they have given throughout my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

1 INTRODUCTION

1.1 Introduction

Robots are used in a wide range of industrial applications. The earliest applications were in materials handling, spot welding, and spray painting. Robots in manufacturing can be applied to jobs that were hot, heavy, and hazardous such as die-casting, forging, and spot welding other than these robots can also be used in assembly operations, pick and place operations and material handling operations.

Robots are generally controlled by dedicated controllers. Previous industrial robotic control systems suffer from the use of outdated technology, although their mechanical structure did not change with years.

PUMA robots are in a common type of industrial robot arm. PUMA robots are not used within industrial applications and manufacturing anymore but they are used in academic studies mostly because of their agreeable price and capabilities.

A standard PUMA manipulator is controlled by a VAL-II based Unimation Mark II controller. There are several disadvantages or limitations of the VAL-II based Mark II controller system, many of which are attributable to the age of the product. To overcome some of these disadvantages, the proposed solution is to control the robot over an external computer.

Developments in fields such as microprocessors, vision, and artificial intelligence will be used to fulfill the needs of robot users, operators and mostly the manufacturing industry. Robotics technology is finding applications in many other fields as well, such as medicine and health care, space exploration, and transportation. In order to use the benefits brought by the new technology, old robots such as PUMA shall be modified, the modification shall enable the use of today's well-known computers.

There were two main motivating factors for the development of a well-known computer based PUMA robot manipulator control system: cost and flexibility. In this thesis, the external supervisory control studies and implementations on PUMA robot, our objectives, problems encountered, solutions and results obtained are explained.

Besides the objective of modifying the control system of PUMA robot the following are also among the objectives of the thesis:

- Obtaining knowledge of robot control, robotic applications and PUMA robot arm.
- Learning and practicing of PUMA robot VAL II commands.
- Controlling of PUMA externally from a well-known, easy to use and cost effective platform, an external computer.
- Excluding the VAL II software control system of PUMA.
- Enabling users to use different kind of sensors like camera, light & infrared detectors in order to control the robot.

While achieving the above objectives following outcomes are expected to be reached at:

- An operational robot (since it was malfunctioning in the beginning).
- Controlling and programming the robot arm without the knowledge of VAL II robot programming language (Detailed information of VAL II can be found at Chapter 2).
- A system, which is flexible and extendible for a variety of manipulators, sensors and other robot hardware.
- A system supporting control algorithm development using various sensory information.
- Programming the robot from a distance with today's modern computers without the dependency on OS. OS (Operating System) is the software that controls the execution of computer programs and may provide various services. Most popular operating systems of today are Linux, Windows and Unix.
- Enabling user to integrate and use OS applications, like speech processor applications, to robotic control.
- Simulating the robot enabling users to try dangerous robot movements in an off-line state.
- Attaching and using third party applications to control the robot such as speech processors, video imaging applications and tools.
- Enabling users to implement third party application programs (C, C++, Java) to control the arm movements of robots. Supervisory system shall enable the use of new data obtained from other applications in order to be used for the modification of the robot pose.
- Determining and executing movement paths without the need of VAL II calculations (users are available of determining movement paths).
- Opening a gate to control the robot within a LAN (Local Area Network, a local computer network for communication between computers) or even a WAN (a computer network that spans a wide area, covering LANs).

Before starting our studies on the supervisory control of the PUMA robot we have examined the previous studies related to our purpose. Although robotic related studies are held all over the world, the studies conducted for industrial robots like the PUMA robots are less in number when considered to those studies on new generation mobile robots like the LEGO robots which support the use of external control over the well-known computer systems. The new generation robots also provide support for direct use of external sensory information or the data obtained from the external sensors (camera input, voice/speech processors etc…) to be processed for the robot control.

When we consider the case of PUMA robots, which are the most available industrial robot arms in university laboratories, we have observed that most of the studies were conducted on NOKIA / KAWASAKI manufactured PUMA 500 series robots which is controlled over a well-known computer host and the TEXAS Digital Signal Processor (DSP) based hardware. On the host computer the UNIX-like QNX real-time operating systems are used [20, 32-34]. The studies conducted for the PUMA manipulators that are manufactured or modified by the NOKIA Company were actually out of our scope since Unimate manufactured MARK II controllers have their own characteristics, which do not let the use of external computers easily.

Several approaches towards obtaining an effective control of the PUMA robot with a MARK II controller were developed in the studies conducted by Trident Robotics and Research, Inc. [21, 23, 32, 33, 34], Carnegie Mellon and Stanford Universities. They have produced the TRC004 PUMA interface board, which allows direct access to PUMA joint positions and torques. The hardware, thus the robot is controlled by a special purpose real time operating system called Chimera. The Quality Real Time Systems (QRTS) Company has developed a similar work in order to control the PUMA 560 robot. QRTS Simulink Robotic Toolkit [22, 31] has been implemented for the Puma 560, but again hardware modification is required for the operation of this toolkit. Other than those, simulation and off-line programming studies has been conducted for PUMA robots [23-24, 26, 30], but these simulation packages do not have the ability to control the robot, since this requires an interface (serial port) to be used and communication protocols to be implemented.

A similar study that used DDCMP and logical units in order to control the PUMA arm was held [35], which had limited capabilities since it aimed to control the robot arm joints one at a time and it was not completely visualized as well. Logical units, which are explained in detail within chapter 3, were not completely implemented, thus control and programming capabilities were not totally obtained.

In the scope of this thesis study no hardware modifications are made on the arm's MARK II controller so that available communication capabilities [1-4,13] already present in PUMA are used. Other than those studies regarding the supervisory communication [24-26,35], we did not make any cyclic redundancy checking for communication error handling (Chapter 3) on the external supervisory computer [17] since our PUMA was not operating in a noisy environment and also in order to

obtain a user-friendly control an interactive control software (Chapter 4-5) has been integrated to the supervisory communication and control software. Thus we have managed to simulate the robot and transmit required control commands from an external computer without any change on the original hardware configuration of the PUMA system.

1.2 Constraints Taken in Consideration

The main constraint to be considered during the studies was to preserve the original configuration both on the hardware and software of the PUMA 760 robot arm and its controller. Changing the configuration of the robot, controller and robot parts might result in malfunctioning of the whole system, which would be hard to recover since the robot was an old device manufactured in the mid 80's by Unimate Inc. Furthermore, the company does not exist today. Just a few companies give service for PUMA robots currently so obtaining spare parts is both hard and expensive. Obtaining technical information and consultancy is also difficult and the limited technical information we have may not be adequate for troubleshooting and recovery.

Another constraint that we have taken into account was not to change, modify or replace original cabling and connections other than those broken. Since cabling and shielding of electrical connections of the robot in its original state were done extremely well and neat. All the materials used are good in quality. Interruption on those connections may not cause malfunctioning of the system but may result in power leakages, EMI problems thus incorrect execution of the robot arm.

1.3 Hardware Problems

The PUMA robot as a whole system was not operational in the beginning of our studies. Thus we had to work on the hardware of the system. The troubleshooting charts (Appendix A) were then traced for fault diagnoses and isolation. At the end of our effort we have managed to get the robot back into operation. Details related to this work are given in Appendix B.

1.4 Outline of the Thesis

Introduction for the thesis is given within this chapter (Chapter 1), in which we have stated our objectives and previous studies.

In order to understand the work done throughout this thesis the general structure of PUMA 700 series robots and VAL II, which is the software package for the control of PUMA robot arm is described within Chapter 2.

In Chapter 3, available communication methods with PUMA, the communication method used and implementation of the supervisory communication software is described.

Interactive control software is implemented and integrated to supervisory control software of PUMA. The work carried out for this user friendly and useful application and the integration studies are described within Chapter 4. This software is actually operating similar to the teach-pendant of the PUMA system (more detailed explanation for the teach-pendant can be found in Chapter 2).

Another Interactive robot control application, which operates the robot over the supervisory communication port, has been integrated to the system. In this application, we have extracted image frames of a video from which we have determined speed, displacement and direction of the motion information that is inside the image frames. The data and information obtained is used to modify the pose of the PUMA manipulator. In this application a motion estimation algorithm, block-matching method has been integrated to the supervisory control interface, which is used for the path and joint movement modification of the manipulator. This study described within Chapter 5, is implemented in order to demonstrate the features and advantages of the supervisory control system.

Finally in Chapter 6, the test results are given. Conclusion and the proposed future works are also stated with this last chapter. At the end of Chapter 6, appendices are also presented for further reading and understanding.

CHAPTER 2

2 PUMA ROBOT

2.1 General

In this chapter, the robot system architecture both with hardware and the software is described and the kinematical properties are stated briefly that are used within the scope of this thesis.

2.2 The Types Of Puma Robots

PUMAs are probably one of the most common industrial type robots taking place in university laboratories. Originally designed in the mid-70's, the PUMA (Programmable Universal Machine for Assembly) was produced for many years by Unimation. Most known PUMA types can be classified into 3 groups with the specifications displayed in Tables 2.1-2.3.

Table 2.1 Specification of PUMA 200 series robot

DOF	6
Drives	DC Motors
Control	Numerical
Positional Control	Incremental Encoders
Coordinates	Cartesian
Configuration	Revolute
Minimum Reach	0.125 mm
Maximum Reach	0.406 mm
Limit Joint 1	308 deg
Limit Joint 2	314 deg
Limit Joint 3	292 deg
Limit Joint 4	578 deg
Limit Joint 5	244 deg
Limit Joint 6	534 deg
Repeatability	+/- 0.05 mm
Maximum Speed	1.2 m/s
Auxiliary Processors	6 Slave Microprocessors

Programming	Teach Pendant and VAL II
Serial Interface	RS232 or RS423
Memory Buffer	46KB
Battery Buffer	30 Days
Arm Weight	13.2kg
Controller Cabinet Weight	80kg

Table 2.2 Specification of PUMA 500 series robot

DOF	6
Drives	DC Motors
Control	Numerical
Positional Control	Incremental Encoders
Coordinates	Cartesian
Configuration	Revolute
Minimum Reach	0.864 mm (Between Joint 1 and 5)
Maximum Reach	360 deg Working Volume
Limit Joint 1	320 deg
Limit Joint 2	250 deg
Limit Joint 3	270 deg
Limit Joint 4	300 deg
Limit Joint 5	200 deg
Limit Joint 6	532 deg
Repeatability	+/- 0.1 mm
Maximum Speed	1.0 m/s
Auxiliary Processors	6 Slave Microprocessors
Programming	Teach Pendant and VAL II language
Serial Interface	RS232 or RS423
Memory Buffer	46KB
Battery Buffer	30 Days
Arm Weight	63kg
Controller Cabinet Weight	200kg

Table 2.3 Specification of PUMA 700 series robot

DOF	6
Drives	DC Motors
Control	Numerical
Positional Control	Incremental Encoders
Coordinates	Cartesian
Configuration	Revolute
Minimum Reach	0.125 mm(Between Joint 1 and 5)
Maximum Reach	360 deg Working Volume
Limit Joint 1	320 deg
Limit Joint 2	220 deg
Limit Joint 3	270 deg
Limit Joint 4	532 deg
Limit Joint 5	200 deg
Limit Joint 6	600 deg

Repeatability	-762 model +/- 0.2mm
	-761 model +/- 0.2mm
Maximum Speed	1.8 m/s
Auxiliary Processors	6 Slave Microprocessors
Programming	Teach Pendant VAL II language
Serial Interface	RS232 or RS423
Memory Buffer	46KB
Battery Buffer	30 Days
Arm Weight	-762 model 590kg
	-761 model 600kg
Controller Cabinet Weight	200kg

Note that, PUMA 700 and PUMA 500 series robots have similar specifications. The main difference between the robots is actually at their loading capacities. Thus the study to modify the control system on one of the series' may help to the study towards the modification of other series of robots. Knowledge gained in this thesis on PUMA 760 robot can be used for PUMA 500 series robots, as well. The extra attention and effort needed to be spent only on the software work due to different link and joint parameters of the PUMA 500 series robot. This assertion will be backed up with the explanations made in Chapters 3 and 4.

2.3 PUMA 760

The PUMA 760 robot system is composed of two separated parts: The robot arm and its control computer. In the following section the control computer and the major components are described within our concern of this study.

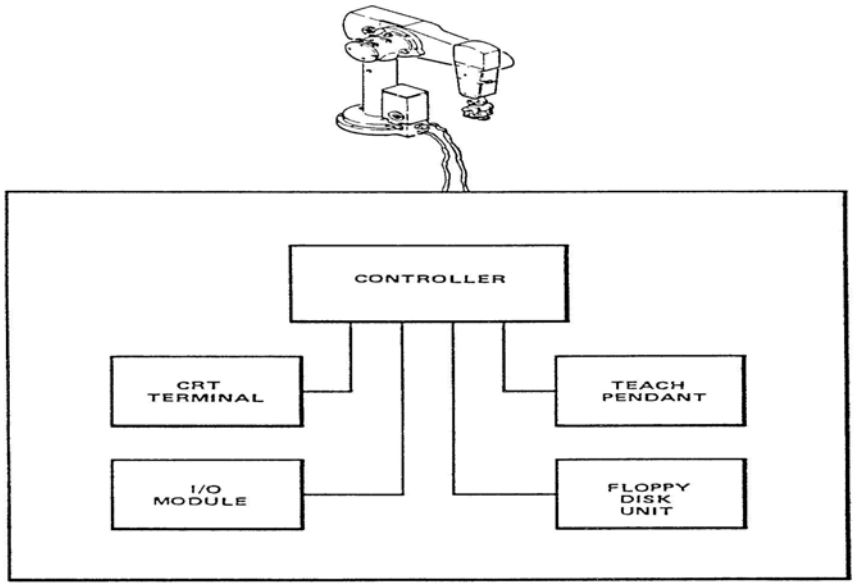


Fig. 2.1 Robot Arm & The Controller

2.3.1 The Control Computer

The controller is the master component of the electrical system. All signals to and from the robot pass through the controller and are used by it to perform real-time calculations to control arm movement and position. Operating controls and indicators are located on the front and top panel of the controller. Connections for the robot arm, terminal, floppy disk drive and accessories are located on the controller rear panel. Software is stored in the computer memory located in the controller. The software interprets the operating instructions for the robot arm, and the controller transmits these instructions to the arm. From incremental encoders and potentiometers in the robot arm, the controller/computer receives data about arm position. This provides a closed loop control of arm motions. A floppy disk drive is available to record the programs on diskettes.

The Unimation Mark II controller is a typical industrial robot controller. It consists of ten components:

- DEC LSI-11 computer with ADAC parallel interface board, DLV11-J serial interface board, CMOS board, and EPROM board.
- Servo interface board
- Six digital servo boards.
- Two power amplifier assemblies
- Power amplifier control board
- Clock/terminator board
- Input/output interface board
- Two power supplies
- High power function board
- Arm cable board

2.3.1.1 DEC LSI-11/2 Computer

The DEC LSI-11/2 computer system, which is a 16-bit processor based computer with up to 32K words of memory, has I/O interface board, DLVII-J serial interface board, CMOS random access memory board, EPROM (erasable, programmable read-only memory) board and a parallel interface board. It does not have hardware floating point support. This primitive twenty-year-old processor is used to compute the set points, which make up the robot joint level trajectory. It communicates via the four - port asynchronous DLV11-J serial interface card to a floppy disk drive, a teach pendant, a serial terminal and optional accessory. Commands are usually entered at the terminal or the teach pendant. The set point can be updated by the LSI-11 based software at one of several user selectable periods where the default update period is 28ms. Every 28 ms new set points for each joint are transferred from the LSI-11 to the servo interface board, which in turn distributes the appropriate set point to each digital servo board.

2.3.1.2 Digital Servo Boards

There are six digital servo boards, one per joint of the manipulator. Each servo board consists of a 6503 8-bit 1MHz microprocessor with 2,048 bytes of EPROM memory and 128 bytes of RAM, a counter/timer, and some parallel input/output capability. Each processor implements the position loop for its joint at approximately 1KHz. Computations are performed in fixed-point triple precision (24 bits).

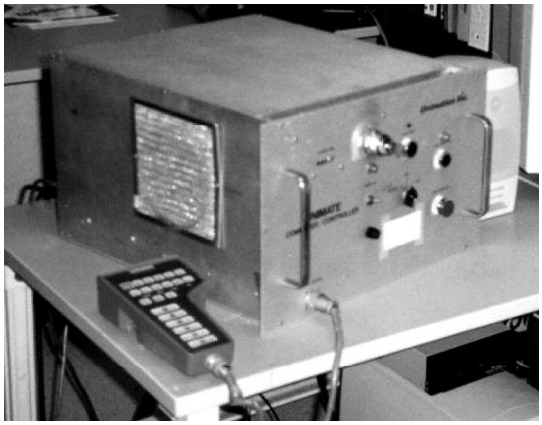


Fig. 2.2 The Controller Computer DEC LSI-11 and The Teach Pendant

2.3.1.3 Peripherals

Connected to the Mark II controller are a floppy disk drive, teach pendant, and terminal. Programs entered from the terminal can be stored on the floppy drive. The teach pendant is used to move the manipulator directly either in cartesian or joint space, and can also be used to program motions by moving the manipulator as opposed to entering desired end-effector configurations from the terminal.

The overall structure of the system is shown in Fig.2.3. This system is connected to the robot arm, which will be described in the next section.

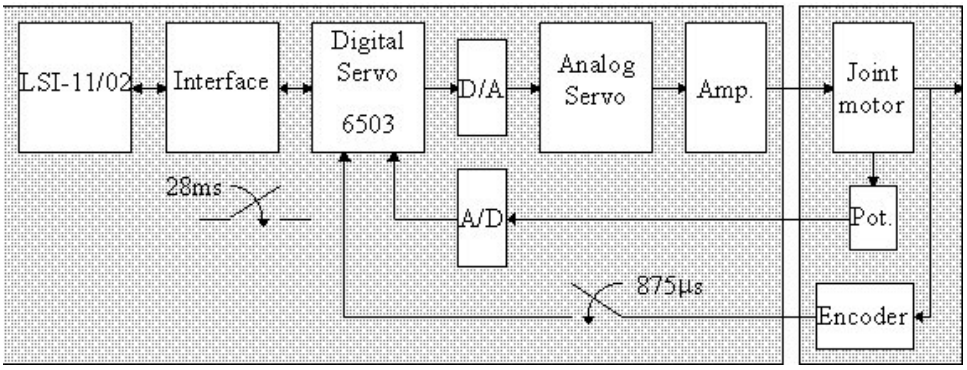


Fig. 2.3 The Overall System Structure

2.3.2 PUMA 760 Robot Arm

Next to the controller there is the robot arm connected to it, which is shown in Fig.2.4. The robot arm is a mechanical chain of links and joints incorporating 6 degrees of freedom (DOF). A DC servomotor controls each joint. It is similar to a human torso, shoulder, arm, and wrist. Fig.2.5 shows above defined components in the chain mechanical structure of the arm. The components of the robot arm are the trunk, shoulder, upper arm, forearm, wrist, and mounting flange. The robot arm members contain the various servomotors and gear trains.

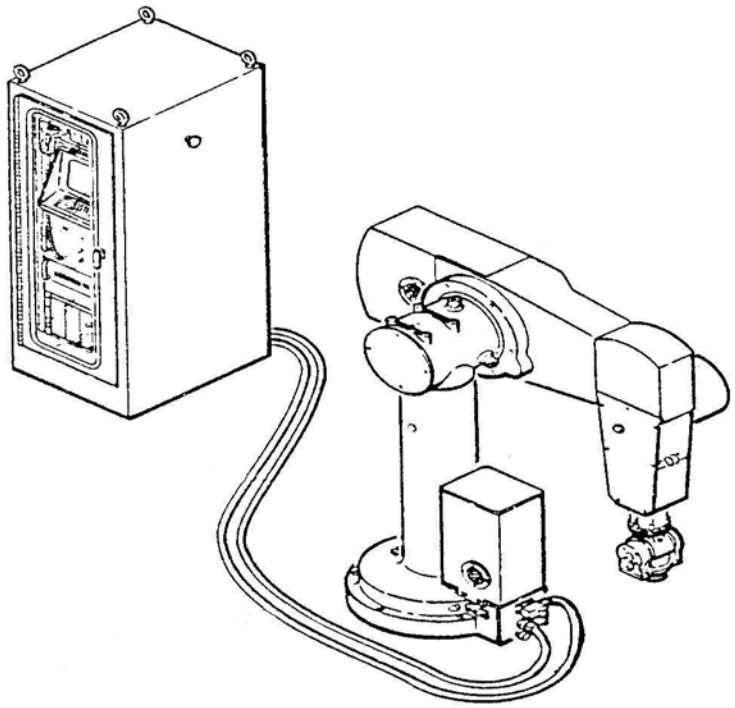


Fig. 2.4 PUMA Robot and Control Cabinet with the Assembly

Table 2.4 defines the joint axes for the PUMA robot arm.

Table 2.4 Robot Arm Axes

Joint	**Description**
Waist – Joint 1	Joint 1 axis is perpendicular to the mounting plane and coincident with the centerline of the trunk
Shoulder – Joint 2	Joint 2 axis is perpendicular to and intersects Joint 1 axis. It is coincident with the centerline of the shoulder.
Elbow – Joint 3	Joint 3 axis is parallel to the Joint 2nd axis.
Wrist Roll – Joint 4	Joint 4 axis is perpendicular to and intersects Joint 3 axes. It is coincident with the centerline of the forearm.
Wrist Bend – Join 5	Joint 5 axis is perpendicular to and intersects the Joint 4 axis.
Wrist Swivel – Joint 6	Joint 6 axis is perpendicular to and intersects Joint 5 axis. Joint 6 is coincident with the centerline of the mounting flange.

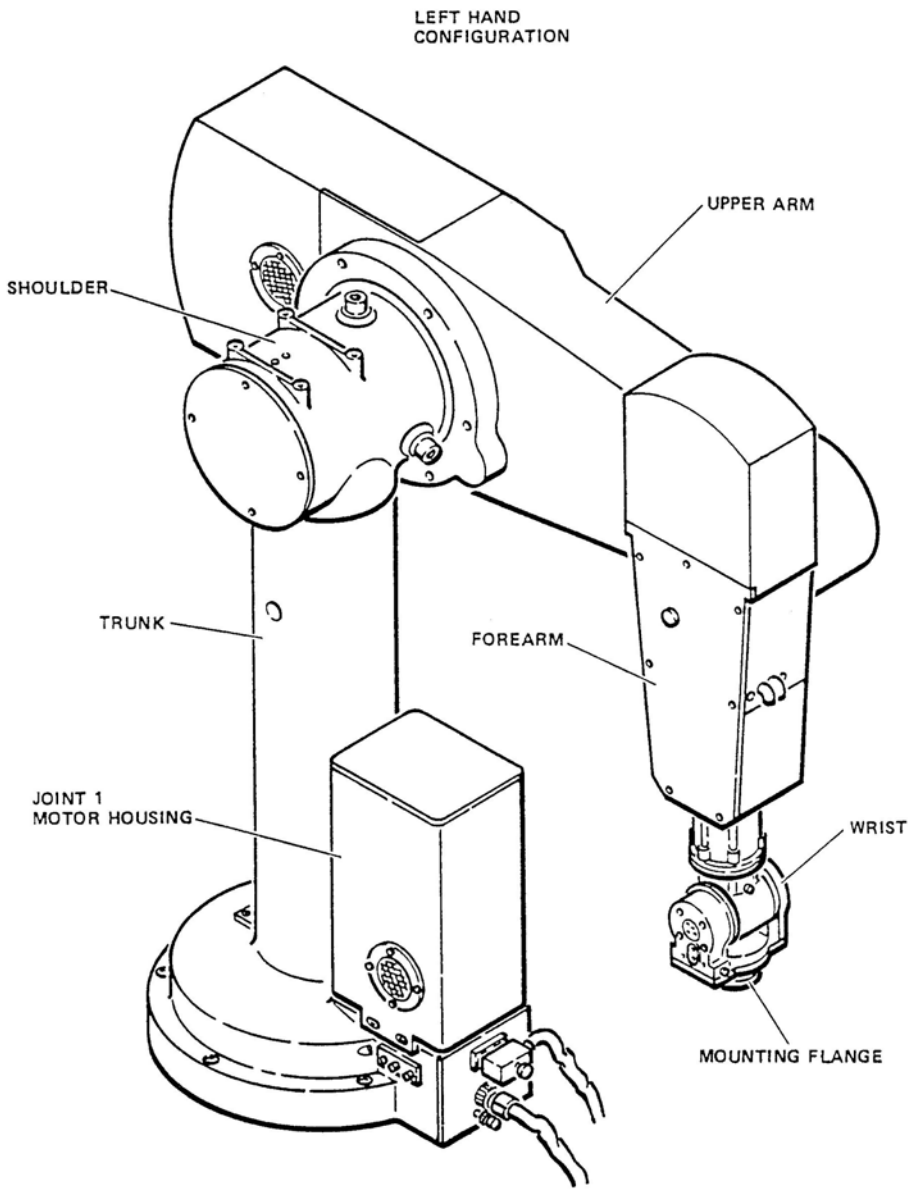


Fig. 2.5 The 6 DOF Robot Arm

From the mechanical structure shown in Fig.2.5 and the axes definitions (Table 2.4) one can drive the followings for joints:

Joint 1 is a revolute axis that rotates about the world Z-axis.

- A positive change of joint angle corresponds to a positive rotation about the world Z-axis.

Joint 2 is a revolute joint that rotates about a horizontal axis.

- When joint 1 is positioned at zero degrees, the axis of rotation of joint 2 will be parallel with the world Y-axis. In this position, a positive change in joint angle corresponds to a positive rotation about the world Y-axis.
- When joint 2 is positioned at –90 degrees, the inner link of the robot will point vertically in the direction of the positive world Z-axis.

Joint 3 is a revolute joint that rotates about a horizontal axis, which is parallel with the axis of rotation of joint 2.

- As with joint 2, when joint 1 is positioned at zero degrees, the axis of rotation of joint 3 will be parallel with the world Y-axis. In this position, a positive change in joint angle corresponds to a positive rotation about the world Y-axis
- When joint 2 is positioned at –90 degrees and joint 3 is positioned at +90 degrees (the "straight-up" position), the inner link of the robot will be vertical and the outer link will be approximately [6] vertical.
- The reason that the outer link may be slightly off vertical in some PUMA robots [6] is because, for some models, the axis of rotation of joint 4 does not intersect the axis of rotation of joint 3. When joint 2 is positioned at –90 degrees and joint-3 is positioned at +90 degrees, the axis of rotation of joint 4 must be vertical. If joint 4 does not intersect joint 3, the outer link will appear to be slightly tilted.

Joint 4 is a revolute (roll) wrist axis. When joints 2 and 3 are in their straight-up positions, the axis of rotation of joint 4 will be parallel to the world Z-axis.

- A positive rotation of this joint turns the robot's end effector in a positive direction relative to the world Z-axis.
- This joint is omitted in the 5-axis configuration.

Joint 5 is a revolute (pitch) joint. The axis of rotation of joint 5 is perpendicular to the axis of joint 4.

- When joints 2 and 3 are in their straight-up positions and joint 4 is positioned at zero degrees, the axis of rotation of joint 5 will be parallel to the world Y-axis. In this position, a positive change in joint angle corresponds to a positive rotation about the world Y-axis

Joint 6 is a revolute (roll) axis whose axis of rotation defines the nominal Z-axis of the robot's tool frame of reference.

- The axis of rotation of joint 6 is perpendicular to the axis of rotation of joint 5. Furthermore, the axes of rotation of joints 4, 5, and 6 intersect at a point in the center of the wrist.
- When joints 2 and 3 are in their straight-up positions and joints 4 and 5 are positioned at zero degrees, the axis of rotation of joint 6 will be parallel to the world Z-axis. In this position, a positive change in joint angle corresponds to a positive rotation about the world Z-axis.

Each joint is driven by a permanent magnet DC servomotor through an associated gear train, and each motor contains an incremental encoder and a potentiometer driven through a gear reduction. The position at each joint is measured relative to an initially known absolute position. The potentiometers, incorporated in the motors are used to determine this initial absolute position. The incremental encoder mounted on the shaft of each motor provides information about the changes in position involved with the respective joint. The joint velocity is, however, computationally derived from the positional changes. Joint encoder readings are sampled at every 28 msec. and compared with the calculated positions.

The major axes (axes belonging to joints 1, 2 and 3) are equipped with electromagnetic brakes to lock the arm in a fixed position in space. These brakes are activated when power is removed from the motors. This safety feature removes the risk of injury or damage that could result from the arm collapsing if power is accidentally removed.

Power for the motors is supplied through the cable connecting the robot arm and the controller. This cable bundle, with different cables, also carries feedback signals from the incremental encoders and potentiometers.

Table 2.5 tabulates joint limits and angular resolutions used in simulations and integrated into the external control system developed in this thesis study. Fig.2.6 shows these joint limits [4].

Table 2.5 Limits and angular resolutions

Joint No	Joint Limits	Joint Angular Resolution
Joint 1	320 degrees	0.0050 degrees (min)
Joint 2	220 degrees	0.0035 degrees (min)
Joint 3	270 degrees	0.0092 degrees (min)
Joint 4	532 degrees	0.0082 degrees (min)
Joint 5	200 degrees	0.0080 degrees (min)
Joint 6	532 degrees	0.0111 degrees (min)

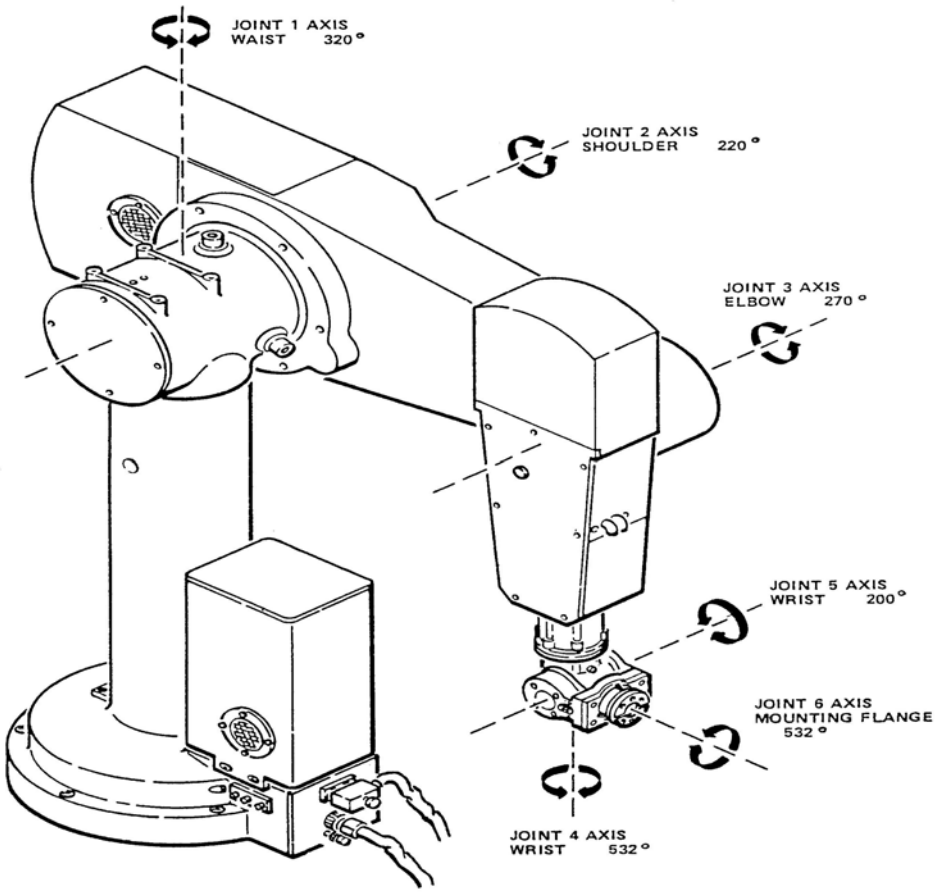


Fig. 2.6 The Movement Limits of PUMA 760 Joints

As a last element in this mechanical chain structure a pneumatic-controlled hand (gripper) takes place, as seen in Fig.2.7. This hand has two stable states: open or close. A stand-alone air compressor supplies the compressed air necessary to open and close the two fingers of the gripper.

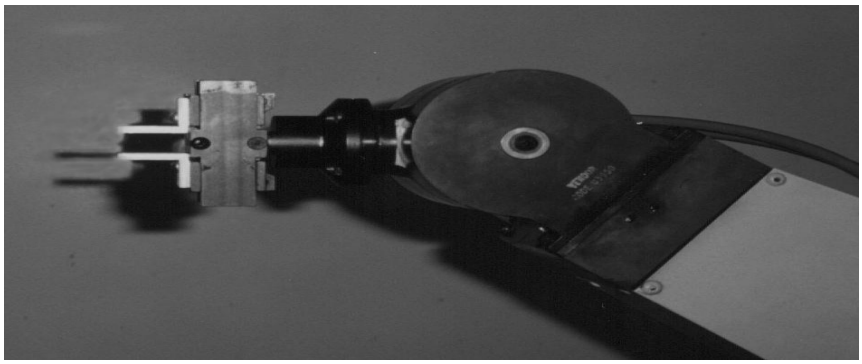


Fig. 2.7 The Gripper

2.4 The Operating System - VAL

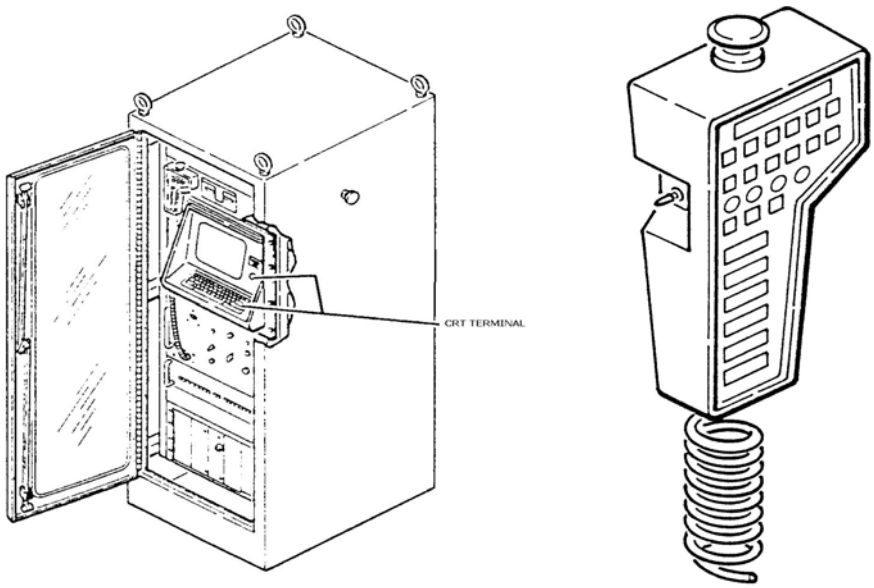
2.4.1 General

Initially, robots were programmed using textual languages such as Pascal and C. The code required to drive robots was very low-level, and very hard to create and maintain, requiring skills in both advanced programming and control theory. Textual robotic languages were developed specifically to address the problems associated with programming robots. These languages introduced "built-in" commands to operate the robot, eliminating (for instance) the need to develop code for motion primitives.

2.4.2 VAL

The system software that controls the PUMA robot arm is called VAL, which is supplied by Unimation to be used with its controller called as MARK. The software is a sophisticated programming language and a complete robot control system, but has disadvantages mostly due to age of the product. VAL is stored in the controller computer memory. The controller also houses operating controls for the robot system.

The VAL programming language consists of a full set of English language instructions for teaching and editing. Work programs are entered into the computer/controller using either of two different procedures, or a combination of both. The programs can be entered with the teach pendant using the teach-by-showing method, or using the CRT and keyboard inputs. Full programming versatility can only be achieved through keyboard inputs, thus the CRT terminal shown in Fig.2.8a).



a. CRT Terminal **b. Teach-pendant**

Fig. 2.8 CRT Terminal and The Teach-pendant

In either programming method that is from CRT terminal or the teach-pendant, all taught and learned points are stored and referenced to a 3-axis coordinate system fixed relative to the robot base. The robot arm base is fixed as shown in Fig. 2.9.

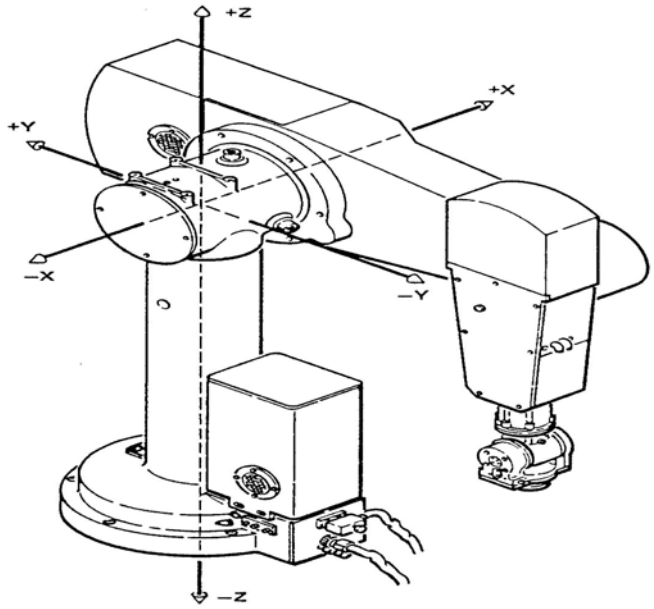


Fig. 2.9 The Base of Robot Arm and 3-axis coordinate frame definition for the base

In order to teach the PUMA system, as stated above, either of two methods, or a combination of both, can be used. In the first method the teach pendant is used to individually manipulate each joint to a desired orientation. This operation is available when the robot arm power is on, since the teach-pendant actually moves the robot as desired by the user. The combination of the six joint orientations determines the arm location. The arm location is entered into the computer memory with the teach pendant. The positions of joints are saved into the memory separately, which ends up with the position / location of the whole robot arm. A series of arm locations will produce a program. The second method is to write a program using VAL instructions. The programs and arm position data are entered into the computer memory with the CRT terminal keyboard.

Two coordinates systems can be selected from both of the programming methods mentioned here. These are the "World Mode" and the "Tool Mode".

- **World Mode:** The reference coordinates for the World Mode are fixed in the robot arm base shown in Fig. 2.9. This mode can be chosen when the gripper is to move parallel to any of the World axes, or to rotate about those axes.

- **Tool Mode:** The reference coordinates for the Tool Mode are fixed in the gripper, with their origin at the center of its mounting flange. This mode can be chosen when the gripper is to move parallel to or rotate about the tool coordinate axes. See Fig. 2.10 for the tool coordinate axes.

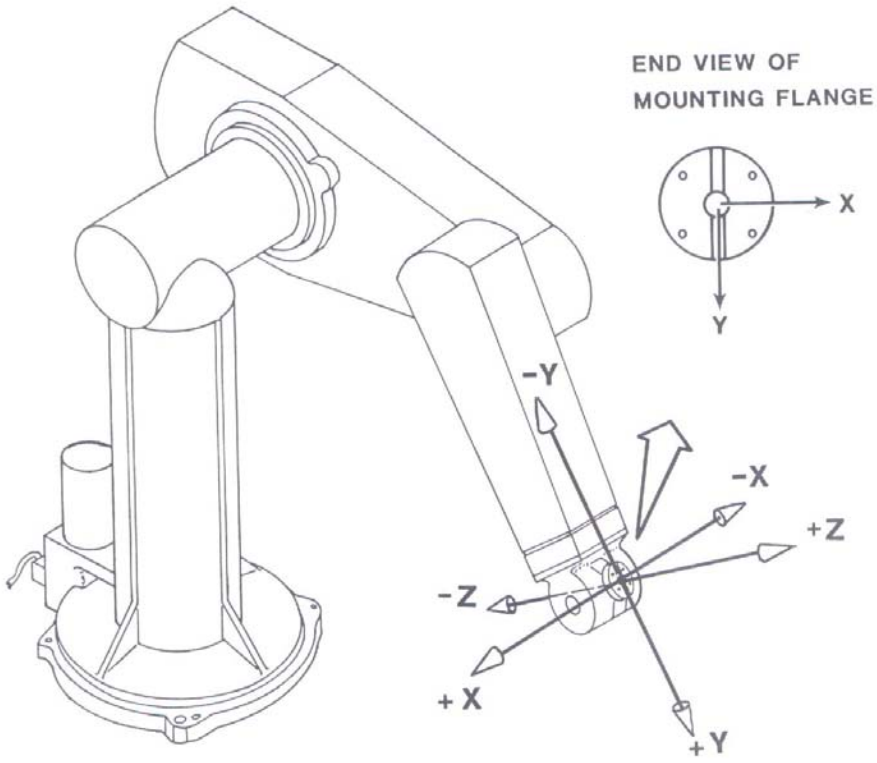


Fig. 2.10 The Tool Coordinate Axes

The program developed may be stored external to the controller on a floppy disk and after completing the program development user may execute it.

VAL actually has two versions: VAL and VAL II. The version of VAL we have used during this thesis work was VAL II. Some of the significant changes and modifications made for the new version are:[1]

- A formal communication capability has been added for supervisory control (see Chapter 3) of the robot system from another computer.
- Some mathematical capabilities have been added, including real variables, variable arrays, and scientific functions.
- A second user program can be executed in parallel with the robot control program. This second program can perform any VAL II operation that does not access the robot.

Throughout the studies of this thesis, VAL II has been used as the control software of the robot arm.

2.4.3 Characteristics of VAL II

VAL II has three modes of operation:

- Edit mode: the user can write a new program or edit an existing one from the terminal.
- Monitor mode: the user can define locations in space using the teach-pendant or the terminal, transfer programs from storage back into control memory, etc.
- Run mode: execution of the robot program.

Within those modes users are related to following characteristics of VAL:

- Data structures (see section 2.4.3.1)
- Motion commands (see section 2.4.3.2)
- End effector and sensor commands (see section 2.4.3.3)
- Computations and Operations (see section 2.4.3.4)
- Program Control (see section 2.4.3.5)

2.4.3.1 Data Structures of VAL II

Available data structures are described below with their examples:

Numeric values: Integer and real values are used within VAL II.
E.g.: Integer: 97 - 1983 0 32767 Real: 6.64 8.5E-3 -337 - 9.4177

Numeric variables: Numeric variables are names assigned to a memory location that can contain different numeric values during the calculation.
E.g.: The following are valid variable names: x count distance.to.part.3

Array variables: An array is a group of values that share a single name. Appending an index enclosed in brackets to the array name specifies an element of an array.
E.g.: x [3] Assignment instruction x = 3.

Mathematical operators: Available mathematical operators are <+, -, *, /, MOD>
E.g.: x = 0.5*(-length)

Locations: Robot locations are used to specify the destination of robot motions. There are two types of locations, which are precision points and transformation.
E.g.: The following are valid location variable names:
p feeder pallet.to.part.3
Precision points are used when the robot location is represented by the exact positions of the individual robot joints. PPOINT (20, 0, 30, -90, -33, 71).

Transformation is a robot-independent representation of the position and orientation of the robot tool. The position of the tool is defined with x, y, and z coordinates, and the tool orientation is defined by three angles measured from the coordinate axes. TRANS (0, 0, 0, -90, 90, 0)

Compound transformation: Compound transformation provides a means of specifying robot locations relative to the location represented by other transformations.
E.g.: plate: object: grasp

2.4.3.2 Motion Commands

Robot motion instructions are given through motion commands. Most commonly used motion commands are:

HERE <A1>: The pendant is used to drive the robot arm to the desired position and orientation, and then the above command captures the value of the location.

WHERE: Will cause the present world coordinates and joint co-ordinates to be displayed.

MOVE <A1>: Moves the end of the arm from its present location to the location A1.
MOVES <A1>: Move to location A1 in a straight line.

APPRO <A1, 50>: Approach location A1 with offset of 50 mm along the z-axis of the end-effector frame.

APPROS <A1, 50>: Approach A1 with straight-line motion with offset of 50 mm along the z-axis of the end-effector frame.

DEPART <50>: Depart along z-axis to a distance of 50 mm.

SHIFT <(LOC1 BY dx, dy, dy)>: Location LOC1 will have x, y, z coordinates changed by dx, dy, dz relative to the original values.

SPEED <60>: The speed of the end effector during the program execution will be 60% of the maximum declared speed.

2.4.3.3 End Effector and Sensor Commands

End effector commands are the ones that affect the gripper whereas sensor commands are used as signal triggers for the system. Some of the available commands are:

OPEN (CLOSEI)
CLOSE (OPENI)
} Open or close gripper. The action will be executed during the next motion

E.g.:
APPRO A1, 50
MOVES A1
CLOSEI
DEPART 50

2.4.3.4 Computations and Operations

VAL II enables users to use some elementary mathematical functions, relational operators and logical operators.

SET <x>: Initializes a location variable.
E.g.: x = plate: object: grasp

Elementary functions: trigonometric, logarithmic, exponential, square root, etc.

Relational operators: LE, LT, GE, GT, NE, EQ (<, >, ==, <>, <=, >=, etc.).

Logical operators: NOT, OR, AND

2.4.3.5 Program Control

The program control instructions control the sequence in which the user program instructions are executed. Thus, they can be used to control the logical flow within the user programs. Below there are main statements that can be used within the user programs.

IF statement:

```
IF (logical expression) THEN
… {Group of instruction}
ELSE
… {Group of instruction}
END
```

WHILE loop:

```
WHILE (logical expression)
… {Group of instruction}
END
```

FOR loop:

```
FOR I=0 TO max
…{Group of instruction}
END
```

CASE statement:

```
CASE {expression} OF
  VALUE <expression>
  …{Group of steps}
  VALUE <expression>
```

DO…UNTIL loop:

```
DO
…{Group of steps}
UNTIL <logical expression>
```

```
…{Group of steps}
ANY
…{Group of steps}
END
```

2.5 Practicing VAL II

VAL programming practices were conducted in order to understand the capabilities, advantages and disadvantages of VAL.

Throughout those practices, all the features and skills described at section 2.5 were tried. Programs were written and executed. It was found that VAL had the following capabilities and advantages:

- VAL is a complete robot control software package,
- Real-time calculations are performed during the actual running of the robot program,
- Built-in commands ease the control programming of the robot arm,
- Does not require pre-knowledge of robot control,
- Enables the use of teach-pendant,
- Enables external computer connection.

Most of the disadvantages that VAL possesses are actually due to its age. Below there are the main disadvantages:

- Calculations related with inverse kinematics require time,
- Programs written in VAL may be difficult to read, not only due to the cryptic nature of code, but because code can be poorly commented,
- Parallel running applications must be mapped serially,
- There is no convenient means for synchronizing two flows of code except by the multiple use of “begin - end” blocks,
- VAL grammar is not expandable that is new actions can only be used via external procedures or functions,

- VAL does not support any simulators,

- Visualization is not possible,

- Programming editor does not exist as it exists within all programming languages today,

- Does not support new sensory information to be considered for the control of the arm,

- Off-line programming is not possible

In order to overcome the disadvantages of VAL and improve the capabilities of the overall system, without modifications on the original hardware, it was decided to use an external computer. The studies conducted for the external computer control are within the scope of Chapter 3 of this thesis.

2.6 Model for PUMA 700 Series

Kinematics is the study of motion without regard to the forces that cause it. Within kinematics one studies the position, velocity and acceleration, and all higher order derivatives of the position variables. The kinematics of manipulators involves the study of the geometric and time based properties of the motion, and in particular how the various links move with respect to one another and with time.

PUMA robots are serial-link manipulators comprising a set of bodies, called links, in a chain, connected by joints. Each joint has one degree of freedom, which are rotational. For a manipulator with 6 joints numbered from 1 to 6, there are 7 links, numbered from 0 to 6. Link 0 is the base of the manipulator, and it is fixed, and link 6 carries the end-effector. Joint i connects links i-1, and i, that is joint 3 for example connects links 3 and 2.

A link may be considered as a rigid body defining the relationship between two neighboring joint axes. A link can be specified by two numbers, the link length and link twist, which define the relative location of the two axes in space. The link parameters for the first and last links are meaningless, but are arbitrarily chosen to be 0. Joints may be described by two parameters. The link offset is the distance from one link to the next along the axis of the joint. The joint angle is the rotation of one link with respect to the next about the joint axis.

To facilitate describing the location of each link we affix a coordinate frame to it — frame i is attached to link i. Denavit and Hartenberg (DH) [11,12] proposed a matrix method of systematically assigning coordinate systems to each link of an articulated chain. The axis of revolute joint i is aligned with z_{i-1}. The x_{i-1} axis is directed along

the normal from z_{i-1} to z_{i-1} and for intersecting axes is parallel to $z_{i-1} \times z_i$. The link and joint parameters may be summarized as:

Link length (a_i):	the offset distance between the z_{i-1} and z_i axes along the x_i axis.
Link twist (α_i):	the angle from the z_{i-1} axis to the z_i axis about the x_i axis.
Link offset (d_i):	the distance from the origin of frame i-1 to the x_i axis along the z_{i-1} axis.
Joint angle (θ_i):	the angle between the x_{i-1} and x_i axes about the z_{i-1} axis.

According to the information above link coordinate assignment can be made as shown in Fig. 2.11.

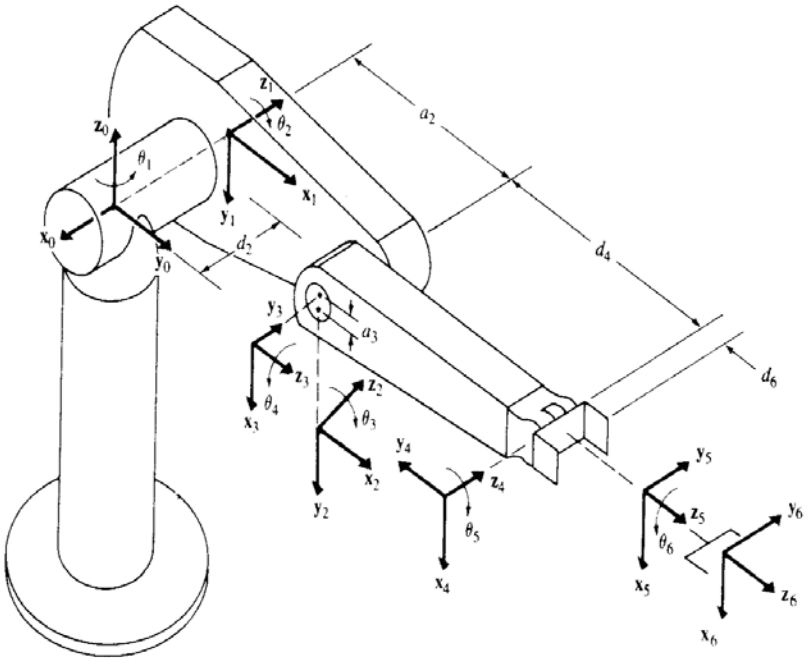


Fig. 2.11 Link Coordinate (Frame) Assignment

It shall be noted that one can have different frame assignment due to the assignments of x and y coordinates but the resulting transformations would not change.

Using the data given in Fig.2.12, which shows the dimensions of PUMA 700 series robot, DH (Denavit and Hartenberg) parameters can be determined as shown on Table 2.6.

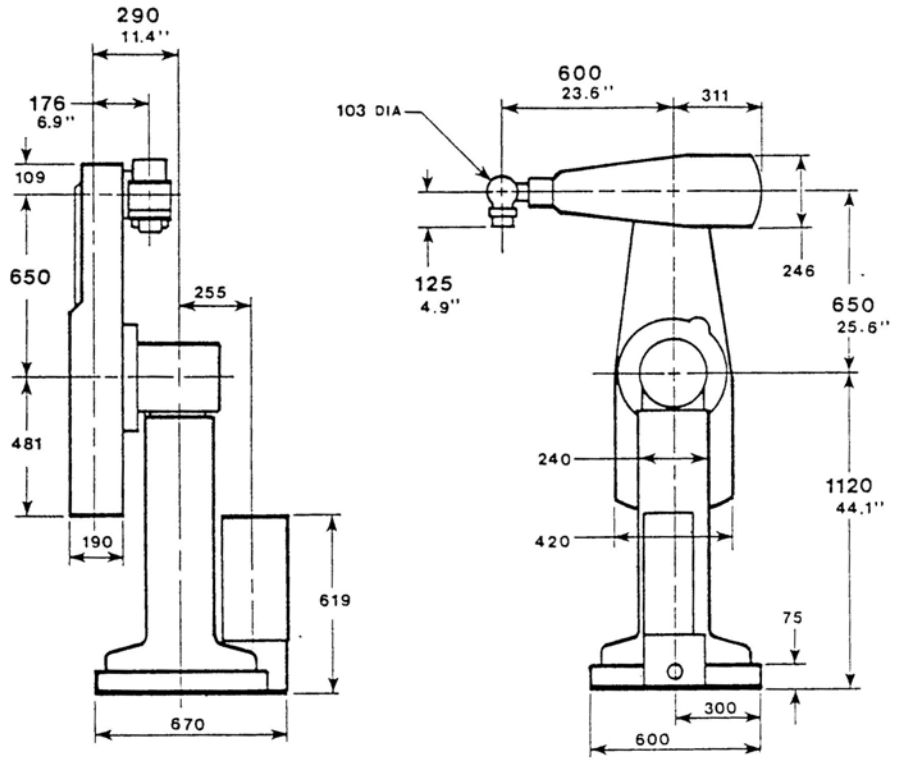


Fig. 2.12 Dimensions of PUMA 700 Series Robots

Table 2.6 DH Parameters For the PUMA 700 Series

Joint	θ_i	α_i	a_i	d_i
1	90	-90	0	0
2	0	0	650mm	290mm
3	90	90	-85mm	0
4	0	-90	0	600m
5	0	90	0	0
6	0	0	0	125mm

It is important that the offset distance and link lengths may change due to the modifications on the mechanical structure of the robot.

After the establishment of the DH coordinate system, a homogenous transformation matrix can be developed [11,12]. The DH representation results in a 4x4 homogenous transformation matrix.

$$ {}^{i-1}A_i = \begin{vmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.1) $$

representing each link's coordinate frame with respect to the previous links coordinate system; that is

$$ {}^{0}T_i = {}^{0}T_{i-1}\ {}^{i-1}A_i \quad (2.2) $$

where ${}^{0}T_i$ is the homogenous transformation describing the pose of coordinate frame i with respect to the world coordinate system 0.

Using (2.1) we can obtain the following transformation matrices:

$$ {}^{0}A_1 = \begin{vmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.3) $$

$$ {}^{1}A_2 = \begin{vmatrix} C_2 & -S_2 & 0 & a_2 C_2 \\ S_2 & C_2 & 0 & a_2 S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.4) $$

$$ {}^{2}A_3 = \begin{vmatrix} C_3 & 0 & S_3 & a_3 C_3 \\ S_3 & 0 & -C_3 & a_3 S_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.5) $$

$$ {}^{3}A_4 = \begin{vmatrix} C_4 & 0 & -S_4 & 0 \\ S_4 & 0 & C_4 & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (2.6) $$

$$ {}^{4}A_5 = \begin{vmatrix} C_5 & 0 & S_5 & 0 \end{vmatrix} $$

$$
{}^{5}A_{6}= \left|\begin{array}{cccc} S_5 & 0 & -C_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{array}\right| \quad (2.7)\ (2.8)
$$

C_i and S_i are defined as $C_i = \text{Cos}\ \theta_i$, $S_i = \text{Sin}\ \theta_i$,

Using the (2.2) - (2.8) we can obtain the T matrix, which specifies the position and orientation of the endpoint of the PUMA with respect to the base coordinate system.

$$T= {}^{0}A_{6} \quad (2.9)$$

The equation above corresponds to:

$$T= {}^{0}A_{1}\,{}^{1}A_{2}\,{}^{2}A_{3}\,{}^{3}A_{4}\,{}^{4}A_{5}\,{}^{5}A_{6} \quad (2.10)$$

Solution to the complete transformation matrix is not given since it is a long expression, but it can be easily calculated with the help of a computer.

The model, which is described within this section, shall be used for the simulation and visualization of the PUMA 700 series robot.

CHAPTER 3

3 SUPERVISORY COMMUNICATION

3.1 General

In this chapter the available communication modes for the PUMA robot arm are investigated. Within that context the supervisory communication mode is described in detail together with the protocol DDCMP (Digital Data Communications Messaging Protocol) and the software implementation of this mode with its protocols are explained.

3.2 Available Modes Of Communication

The controller has two special ports (on its rear panel) to communicate with external computers. These ports are referred as:

- The Alter Port
- The Supervisor Port

VAL II supervisory communication interface is normally used to send information or commands to the controller system. If, however, the command or information, sent in real-time, is to modify the path of the robot while it is moving then the interfacing must be done using the alter port, not through the supervisory port. The communication through alter port is basically for real-time path control.

The Alter port:

This port is also used to handle sensory inputs required for the control of the robot besides the real-time path control purposes. For example, in supplying sensory feedback information to the controller, in order to modify robot motions while they are actually occurring, the use of communication over alter port is necessary. In other words, real-time path control is a means for modifying the program-directed position and orientation of the robot tool while the robot is in action. When this type of path control is in effect, VAL II is in the "ALTER mode". Although an external sensor or camera cannot be directly connected to PUMA arm, it is possible to connect an external sensory device over this serial port (RS-423, RS-232 compatible), alter port. Serial binary data obtained from the sensor (camera, infrared detector, etc.) or a computer supplying position data and instructions can be sent to

VAL II over alter port. Thus, real-time path control data to the controller comes from two sources; either from an external computer, or from a user written VAL program in robot's own computer. In the former, data will be transferred to the controller by using the 'external' alter mode communication protocol. In the latter case the 'internal' alter mode communication protocol will be used for the purpose.

External Alter Mode:

In this mode, VAL II repeatedly establishes communication with the external computer at a rate 36 times/sec. to acquire the data about the modified robot tool trajectory determined and sent by the external computer. This operation continues indefinitely unless either VAL II terminates the communication for some reason, or the external computer sends information about the existence of an exceptional condition.

The alter mode communication is initiated by the controller so that VAL II in response sends an "alter starting" message to the external computer and waits for a start up acknowledgment. VAL II terminates the communication attempt with an error if the external computer does not respond to the call, otherwise, upon receipt of acknowledgement VAL II enters the alter mode communication at the start of the next robot motion. With the beginning of robot's next motion, VAL II sends an "alter running" message to the external computer and expects to receive a complete alter data message from the external computer within about 16 msecs, starting from the instant when it issued out the "alter running" message. VAL II continues to send "alter running" messages, every 28 milliseconds, as long as the alter mode is active. During this time, that is when the alter mode is active, the external computer connected to the PUMA controller can generate and signal an "exception condition", this exception is an interruption to the normal flow of program control, caused by the program itself. The exception condition causes a user-specified VAL II subroutine, which is written by the user, to be processed asynchronously, like a program reaction or function. This process is similar to exception handling programming within well-known languages like C/C++ and Java. Exception handler is a special code, which is called when an exception condition occurs during the execution of a program. If the programmer does not provide a handler for an exception, the program running within "alter mode" will crash down. Using the exception generation and exception handling methods, programmer can control the flow of the program, generate new functions and handle any communication errors between PUMA and the external computer.
VAL II remains in alter mode until the execution of a NOALTER instruction within the program which initiated the alter mode, or the robot program stops executing for any reason. Note that the latter includes unexpected termination of program execution due to an error, such as attempting a joint-interpolated motion, which is the motion where the joint taking the longest time to make the joint change governs the motion and the other joints are slowed in proportion so that all joints accomplish their joint changes simultaneously with the slowest joint. Thus the program for the alter mode crashes but the PUMA stays within the alter mode and in order to

overcome this problem the system requires to be re-started and re-enabling of the alter mode. Alter mode remains in effect and VAL II does not attempt to re-initiate the communication or shutdown the system totally even if continuous-path motions are broken purposely due to program instructions. That is, if a program instruction causes an error then program instructions for real-time control can not be executed, but this error does not effect the status of the communication in the alter mode, it may still be maintained. In case such a situation arises, the controller shall receive a 'NOALTER' command instead of attempting to continue program execution under the ALTER mode. When ending alter mode, VAL II sends the external computer an "alter pausing" message, followed somewhat later by an "alter ending" message. The external computer needs not to respond to either message. To re-enter the alter mode, another ALTER instruction must be executed in the VAL II program.

Internal Alter Mode:

Internal alter is useful when a process control program (written in VAL II) needs to control the robot motion according to a pre-processed sensory data (from sources such as infrared, sound sources, or a camera) set. Supplying pre-processed sensory data to the arm is important in practice. Since within the environment there might be an obstacle, another working robot that needs to be determined and the most effective way of determining such things would be the use of new integrated sensors such as cameras. Other than that, some robots, which examine and fix manufacturing mistakes and non-conformances with soldering, painting, gas leakage or gluing on products, require the use of external intelligent sensors. Internal alter mode is used under those circumstances, which makes it a useful communication mode.

Internal alter does not transmit messages to an external computer. Instead, VAL II expects a program instruction to be executed every 28 milliseconds to transfer data to the robot motion controller.

Internal alter mode shall be used in situations in which the robot motion is modified by inputs from a simple sensor. A process control program reads the sensor input from such a simple sensor to calculate corrections to the robot motion. An instruction incorporated in the control program will send the correction data to the motion program.

The Supervisor port:

This port can be used to interface the robot controller more precisely its program language VAL II, to a supervisory computer via a RS-232 compatible serial line. This link can operate at a maximum speed of 9600 baud (about 1000 characters per second) by using communication protocol software.

The supervisory port can perform the following functions:

- Issues all commands available at the robot terminal.

- Inputs and outputs data directly to and from user programs.

- Downloads and uploads user programs and data.

- Monitors operational status of the robot system.

So, by using these links, a supervisory computer can access several robot controllers and manage them towards the accomplishment of desired tasks simultaneously. So creating a network of robot controllers is possible.

3.3 Choosing The Communication Mode

As explained so far there are two alternative communication modes (alter and supervisory) for controlling the robot externally. Alter mode enables real-time path modification, and handles data transfer to control the robot arm. It also gives a means for sensory data to be obtained and processed coming from the incremental encoders and potentiometers in the robot arm about the actual arm position. Using and processing this data transmitted to the external computer, alter mode instructions are generated. The previously carried out studies [13] and the studies that we have carried out in the lab, show that the information obtained from the modified servo controllers, which hold the required information, are hard to obtain, difficult to read and they are very complex. The most applicable idea is to integrate force / torque transducers to the robot arm [13]. Those transducers (models with ISA cards exist) are to supply data about the 6 joints to the external computer, which analyzes data and sends new path commands to the robot controller within 28 milliseconds.

The advantages of the alter mode control method are clear. However, for this thesis work supervisory mode has been used due to the following reasons:

- Reading data, from the actual existing position and velocity sensors, are not satisfactory. It requires modification on the harness configuration of the system. Trials of gathering accurate information were not successful, probably due to the connections.

- Replacing the sensory system with the one complying the conditions (force/torque transducers) is quite expensive, and procurement demands considerably long time [13].

- The VAL programming language does not support efficient real-time updating of the manipulator trajectory based on sensory data (such as from force / torque or vision sensors). Unimation, manufacturer of the arm, has confirmed that there is an undocumented bug in the tool mode under real-time path control. During real-time control, such as alter mode, the controller does not update the rotation matrix as the robot moves [14].

- Supervisory Communication enables all VAL commands to be executed from an external computer, thus VAL capabilities are carried out to an external computer.

- Supervisory communication mode enables network control of the robot arm if desired.

3.4 Supervisory Mode

The physical communication link for the supervisory communication is the RS-232 serial line (Fig.3.1). The baud rate for the serial port used for the supervisory communication is set at 9600 baud. The communication data are transmitted as 8-bit bytes, with one stop bit for each byte. No parity information is included in the data bytes.

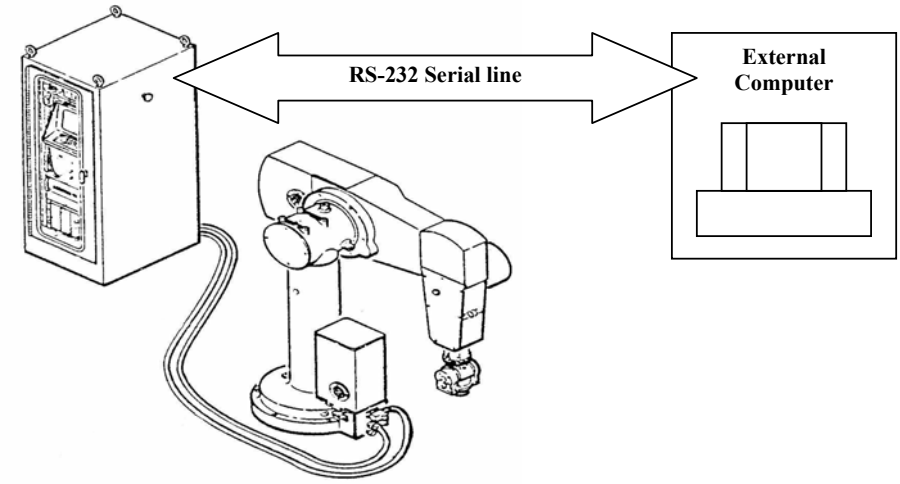


Fig.3.1 Physical Communication Link

Several different types of supervisory communications can occur simultaneously. If there is a physical link for each type, an addressing scheme could be used to direct the messages for each type of communication to the appropriate physical link. Also, the recipient of the messages will know which physical links they come from, and thus will know the nature of each message.

NETWORK and SUPERVISOR switches shall be enabled in order to activate the supervisory port for communication.

In reality every message includes a code for the physical link used for that type of communication. The recipient can then use the code within the message to determine the nature of the message without concerning itself as well as the physical path used for the message. Thus, with such a message format (that is, code plus message), there is no longer a need for individual physical links.

Since VAL II uses only one physical link for its supervisory communications, it uses such a coding scheme within every message to identify the type of communication. Considering the system as being subdivided into separate subunits, referred to as “logical units” do this scheme. Each type of communication is then associated with a particular logical unit. Each logical unit is assigned a ‘logical unit number’ (LUN), which is used during communication as the code for identifying message types. The logical units and LUN's listed in Table 3.1 are defined.

Table 3.1 LUN and Logical Unit

LUN	Logical Unit
0	Network manager
1	Short status information
2	Monitor input and output
3	Monitor asynchronous output
4	Program terminal Input and output
5	Disk input and output

The messages formed by using the LUN’s and commands are both packed into DDCMP packets. DDCMP is the bottom layer communication protocol that ensures each message transmitted or received is 100 % error free. DDCMP, which is a trademark of Digital Equipment Corporation, is a rigorous protocol that automatically handles the detection of transmission errors and re-transmission of messages when an error occurs. DDCMP was created by DEC (Digital Equipment Corporation) for the use with all of their operating systems. DDCMP does not use two or three ways handshaking method but it actually checks every individual packet and requests re-transmission and re-sequencing of packets, thus it is a stop-and-wait type of protocol. This method increases reliability, but on the other hand it takes time, which caused DDCMP packaging hardware to be used in 1980’s, but with today’s powerful computers we are able to form DDCMP packets faster.

In the supervisory mode the message packets are formed as shown in Fig.3.2.

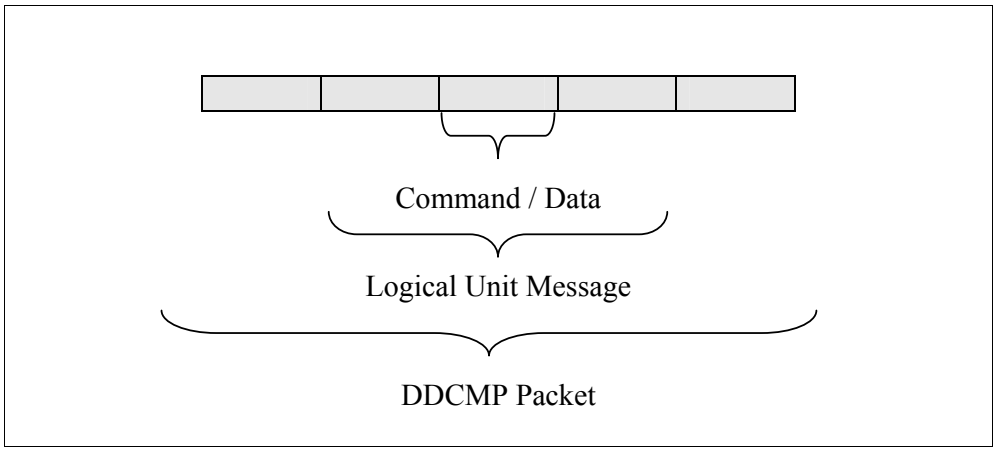


Fig.3.2 Complete Packet

3.4.1 General Format for the Logical Units

The format of the logical unit records sent and received within the supervisory communication system is shown in Fig. 3.3. This format shall be used within the supervisory communication for both messages transmitted by VAL and for messages transmitted by the supervisory external computer.

ID	Function Code	Function Qualifier	Message Data
1 byte	1 byte	2 bytes	0-256 bytes

Fig.3.3 Format of message for Logical Units

The ID byte identifies the communication protocol in use and the logical unit associated with the message data. Available ID byte assignments are listed in Table 3.2.

Table 3.2 ID Byte Bit Assignment

Bits	Description
7-6	Protocol Version
5-0	LUN (Logical Unit Number)

For the logical unit number 3 (Monitor asynchronous output), the ID byte will be formed as shown below within Fig.3.4.

0	0	0	0	0	0	1	1

Fig.3.4 ID Byte for LUN 3

The function code is used in messages transmitted by VAL II in order to define how the supervisory computer shall respond to the data message. The function code can be one of the listed values in Table 3.3.

Table 3.3 Function Code

Function Code	**Description**
0	Transmit a Command
1	Abort Outstanding Communication
2	Read Data
3	Write Data
4	Read After Writing Data
5	Read characteristics

Supervisory computer, in reply to a transmission from a logical unit, shall set the acknowledge bit (7^{th} bit) of the function code byte received from VAL II.

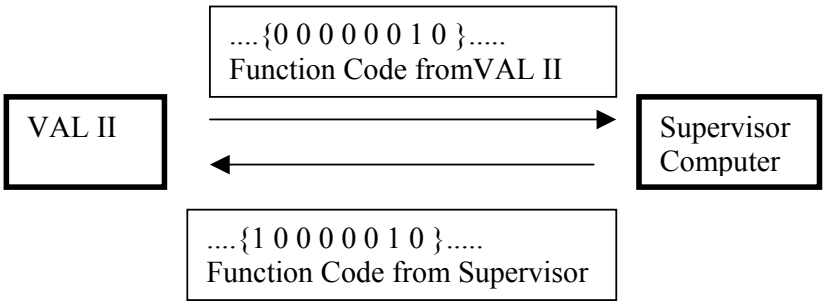


Fig. 3.5 Function Code Reply Rule

Note that because of the acknowledgement bit, in the reply message the function code shall be {'the received value' + 128}.

The Function qualifier bytes in a message provide further control on communications and messages thus the robot and by providing new command sets previously defined by Unimation. The function qualifier bytes differ according to the logical unit used.

3.4.2 Network Manager - LUN 0

This logical unit conveys certain commands to the VAL II system from the supervisory system unlike all the other logical units, because it is the only logical unit that has communications initiated by the supervisory system. The network manager logical unit never initiates communications itself. Transmissions to and from the network manager are not acknowledged by a reply message, thus this logical unit does not require reply message creation for the supervisor.

Format of the message transmitted to the network manager logical unit is shown in Fig. 3.6. Different from all, the message data is conveyed with the qualifier bytes.

ID	**Function Code**	**Function Qualifier**
1 byte	1 byte	2 bytes
LUN=0	FC=0	Command Code
00000000	00000000	See table 3.4

Fig.3.6 LUN 0 Format

Table 3.4 Command Codes for Logical Unit 0

Code	Command
0	Abort the operation in progress.
1	Start the short status information on logical unit.

3.4.3 Short Status Information - LUN 1

The short status information logical unit is used to provide information to the external supervisory computer.

This logical unit is activated by a command to the network manager logical unit. When this logical unit is activated, it immediately transmits a status message to the supervisor. Thus LUN 0 is used to trigger LUN 1. Fig. 3.7 shows the format of the records transmitted by LUN 1.

ID	**Function Code**	**Function Qualifier**	**Message Data (Status Message)**
1 byte	1 byte	2 bytes	14 bytes
LUN=1	FC=3	FQ=0	See Fig. 3.8
00000001	00000011	00000000 00000000	Unknown

Fig.3.7 LUN 1 Format

Message Data / Status Message													
1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte	11th byte	12th byte	13th byte	14th byte
System status bytes (See Table 3.5)	Status Bits	First 8 characters of the user program currently executing.								Error code for the terminated execution.		Loops completed for the program execution	

Fig.3.8 Message Data / Status Message

Table 3.5 System Status Byte Indications

Value of 1st Byte	Meaning
0 {00000000}	MANUAL MODE is displayed on the teach pendant.
1 {00000001}	ARM POWER is turned off.
2 {00000010}	A user program is executing
3 {00000011}	The system is in HOLD mode.
4 {00000100}	A "fatal" error has occurred
5 {00000101}	COMPUTER Mode is selected on the teach pendant.
6 {00000110}	The user program is waiting for terminal input.
7 {00000111}	The user program is in a WAIT state.

Supervisor computer must reply LUN 1 messages (see Fig.3.9); the reply for the status message shall be two's complement negative value of the originally received message. A reply of "1" is used in order to indicate successful receipt of transmission.

1 byte	1 byte	2 bytes
LUN=1	FC=3+128 = 131	$(\text{Status message})'_2$
00000001	10000011	Unknown

Fig.3.9 Reply Format for LUN 1
(Created by the Supervisory Computer)

3.4.4 Monitor Input / Output - LUN 2

The monitor input and output logical unit serves the role of the VAL II system terminal while the supervisory system has control of the VAL II system. All monitor commands are input to the VAL II system through this logical unit and all output (except for that of LUN 3) is directed to this logical unit.

ID	Function Code	Function Qualifier	Message Data
1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC	FQ	MD

Fig.3.10 LUN 2 Format

The message data should be interpreted as ASCII text and contains embedded carriage return (CR) and line feed (LF) characters, to define individual lines of text. If the complete text to be transmitted is longer than 256 characters it is transmitted in 256-character segments. Thus, a line of text can begin in one record and be completed in the next.

Considering the description above following conversions shall be made in order to form the message data for the command "DO READY".

```
>> Command='DO READY';

>> ASCII_Command=double(Command)

ASCII_Command =
   68   79   32   82   69   65   68   89


>> Message_Data=dec2bin(ASCII_Command,8)
Message_Data =

01000100
01001111
00100000
01010010
01000101
01000001
01000100
01011001
```

Communications, which can occur through the monitor I/O logical unit, are described within this section with their reply messages. According to LUN 2 specifications, function code for this logical unit can be 1,2,3 or 4.

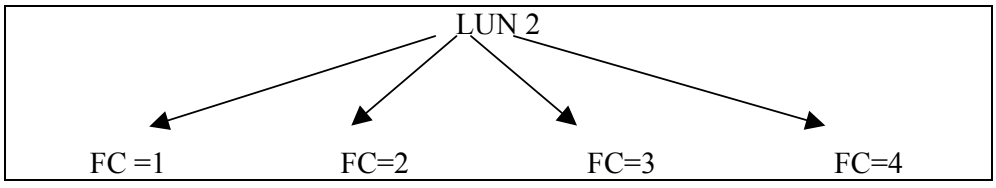


Fig.3.11 LUN 2 Having Four Function Codes

FC=1 (Abort Outstanding Communication)

When a message is received from VAL II with function code as 1, VAL II cancels the last transmission. Once this message is transmitted by VAL, the logical unit ignores all receptions until it receives a message containing an abort function code, afterwards normal communications can occur.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=1	FQ=0	None
00000010	00000001	00000000 00000000	N/A

Fig.3.12 VAL II transmits LUN=2 FC=1 Message

The binary representation of the "Abort Outstanding Communication" message is shown in Fig. 3.12, in which the representation of the message has been given according to the general format of LUN messages as shown with Fig. 3.3. The LUN byte (1st byte), function code FC (2nd byte), function qualifier FQ (3rd and 4th bytes) are present in the message but the message data field (the 4th column in Fig. 3.12) is not used for "Abort Outstanding Communication" message.

If the message (Fig. 3.12) with function code 1 is received, the supervisor shall respond with the message shown in Fig.3.13 in order to maintain normal communications, otherwise the communication will be broken by the supervisory unit of the controller assuming that there exists an error with communication.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=1+128 = 129	FQ=1	None
00000010	10000001	00000000 00000001	N/A

Fig.3.13 VAL II transmits LUN=2 FC=1 Message
(Created by the Supervisory Computer)

The binary representation of the reply to the "Abort Outstanding Communication" message received from the PUMA is shown in Fig. 3.13, in which the representation of the message has been given according to the general format of LUN messages as shown with Fig. 3.3. The LUN byte (1st byte), function code FC (2nd byte), function qualifier FQ (3rd and 4th bytes) are present in the message but the message data field (the 4th column in Fig. 3.13) is not used when replying the "Abort Outstanding Communication" message since there is no need for any data transmission for the acknowledgment of the received message.

FC=2 (Read Data)

Function code with 2 is used by VAL II to request input from the supervisor computer. The value of the function qualifier determines the type of response expected, if it has a value less than or equal to zero, VAL II expects a new monitor command otherwise a reply to a previous prompt is requested.

A read data transmission, which is transmitted by the PUMA controller actually demands a command from the supervisory external computer, occurs as soon as VAL II has completed processing the preceding command and is ready to accept another. A read data transmission is used by the PUMA, in order to request a command / message or to indicate that a new command / message can be evaluated and executed.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=2	FQ (variable)	None
00000010	00000010	Unknown	N/A

Fig.3.14 VAL II transmits LUN=2 FC=2 Message

The binary representation of the to the "Read Data" message received from the PUMA is shown in Fig. 3.14, in which the representation of the message has been given according to the general format of LUN messages as shown with Fig. 3.3. The LUN byte (1st byte), function code FC (2nd byte), function qualifier FQ (3rd and 4th bytes) are present in the message but the message data field (the 4th column in Fig. 3.14) is not used, since PUMA does not transmit a message in its request of a command or a message. The message data field is used in the "Read after Write" (Fig. 3.18) messages since those messages contain data, which are sent to the external supervisory computer, in order to get user or operator response.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=2+128 =130	FQ	Data / Command E.g.: DO READY (8 byte command)
00000010	10000010	00000000 00000001	01000100 01001111 00100000 01010010 01000101 01000001 01000100 01011001

Fig.3.15 Reply Message for LUN=2 FC=2 Message

The message created by the supervisor for the 'DO READY' command is shown in Fig.3.15.

Note that if the VAL II INTERACTIVE system switch is disabled, several types of messages are suppressed. For example, messages, which are uniquely identified by the function qualifier, are omitted. Also, queries such as "Are you sure (Y/N)?" can be suppressed such that they are not issued, and all header information is eliminated both from the output and from operations such as NET, STATUS, and WHERE.

FC=3 (Write Data)

This communication with function code 3, outputs an error message or informational message from VAL II to the supervisor computer.

The function qualifier, FQ represents error codes, which are always two's-complement negative numbers. If an error message is being output, for example, 'ARM POWER off' then the corresponding error code for this error is –311 as function qualifier.[1,2,3] The supervisory computer can use this error code to identify the error without interpreting the text of the error message but this feature has not been implemented within the software of the supervisor computer.

The function qualifier is zero or a positive number when the message is informational (E.g.: A reply for the command 'WHERE', which would be transmitted to the supervisor computer via function code 3, would have function qualifier as 0 since it is an informational in nature).

Figs.3.16 and 3.17 show the messages received from VAL and the reply message created by the supervisor computer accordingly.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=3	FQ (variable)	Message
00000010	00000011	Unknown	Unknown

Fig.3.16 VAL II transmits LUN=2 FC=3 Message

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=3+128 =131	FQ=1	None
00000010	10000011	00000000 00000001	N/A

Fig.3.17 Reply Message for LUN=2 FC=3 Message

The message received shall be converted to ASCII characters in order to inform the user with the message received. Thus the following conversions shall be made within the supervisory communication software.

```
>>Received_Message_binary =

00101010 01000001 01010010 01001101 00100000 01010000 01001111 01010111   01000101
01010010 00100000  01101111  01100110 01100110
00101010

>> ASCII_Received_Message=bin2dec(Received_Message_binary)'

ASCII_Received_Message =

 Columns 1 through 10

   42   65   82   77   32   80   79   87   69   82

 Columns 11 through 15

   32  111  102  102   42

>> char(ASCII_Received_Message)

ans =

*ARM POWER off*
```

FC=4 (Read After Write)

VAL II outputs a message and requests input of a reply. Similar to function code 3 communication explained above but the supervisor is expected to provide a message in its reply. Similar to function code 3, some messages received via function code 4 have predefined message codes that can be used by the supervisor to identify the message but message but this feature is not implemented within the software of the supervisor computer for this thesis work.

Figs.3.18 and 3.19 show the messages received from VAL and the reply message created by the supervisor computer according to the received message correspondingly.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=4	FQ (variable)	Message
00000010	00000100	Unknown	Unknown

Fig.3.18 VAL II transmits LUN=2 FC=4 Message

1 byte	1 byte	2 bytes	0-256 bytes
LUN=2	FC=4+128 =132	FQ=1	Reply Message
00000010	10000100	00000000 00000001	Unknown

Fig.3.19 Reply Message for LUN=2 FC=4 Message

Messages like "Are you sure (Y/N)?" are sent to the supervisor using the LUN 2 with function code 4 and the supervisor shall respond accordingly using the same LUN with function code 4.

3.4.5 Monitor Asynchronous Output - LUN 3

LUN 3 used by VAL II in order to transmit monitor outputs during executing of user programs. Those outputs include messages indicating the beginning and end of program execution and any error messages that might be output during the execution. Error messages, which result from hardware malfunctions (E.g.: ″∗[Fatal] Servo dead∗"), are reported through this logical unit.

The function code for this logical unit is 3 since only the writing data function is used, which means only messages are transmitted to the supervisory computer. Fig.3.20 shows the format for the LUN 3.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=3	FC=3	FQ (variable, see table 3.6)	Message Data
00000011	00000011	Unknown	Unknown

Fig.3.20 VAL II transmits LUN=3 Message

Table 3.6 FQ Value and Corresponding Meaning for LUN 3

FQ value	Description
2	(No message data) LUN 3 thus VAL, transmits a message when an execution of user program begins.
3	Program Completed.
4	This message is transmitted when the program execution terminates. Transmitted message carries the program name and the stopped step number of the program
5	Whenever a DO command is processed, a message is transmitted with this FQ, no message data exist for this FQ.
6	Whenever processing of a DO command is completed, a message is transmitted with this FQ, no message data. Exist for this FQ.
7	In TRACE mode, VAL transmits a message for each instruction step executed. The message data is the instruction step itself.

8	This message is transmitted when a PAUSE instruction is executed.
9	This message is transmitted when a HALT instruction is executed.
12	Similar to FQ=5, but this time instead of a Do command, this message is transmitted when a user program is executed.
13	Similar to FQ=6, but this time instead of a Do command, this message is transmitted when a user program is completed.
14	Similar to FQ=7, but this message is transmitted for the user program.

The supervisor computer shall reply to the VAL system as shown with Fig.3.21. The value for the function qualifier shall be set to 1 in order to indicate acknowledgement.

1 byte	1 byte	2 bytes
LUN=3	FC=3+128 =131	FQ = 1
00000011	10000011	00000000 00000001

Fig.3.21 Reply Message for LUN=3

3.4.6 Program Terminal Input and Output - LUN 4

This logical unit is similar to LUN 2, but it is only used for program I/O instead of monitor I/O. All requests for program input instruction and program output are directed to the supervisory computer via logical unit 4. Fig.3.22 shows the format for LUN=4.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC (variable)	FQ (variable)	Message Data
00000100	Unknown	Unknown	Unknown

Fig.3.22 Format for the LUN 4 Message

Communications, which can occur through the Program terminal I/O logical unit, are described within this section with their reply messages. According to LUN 4 specifications, function code for this logical unit can be 1,3 or 4.

FC=1 (Abort Outstanding Communication)

When a message is received from VAL II with function code as 1, VAL II cancels the last transmission. Once this message is transmitted by VAL, the logical unit

ignores all receptions until it receives a message containing an abort function code, afterwards normal communications can occur.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=1	FQ=0	None
00000100	00000001	00000000 00000000	N/A

Fig.3.23 VAL II transmits LUN=4 FC=1 Message

If the message (Fig.3.23) with function code 1 is received, the supervisor shall reply with the message shown with Fig.3.24 in order to enable normal communications.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=1+128 = 129	FQ=1	None
00000100	10000001	00000000 00000001	N/A

Fig.3.24 VAL II transmits LUN=4 FC=1 Message
(Created by the Supervisory Computer)

FC=3 (Write Data)

This communication with function code 3 transmits the output specification of a TYPE instruction.

Figs.3.25 and 3.26 show the messages received from VAL and the reply message created by the supervisor computer correspondingly.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=3	FQ= 0	Message
00000100	00000011	00000000 00000000	Unknown

Fig.3.25 VAL II transmits LUN=4 FC=3 Message

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=3+128 =131	FQ=1	None
00000100	10000011	00000000 00000001	N/A

Fig.3.26 Reply Message for LUN=4 FC=3 Message

FC=4 (Read After Write)

VAL II outputs a message and requests input of a reply. Similar to function code 3 communication explained above but the supervisor is expected to provide a message in its reply.

Figs.3.27 and 3.28 show the messages received from VAL and the reply message created by the supervisor computer according to the received message correspondingly.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=4	FQ=0	Message
00000100	00000100	00000000 00000000	Unknown

Fig.3.27 VAL II transmits LUN=4 FC=4 Message

1 byte	1 byte	2 bytes	0-256 bytes
LUN=4	FC=4+128 =132	FQ=1	Reply Message
00000100	10000100	00000000 00000001	Unknown

Fig.3.28 Reply Message for LUN=2 FC=4 Message

Supervisory computer software did consider the program terminal input and output logical unit, thus it can reply to the received messages from LUN 4 but since VAL programming from an external computer was not our main concern, this logical unit has not been tested.

3.4.7 Disk Input and Output - LUN 5

The information between VAL II and the Unimation floppy disk drive is actually communicated via LUN 5. Thus supervisory computer and VAL use this logical unit when the DISK.NET switch is enabled.

Fig. 3.29 shows the format of logical unit 5.

1 byte	1 byte	2 bytes	0-256 bytes
LUN=5	FC (variable)	FQ (variable)	Message Data
00000101	Unknown	Unknown	Unknown

Fig.3.29 Format for the LUN=5 Message

The function code and function qualifier in the messages transmitted convey the disk operation performed. All the possible disk operations are given with Table 3.7 below with function codes and qualifiers for each operation. The table below only represents the received messages from the supervisor side of view.

Table 3.7 Possible Disk Operation Using LUN 5

FC Value	FQ Value	Description
1	0	Abort Outstanding Communication.
0	0	Close the file or directory that was opened last.
0	1	Open the file with the specified filename, which shall be obtained from the message data bytes.
0	2	Open a file for writing data with the specified filename, which shall be obtained from the message data bytes.
0	4	Open the disk directory for reading.
0	5	Delete a file with the specified filename, which shall be obtained from the message data bytes.
0	7	Compressing the disk operation directs the Unimation's floppy disk controller to repack the files and recover inaccessible space.
0	8	Format the disk directs the Unimation's floppy disk controller to format a new diskette.
2	0	Requesting a block of data from the disk. The block of data (256 bytes) shall be sent via the reply message's message data.
3	0	Writes a block of data, received within the message data, to the disk.

LUN 5 properties were added to the supervisory control, but did not work, while no error messages were received also. Possible problem, which cannot be proved, is that the DISK.NET switch does not respond.

3.5 DDCMP

The Digital Data Communications Message Protocol (DDCMP) is a data link control procedure ensuring a reliable data communication path between communications devices connected by data links. DDCMP has been designed to operate over full-duplex and half-duplex synchronous and asynchronous channels in both point-to-point and multipoint modes. It can be used in a variety of applications such as distributed computer networking, host/front-end processing, remote terminal concentration, and remote job entry-exit system operation [15].

For DDCMP this thesis work actually used two message formats:

- Control Messages
- Data Messages

Other than those, maintenance messages exist in order to cover messages with error.

3.5.1 Control Messages

Control messages carry:

- Channel information,
- Transmission
- Status,
- Initialization notification; between the protocol modules.

The individual fields are specific for each type of control message. Control messages have the following general form:

ENQ	TYPE	SUBTYPE	FLAGS	RCVR	SNDR	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.30 General Form of Control Messages (DDCMP)

Table 3.8 Control Message Field Descriptions

ENQ	The control message identifier, having a constant value of 5.
TYPE	The control message type. This value denotes the type of each control message.
SUBTYPE	The subtype or type modifier field, providing additional information for some message types. Its use is specific for each message type.
FLAGS	The link flags, which are used to control link ownership and message synchronization. bit 0 = Quick sync flag (QSYNC flag), used to notify the receiver that the next message will not abut this message and resynchronization should follow this message. The quick sync flag reduces the length of sync sequences on synchronous links. bit 1 = Select flag (SELECT flag), used to control transmission ownership on multipoint and half-duplex links. Reverses link direction on half-duplex links. Invites a tributary to send and signals end of tributary selection on multipoint links.
RCVR	The control message receiver field, which is used to pass information from the data message receiver or slave station to the

	data message sender or master station. Its use is specific for each control message type.
SNDR	The control message sender field, which is used to pass information from the data message sender or master to the data message receiver or slave. Its use is specific for each control message type.
ADDR	The station address field. For our purpose that we use DDCMP for point-to-point link, this field is equal to 1. But in order to create a control network this field shall be used.
BLKCK3	The block check value calculated by CRC (Cyclic Redundancy Check) method on fields ENQ through ADDR.

SUBTYPE and FLAGS form a 2-byte quantity. The first byte contains the 8 low-order bits of the SUBTYPE. The second byte contains the 6 high-order bits of the SUBTYPE, the SELECT flag the highest order or most significant bit of the byte, and the QSYNC flag the next bit in the byte. Thus the sequence shall be S | Q | Subtype.

There are control messages, which are used within the supervisory software. These are:

- Start Message (STRT)
- Start Acknowledge Message (STACK)
- Acknowledge Message (ACK)
- Negative Acknowledge Message (NAK)
- Reply To Message Number (REP)

Start Message (STRT)

The STRT message is used to establish initial contact and synchronization on a DDCMP link. It is used only on link startup or re-initialization. It operates with the start acknowledge message STACK, thus a STACK message shall be sent to the STRT initiator to continue messaging. The start sequence resets message numbering at the transmitter and addressed receiver. The form of the STRT message is:

ENQ	STRTTYPE	STRTSUB	FLAGS	FILL	FILL	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.31 STRT Message Format

Using the DDCMP protocol specification, STRT message would become (Fig.3.32):

ENQ	STRTTYPE	STRTSUB	FLAGS	FILL	FILL	ADDR	BLKCK3
5	6	0	3	0	0	1	CRC Value

Fig.3.32 STRT Message (Integer Representation, Base 10)

Using the data in Fig.3.32, we can obtain the binary representation of the STRT message as:

00000101 | 00000101 | 11000001 | 00000000 | 00000000 | 00000001 | CRC (2 bytes)

Which corresponds to

5 | 6 | 192 | 0 | 0 | 1 | CRC (2 bytes)

CRC generation is described at following section of this chapter. A function (crcoeg) has been developed in order to calculate the CRC value for a given set of values as well.

It shall be noted that, while calculating CRC, header fill bits (16 bits) shall be added since those bits are included at the supervisory communication. Those fill bits are all '1's thus they correspond to 255 | 255 as the integer representation for the first 2 byte fields.

```
>> CRC=crcoeg([255 255 5 6 192 0 0 1])

CRC =

    0    1    1    1    0    1    0    1
    1    0    0    1    0    1    0    1

>> binvec2dec(fliplr(CRC(1,:)))

ans =

   117

>> binvec2dec(fliplr(CRC(2,:)))

ans =

   149
```

Using the calculated CRC value, STRT message can be finalized as:

5 | 6 | 192 | 0 | 0 | 1 | 117 | 149 in integer representation or
00000101 | 00000101 | 11000001 | 00000000 | 00000000 | 00000001 | 01110101| 10010101
in binary representation, which is actually used via communication.

Start message is actually formed and sent by the VAL II system waiting for an acknowledgement, which is the start acknowledgement message.

Start Acknowledge Message (STACK)

The STACK message is returned in response to a STRT when the VAL II system has completed initialization and has reset its message numbering. The form of the STACK message is (Fig. 3.33):

ENQ	STACKTYPE	STCKSUB	FLAGS	FILL	FILL	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.33 STACK Message Format

Using the DDCMP protocol specification, STACK message would become:

ENQ	STACKTYPE	STACKSUB	FLAGS	FILL	FILL	ADDR	BLKCK3
5	7	0	3	0	0	1	CRC Value

Fig.3.34 STACK Message (Integer Representation, Base 10)

Using the data in Fig.3.33, we can obtain the binary representation of the STACK message as:

00000101 | 00000110 | 11000001 | 00000000 | 00000000 | 00000001 | CRC (2 bytes)

Which corresponds to

5 | 7 | 192 | 0 | 0 | 1 | CRC (2 bytes)

```
CRC=crcoeg([255 255 5 7 192 0 0 1])

CRC =

   0   1   0   0   1   0   0   0
```

```
   0    1    0    1    0    1    0    1

>> binvec2dec(fliplr(CRC(1,:)))

ans =

   72

>> binvec2dec(fliplr(CRC(2,:)))

ans =

   85
```

Using the calculated CRC value, STACK message can be finalized as:

5 | 7| 192 | 0 | 0 | 1 | 72 | 85 in integer representation or

00000101 | 00000110 | 11000001 | 00000000 | 00000000 | 00000001 | 01001000 | 01010101

in binary representation, which is actually used by the communication.

Acknowledge Message (ACK)

The ACK message is used to acknowledge the correct receipt of numbered data messages. It conveys the same information as the RESP field of the control message (see next section for the control message) in numbered messages and is used when acknowledgements are required, and when no numbered messages are to be sent in the reverse direction. The form of the ACK message is:

ENQ	ACKTYPE	ACKSUB	FLAGS	RESP	FILL	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.35 ACK Message Format

Using the DDCMP protocol specification, ACK message would become:

ENQ	ACKTYPE	ACKSUB	FLAGS	RESP	FILL	ADDR	BLKCK3
5	1	0	3	Value	0	1	CRC Value

Fig.3.36 ACK Message (Integer Representation, Base 10)

RESP stands for the response number used to acknowledge correctly received messages, thus this information is obtained from the incoming data packet.

Using the data in Fig.3.36, we can obtain the binary representation of the ACK message as:

00000101 | 00000001 | 11000000 | $(RESP)_2$ | 00000000 | 00000001 | CRC (2 bytes)

Which corresponds to:

5 | 12| 192+REASON | RESP | 0 | 1 | CRC (2 bytes).

For this set of bits represented with base 10, we cannot calculate the CRC value thus the NAK message similar to ACK, since the NAK message is generated according to the data packet received with error(s).

For this set of bits represented with base 10, we cannot calculate the CRC value thus the ACK message, since the ACK message is generated according to the data packet received.

Negative Acknowledge Message (NAK)

The NAK message is used to pass error information from the data receiver to the data sender. The error reason is included in the subtype field. The NAK message also includes the same information as the ACK message, thus serving two functions: acknowledging previously received messages and notifying the master of some error condition. The form of the NAK message is:

ENQ	NAKTYPE	REASON	FLAGS	RESP	FILL	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.37 NAK Message Format

Using the DDCMP protocol specification, NAK message would become:

ENQ	NAKTYPE	REASON	FLAGS	RESP	FILL	ADDR	BLKCK3
5	2	Value	3	Value	0	1	CRC Value

Fig.3.38 NAK Message (Integer Representation, Base 10)

REASON field identifies the source and reason for the negative acknowledgement. The value of this field is used for error recovery.
Using the data in Fig.3.38, we can obtain the binary representation of the NAK message as:

00000101 | 00000010 | 11(REASON)$_2$ | (RESP)$_2$ | 00000000 | 00000001 | CRC (2 bytes)

Which corresponds to:

5 | 12| 192+REASON | RESP | 0 | 1 | CRC (2 bytes).

For this set of bits represented with base 10, we cannot calculate the CRC value thus the NAK message similar to ACK, since the NAK message is generated according to the data packet received with error(s).

Reply To Message Number (REP)

The REP message is used to request received message status from the data receiver. It is usually sent when the data transmitter has transmitted data messages and has not received a reply within a time-out period. The response to a REP is either an ACK or NAK depending on whether the receiver has or has not received all messages previously sent by the transmitter. The form of the REP message is:

ENQ	REPTYPE	REPSUB	FLAGS	FILL	NUM	ADDR	BLKCK3
8-bits	8-bits	6-bits	2-bits	8-bits	8-bits	8-bits	16-bits

Fig.3.39 REP Message Format

Using the DDCMP protocol specification, REP message would become:

ENQ	REPTYPE	REPSUB	FLAGS	FILL	NUM	ADDR	BLKCK3
5	3	0	3	0	Value	1	CRC Value

Fig.3.40 REP Message (Integer Representation, Base 10)

NUM field used within this control message represents the number of the last sequential numbered data message sent by the transmitter, excluding the re-transmitted ones.

Similar to ACK and NAK we are not able to generate the REP message without the value of the NUM field.

3.5.2 Data Messages

Data Messages carry user data over DDCMP links; each data message also carries the number assuring correct massage sequencing. The starting number is initialized with control messages STRT and STACK, which are mentioned above. For each data massage, the receiver must acknowledge the transmitter for the correct receipt of data message, using the sequence number. The form of data message is:

SOH	COUNT	FLAGS	RESP	NUM	ADDR	BLKCK1	DATA	BLKCK2
8-bits	14-bits	2-bits	8-bits	8-bits	8-bits	16-bits	X-bytes	16-bits

Fig.3.41 Data Message Format

Table 3.9 Data Message Field Descriptions

SOH	The numbered data message identifier. It has a constant value of 129.
COUNT	The byte count field. It specifies the number of 8-bit bytes in the DATA field. The value zero is not allowed.
FLAGS	The link flags, which are used to control link ownership and message synchronization. bit 0 = Quick sync flag (QSYNC flag), used to notify the receiver that the next message will not abut this message and resynchronization should follow this message. The quick sync flag reduces the length of sync sequences on synchronous links. bit 1 = Select flag (SELECT flag), used to control transmission ownership on multipoint and half-duplex links. Reverses link direction on half-duplex links. Invites a tributary to send and signals end of tributary selection on multipoint links.
RESP	The response number, which is used to acknowledge correctly received messages.
NUM	The transmit number, which is used to denote the number of this data message.
ADDR	The station address field. For our purpose that we use DDCMP for point-to-point link, this field is equal to 1. But in order to create a control network this field shall be used.
BLKCK1	The block check value calculated by CRC (Cyclic Redundancy Check) method on fields SOH through ADDR.
DATA	The numbered message data field. This field is totally transparent

	to the protocol and has no restrictions on bit patterns, groupings, or interpretations. The only requirement is that it contains the number of 8-bit bytes specified in the COUNT field.
BLKCK 3	The block check value calculated by CRC (Cyclic Redundancy Check) method on the DATA field only.

COUNT and FLAGS form a 2-byte quantity. The first byte contains the 8 low-order bits of the COUNT. The second byte contains the 6 high-order bits of the COUNT, the SELECT flag the highest order or most significant bit of the byte, and the QSYNC flag the next bit in the byte. Thus the sequence shall be S | Q | Count.

After synchronization and acknowledgement of communications between supervisory computer and the PUMA robot control system data messages are used in order to control the robot. VAL sends its messages using those data messages as well. In order to send a "DO READY" command supervisory computer must do the following:

a) Convert Command into ACII code,
b) Insert converted command into VAL's logical units,
c) Pack the logical unit into DDCMP (CRC calculation required),
d) Transmit the DDCMP packet.

```
>> Command='DO READY';

>> CommandDecimal=double(Command) % Convert Command to its Decimal Representation

CommandDecimal =

   68   79   32   82   69   65   68   89
```

"DO READY" command when fitted into LUN 2 can be represented as shown in Fig. 3.42:

2	130	0	1	68	79	32	82	69	65	68	89

Fig.3.42 Decimal Representation of "DO READY" in LUN 2

Using the DDMP data message structure (Fig.3.41), and the overall frame structure, we obtain the packet as shown with Fig.3.43. Note that, NUM and RESP fields are assumed to be 1 and 2 respectively and the COUNT is equal to 12.

255	255	129	12	192	0	1	1	BLKCK 1	…

…	2	130	0	1	68	79	32	82	69	…

…	65	68	89	BLKCK2	255	255

Fig.3.43Decimal Representation of "DO READY" command, which is fitted into a DDCMP packet without CRC values.

DDCMP data messages as described earlier, use CRC for error checking, which consume time. As seen from previous figure (Fig. 3.43), BLKCK 1 and BLKCK 2 shall be calculated in order to complete the total packet. We have developed a fast CRC calculation function in our studies, this is a function called "crcoeg", which calculates CRC value for a given set of data and its output is binary.

In order to calculate BLKCK 1 (2-bytes) value for a DDCMP packet, the function (crcoeg) takes the first 8 bytes of the packet and outputs the result of the BLKCK 1 in binary representation in which the first 8-bits of the result is the 1st byte and the second 8-bits of the result is the second byte of the BLKCK 1 field (Fig. 3.43). The process is actually the same for BLKCK 2 but this time the CRC calculation function takes the data field of the DDCMP packet, which is the 12-bytes after the BLKCK 1 field for the representation in Fig. 3.43.

```
>> BLKCK1=crcoeg([255 255 129 12 192 0 1 1])        } Function call for BLKCK1.

BLKCK1 =                                            } BLKCK 1 value in binary

   1   1   1   1   0   0   1   0
   0   1   0   0   0   0   0   0

>>BLKCK1_1stByteInDecimal=(binvec2dec(fliplr(BLKCK1(1,:))))   } BLKCK1 1st byte in decimal
BLKCK1_1stByteInDecimal = 242

>>BLKCK1_2ndByteInDecimal=(binvec2dec(fliplr(BLKCK1(2,:))))   } BLKCK1 2nd byte in decimal
BLKCK1_2ndByteInDecimal =64

For BLKCK 2 we use the function 'crcoeg' as well.

>> BLKCK2=crcoeg([2 130 0 1 68 79 32 82 69 65 68 89])

BLKCK2 =

   1   0   0   0   0   0   0   1
   1   0   1   0   0   1   1   0
```

```
>>BLKCK2_1stByteInDecimal=(binvec2dec(fliplr(BLKCK2(1,:))))

BLKCK2_1stByteInDecimal =129

>>BLKCK2_2ndByteInDecimal=(binvec2dec(fliplr(BLKCK2(2,:))))

BLKCK2_2ndByteInDecimal =166
```

The program code of the "crcoeg" function that calculates CRC for a block check is given below related to (BLKCK1 and BLKCK2) parameters in the DDCMP packet.

```
%                       CRC CALCULATOR                                  %
%                       Developed By Onder E. GEBIZLIOGLU               %
%                                                                       %
%                                                                       %
%                                          Modified @ 04/04/2003        %
%                                                                       %
% INPUT & OUTPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT is a decimal array of size (1,X) : outdata
% OUTPUT is crcf where crcf(1,:) is the 1st 8_bits and crcf(2,:) the 2nd
% 8_bits total of 16_bits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Functions used are appzero and shiftone (thus appzero.m % shiftone.m must
% be in the same file folder with crcoeg.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function crcf=crcoeg(outdata)

[q,q1]=size(outdata);

for i=1:1:q1
   out(i,:)=dec2binvec((outdata(i)*2),9);
end

%Initialize CRC variablescrc=dec2binvec(hex2dec('fffffe'));
v=dec2binvec(hex2dec('1fffe'));
w=dec2binvec(hex2dec('A001')*2);
one= dec2binvec(hex2dec('ffffff'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CRC ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:q1
   r=appzero(crc,out(i,:));
   crc=xor(r(1,:),r(2,:));

   for xx=1:1:8
      r1=appzero(crc,v);
      crc=and(r1(1,:),r1(2,:));
      s=shiftone(crc);
      crc=s(2,:);
        if  bitand(binvec2dec(crc),1)
```

```
            r4=appzero(crc,w);
            crc=xor(r4(1,:),r4(2,:));

        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rw=appzero(crc,dec2binvec(hex2dec('1fe')));
% First 8 Bits
crcf(1,:)=fliplr(dec2binvec((binvec2dec(and(rw(1,:),rw(2,:)))/2),8));
% Second 8 Bits
crcf(2,:)=fliplr(dec2binvec((binvec2dec(and(rw(3,:),rw(4,:)))/512),8));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

According to the DDMP, the algorithm for computing the cyclic redundancy check is as follows [15]:

a) Consider the header or data portion of the message as it appears on a serial line (LSB of the first byte first, MSB of the final byte last) and append 16 zeros after the header or data.

b) Take the string of bits constructed in 1, and treat each bit as the coefficient of a term of a polynomial with the LSB of the first byte being the coefficient of the highest order polynomial term. The highest order term is A * X^63 for a header block and A * X^(8 * <count> + 15) for a data block where A is the least significant bit of the first byte of the header or data. The lowest order term is 0 * X^0 for both cases.

c) Divide the polynomial constructed in 2, by the CRC-16 polynomial X^16 + X^15 + X^2 + 1 using synthetic division and using modulo 2 arithmetic on the coefficients obtaining a quotient that is discarded and a 16-bit remainder.

d) Transmit the coefficients of the remainder as the block check bytes following the original message bits, transmitting the coefficients of the highest order term (X^15) first. Thus, the first byte represents coefficients of the X^8 through X^15 terms of the remainder (from left to right) and the second byte represents coefficients of the X^0 through X^7 terms of the remainder (also from left to right).

This CRC error-checking algorithm runs both on the transmitter and on the receiver sides of the communication, thus communicating sides shall both calculate the CRC values for incoming and outgoing packets. They perform the same algorithm and compare the received block check bytes with the computed block check bytes. If the bytes are not identical, an error has occurred. However, we did note during our studies that CRC calculation for the incoming packets was unnecessary for the supervisor side, since no communication error was met on that side. Thus we have

assumed that the messages sent from PUMA were received totally correct. This assumption, which was tested to be correct for our robot's workspace later, has decreased the number of calculations for each transaction of a particular command that supervisory computer should handle by 25%

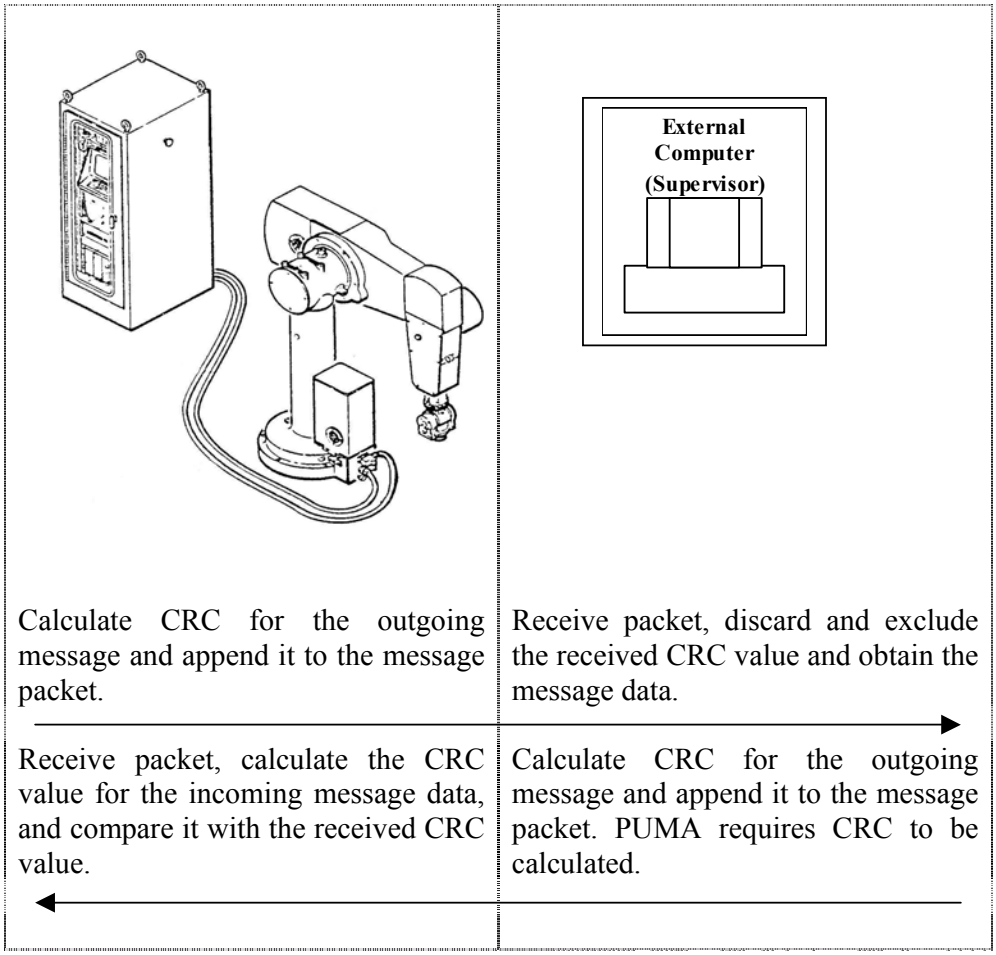


Fig.3.44 CRC Calculation Status for the Communication

3.6 Supervisory Software

According to the supervisory rules and assumptions described previously, supervisory software has been designed and implemented. This software enables the user to send VAL commands directly to the PUMA robot just like the monitor (controller) commands within the PUMA system.

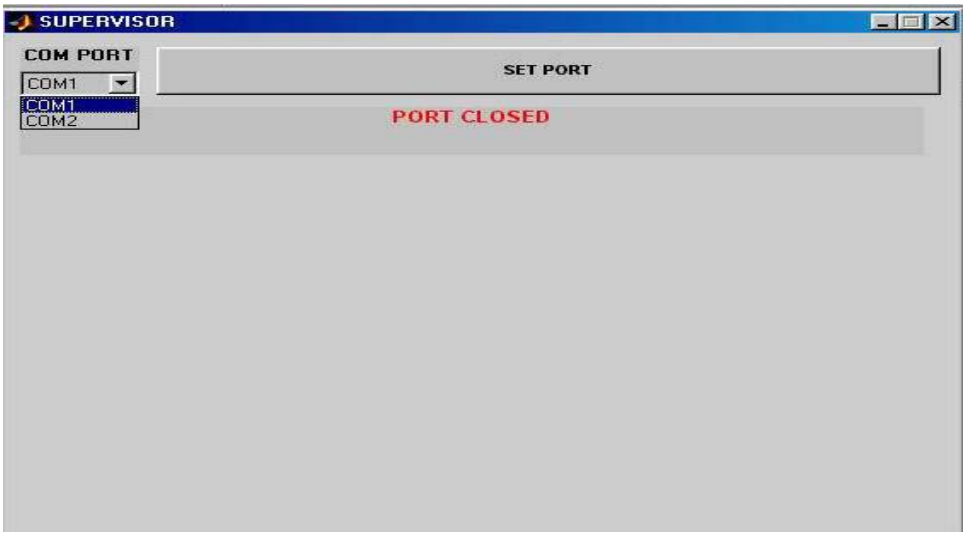


Fig.3.45 Supervisory Communication Port Selection

Supervisory communication is provided through Com 1 or Com 2 ports of the supervisory computer, which can be selected using the supervisory software initial graphical user interface shown in Fig.3.45. Once the port is selected the communication port settings are conducted automatically according to the supervisory port specifications declared by the Unimation.

The baud rate for the serial line for the supervisory communication is 9600 baud, which is set by the PUMA producer Unimation. The communication data are transmitted as 8-bit bytes, with one parity bit for each byte and no parity information is included in the data bytes. After the selection of the communication port (COM 1 | COM 2), the supervisory system sets up the communication parameters according to this information given above. The code setting the parameters for the communication port is given below.

```
set(s,'BaudRate',9600)
set(s,'StopBits',1)
set(s,'Parity','none')
set(s,'BreakInterruptFcn','')
set(s,'ByteOrder','littleEndian')
set(s,'BytesAvailableFcn','')
set(s,'BytesAvailableFcnMode','byte')
set(s,'DataBits',8)
set(s,'DataTerminalReady','on')
set(s,'ErrorFcn','')
set(s,'FlowControl','none')
set(s,'ReadAsyncMode','continuous')
set(s,'RequestToSend','on')
set(s,'Terminator','')
```

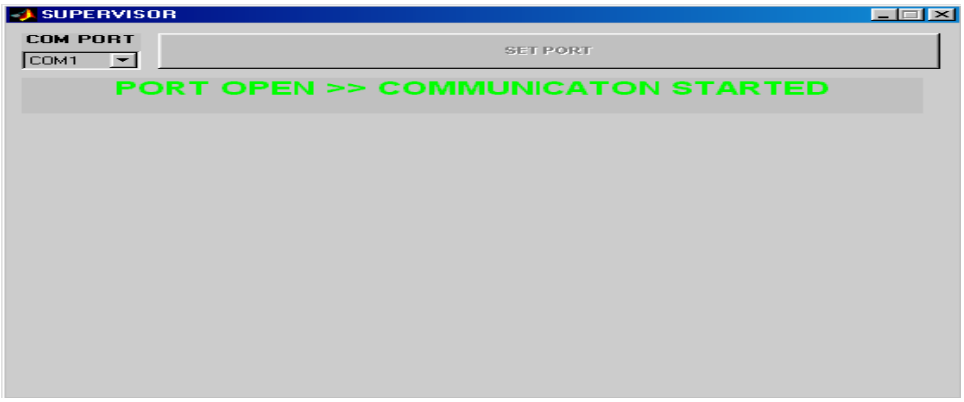


Fig.3.46 Communication Initialization

Once the port is opened the supervisory software will be waiting for the PUMA supervisory communication to initialize the communication according to the control messaging (STRT message) of DDCMP protocol, which is described within this chapter.

Note that the supervisory computer software will cause a timeout error when the communication does not start within the selected timeout period.

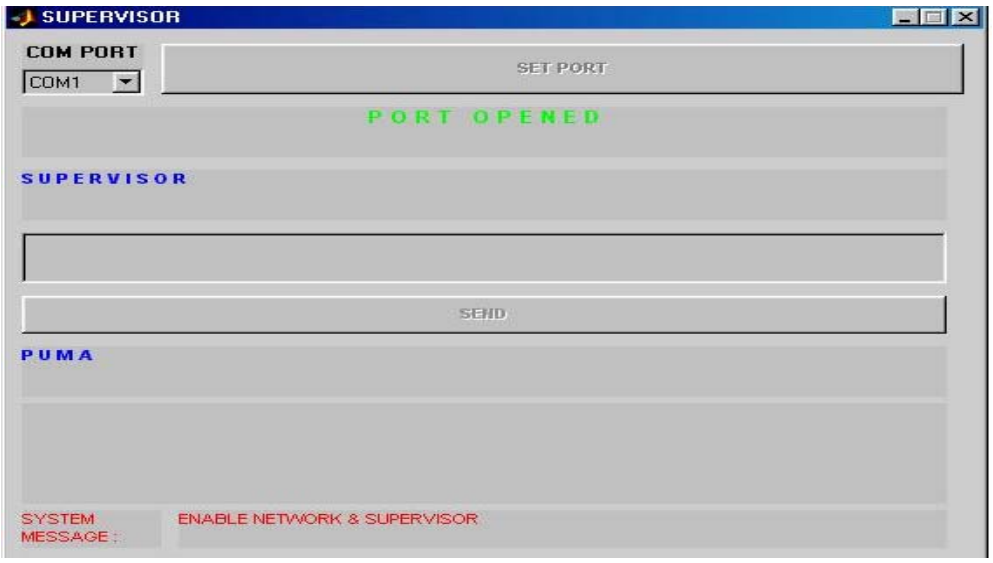


Fig.3.47 Supervisory Software Messaging User Interface

The supervisory software opens the supervisory software messaging GUI for command and data messaging between the supervisory computer and the PUMA robot system, which enables all VAL commands to be transmitted and executed by the robotic arm.

Fig.3.48 Supervisory Message / Command Box

The supervisory commands or messages to be transmitted are to be entered to the Message / Command Box within the supervisory software messaging screen. Fig. 3.48 shows the message / command box which holds the message or the command (e.g. “CALIBRATE”) to be transmitted to PUMA. The commands entered are to be executed or replied by the PUMA. It shall be noticed that the messages or commands, which are entered within this dialogue, shall be complete VAL messages, thus VAL and robotic knowledge is required to use the supervisory software but a control software is also implemented above the supervisory software (explained within Chapter 4) which does not require the operator or the user to have the knowledge of VAL. This control software is described with the following chapter.

Fig.3.49 PUMA System Messages

The messages and requests sent by the PUMA (if any) are displayed within the supervisory software-messaging screen as well. Those messages may require the user to take action for the execution of a specific command. Also user requests such as “WHERE” are replied and displayed by the PUMA system message box. Note that most of the commands and requests of PUMA system are handled by the supervisory software thus they are not displayed. Fig. 3.49 displays a message “Are You Sure? (Y/N) received from VAL II. The external supervisor must reply to this message in order to continue messaging otherwise VAL will go on standby, waiting for the reply of its request or message.

Fig.3.50 System Messages

The overall supervisory communication system messages are also displayed within the supervisory software-messaging screen. The system messages are important since they lead the user to take required action for the execution of the overall system. The system messages are most important when initializing the supervisory communication since the operator or the user has to take some necessary action such as enabling the network and the supervisory port of the PUMA controller. Fig. 3.50 shows the system messages box, which displays a system message "ENABLE NETWORK & SUPERVISOR", guiding the user to enable the network and supervisor ports of the controller thus the VAL.

3.6.1 Operation of the Supervisory Software

The supervisory software actually follows the rules of the DDCMP and the Logical Units, which are described previously within this chapter.

The DDCMP protocol has been totally implemented since it is the bottom layer protocol of the supervisory communication, which accepts a single transmission request from a middle layer and can simultaneously be reading a single message from the communication interface. The DDCMP is implemented using its features described in the related item of this chapter.

The logical units are also implemented as a whole within the supervisory software but LUN 0, LUN 4 and LUN 5 are not completely used since robot programming through the supervisory communication is handled with another control software (Chapter 4) which does not require the knowledge of VAL. According to this information the reply message comparison priority has been reformed and LUN 1, 2, 3 cases are compared and handled before the others. Although this seems to be a lack of the supervisory communication, the control software fills in the gaps enabling most of the VAL specifications and command to be executed and even beyond those VAL command set many more commands can be generated by the use of third party applications and the control software.

3.6.2 Advantages and Disadvantages of Supervisory Software

Supervisory software has all disadvantages that the VAL has since it is implemented on top of it, but it brings many more advantages since it is executed on a personal computer. Advantages that are specific to the supervisory communication can be listed as follows:

- Operates on a personal computer (Operating system free),
- Enables the integration of third party applications,
- Enables network control (requires more work),
- Enables the execution of all VAL monitor commands.

Specific robotic programming can increase advantages that supervisory communication brings. Thus those advantages listed above are the most basic advantages of the supervisory software and communication. In our studies we have developed control software that is integrated to the supervisory software and we managed to add many more special functions to the overall system. Discarding the integrated software modules implemented above the supervisory communication software, the disadvantages of the supervisory software can be listed as follows:

- Operation and execution of commands are slow,
- VAL robot programming is not enabled (handled by the control software),
- Requires VAL to start up and initialize the robotic system,
- Requires VAL and robotic knowledge (handled by the control software),
- Requires MATLAB to be installed on the supervisory computer,
- Requires high memory for better performance,
- Communication buffer problems may occur due to OS.

Most of the disadvantages are due to the programming language (MATLAB) and hardware performances, which can be overcome later on. But notice that the advantages it brings are more important since it enables more studies to be carried out on the robotic arm. Image processing, speech processing, network studies can be integrated to the robot via the supervisory communication software.

CHAPTER 4

4 CONTROL INTERFACE

4.1 General

In order to supply the user with a user-friendly, easy to use GUI, interactive controlling and simulation software has been developed. This interface actually collects user requests, and sends them to the supervisory system, which is mentioned in the previous chapter.

The supervisory program is integrated with an interface to the Robotic Toolbox written for MATLAB [16][17]. The toolbox has been modified enabling the user to send movement commands (joint angle movements) without the knowledge of VAL robot programming language.

In the beginning of the studies the control interface was designed and implemented with C++. This version of the control interface had only the manipulator kinematics calculation capability. The implementation was specific for the PUMA 700 series robot. Although it responds faster in runtime, the development with C++ was hard especially for user interface and any change within the design required time consuming rework. Thus the code was transferred to the MATLAB environment.

4.2 The Choice of MATLAB for the Robot Control Interface

The robot control interface has been implemented with MATLAB in order to use its benefits and advantages upon simulation, graphics and matrix calculations. MATLAB has become very popular within robotic applications since manipulator mathematics especially kinematics use matrix calculations very commonly.

4.2.1 Advantages and Disadvantages in Using MATLAB

As stated before the most important advantage of using MATLAB is the ease of programming for matrix calculations, since MATLAB is specifically designed for matrix operations and thus have built-in functions for matrices.

Advantages and benefits obtained using MATLAB can be listed as follows:

- Ease of matrix definitions and operations;
- Ease of programming;
- Ease of modification on existing code;
- C / C++ and Java code can be called and executed within the MATLAB code;
- Independent of operating system;
- Requires less training required to start using it, if compared with other programming environments and languages (Microsoft Visual C++ or Visual Basic) which is very important for future work and improvement;
- Ease of integration with existing robotic applications available;
- Enables the use of Simulink, which is a package supplied with MATLAB, uses a graphical data-flow language to represent the mathematics of signal processing and control theory. The Simulink package is not used within our approach but it is very popular for designing control systems and especially on motor control;
- Code conversion to C and Java is possible with small loss (this feature has not been tested within the studies of this thesis);
- Enables embedded code generation specifically for Motorola and Texas Instruments components;
- Enables the use of Toolboxes within MATLAB, which are specialized collections of M-files (MATLAB language programs) built specifically for solving particular classes of problems.

Other than those advantages we obtain, MATLAB also brings some disadvantages but advantages are more valuable considering the studies performed.

The disadvantages of using MATLAB as the development environment are listed below:

- Slow when compared to C / C++;
- Execution of code requires MATLAB to be installed;
- Code cannot be protected easily;
- MATLAB uses a great amount of memory and thus slows down the OS.

4.3 Robotic Toolbox

MATLAB has increased its power with the use of built in toolboxes within its package. The MATLAB version 6.5.0 Release 13 includes the following built-in toolboxes:

- Communications: Design and analyze communications systems
- Control System: Design and analyze feedback control systems
- Curve Fitting: Perform model fitting and analysis

- Data Acquisition: Acquire and send out data from plug-in data acquisition boards
- Database: Exchange data with relational databases
- Filter Design: Design and analyze advanced floating-point and fixed-point filters
- Financial: Model financial data and develop financial analysis algorithms
- Fuzzy Logic: Design and simulate fuzzy logic systems
- Image Processing: Perform image processing, analysis, and algorithm development
- Instrument Control: Control and communicate with test and measurement instruments
- LMI Control: Design robust controllers using convex optimization techniques
- MATLAB Link for Code Composer Studio: Use MATLAB with RTDX enabled Texas Instruments digital signal processors
- Mapping: Analyze and visualize geographically based information
- Model Predictive Control: Control large, multivariable processes in the presence of constraints
- Mu-Analysis and Synthesis: Design multivariable feedback controllers for systems with model uncertainty
- Neural Network: Design and simulate neural networks
- Optimization: Solve standard and large-scale optimization problems
- Partial Differential Equation: Solve and analyze partial differential equations
- Robust Control: Design robust multivariable feedback control systems
- Signal Processing: Perform signal processing, analysis, and algorithm development
- Spline: Create and manipulate spline approximation models of data
- Statistics: Apply statistical algorithms and probability models
- Symbolic Math: Perform computations using symbolic mathematics and variable-precision arithmetic
- System Identification: Create linear dynamic models from measured input-output data
- Virtual Reality: Create and manipulate virtual reality worlds from within MATLAB and Simulink
- Wavelet: Analyze, compress, and de-noise signals and images using wavelet techniques.

Other than those built-in toolboxes, there are other toolboxes implemented by individuals or workgroups for other research areas and the robotic toolbox is one of those toolboxes. The Robotic Toolbox [16] provides many functions that are useful in robotics including such things as kinematics, dynamics, and trajectory generation and it actually uses some of the functions pre-defined with the initially generated standard toolboxes. The robotic toolbox is useful for simulation and for analyzing results from experiments with real robots as well.

The robotic toolbox is written using the object-oriented programming technique. A specific robot within the robotic toolbox is actually an object (object is an unique instance of a data structure defined according to the template provided by its class. Each object has its own values for the variables belonging to its class and can respond to the functions defined by its class) of the robot class, which is actually previously generated. A class is the prototype for an object in an object-oriented language; analogous to a derived type in a procedural language. A class may also be considered to be a set of objects, which share a common structure and behavior. The structure of a class is determined by the class variables, which represent the state of an object of that and the behavior is given by a set of methods (functions) associated with the class.

In order to use the robotic toolbox, PUMA 760 instance shall be created, that is we have to define a new object using the robot class. In other words, we have to define and describe our PUMA 760 series robot to the robotic toolbox, using its specifications. Thus a PUMA 760 robot definition has been generated using the results found and described within Chapter 2 and section 2.6 “Model for PUMA 700 Series”, in which we have assigned the link coordinate frame and determined the Denavit and Hartenberg (DH) [11,12] parameters of the robot.

Technically we have defined a PUMA 760 object in the current workspace of MATLAB, which described the kinematical characteristics of a Unimation Puma 760 manipulator using standard DH conventions, thus each link was defined and described to the robotic toolbox software.

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. Variables can be added to the workspace by using functions, running M-files, and loading saved workspaces.

```
>> L{1} = link([ pi/2      0       0       0       0], 'standard');
>> L{2} = link([ 0         .4318   0       0       0], 'standard');
>>L{3} = link([-pi/2       .0203   0       .15005  0], 'standard');
>>L{4} = link([pi/2        0       0       .4318   0], 'standard');
>>L{5} = link([-pi/2       0       0       0       0], 'standard');
>>L{6} = link([0           0       0       0       0], 'standard');.
>> L{6} = link([0           0       0       0       0], 'standard');
```

Using the object link definition above and the kinematical theory described within Chapter 2 (the transformation matrices), we have managed to plot the PUMA 760 figure, which is actually the same for all PUMA 700 series robots since their kinematical characteristics are the same.

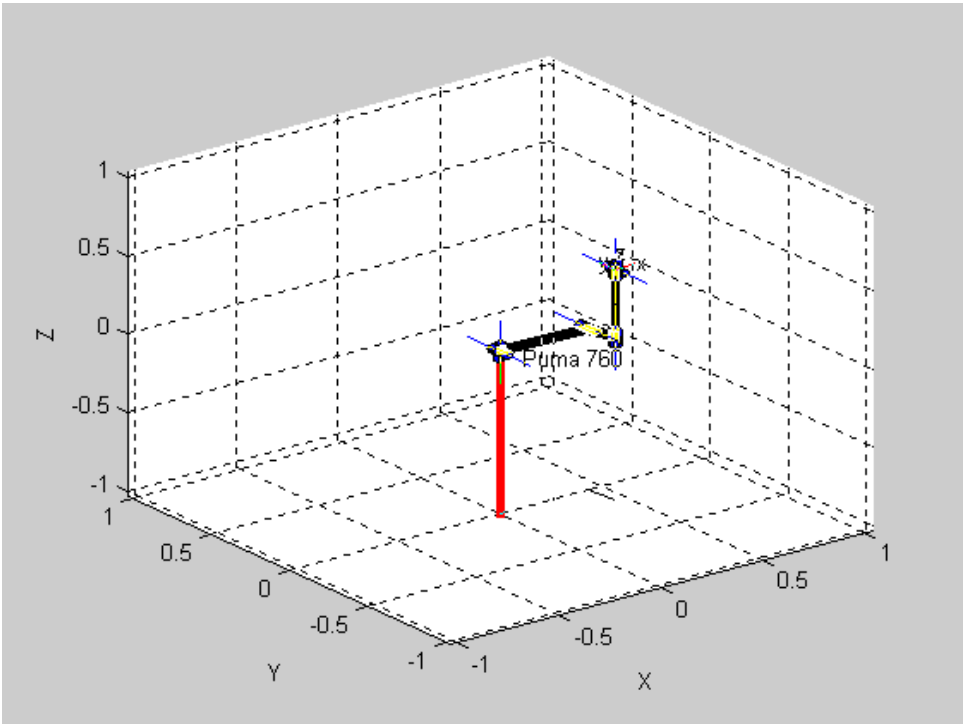


Fig.4.1 PUMA 760 Plotting

Translating the base of the PUMA with the "transl" function can modify the base position of the robot. This function returns the translational part of a homogenous transform as a 3-element column vector.

```
>> transl([-0.5 -0.5 0])

ans =
   1.0000         0         0   -0.5000
   0         1.0000         0   -0.5000
   0         0         1.0000         0
   0         0         0         1.0000
```

According to the robot object definition, the base is located at [0 0 0] in the cartesian coordinate space by default, since while creating the robot we have defined the base coordinates of the robot as [0 0 0] in cartesian space.

```
>> p760.base                %base transformation by default

ans =
    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1

>> p760.base=p760.base*transl([-0.5 -0.5 0]);

>> p760.base                %translated base transformation

ans =
         1.00         0         0        -0.50
         0         1.00         0        -0.50
         0         0         1.00         0
         0         0         0         1.00
```

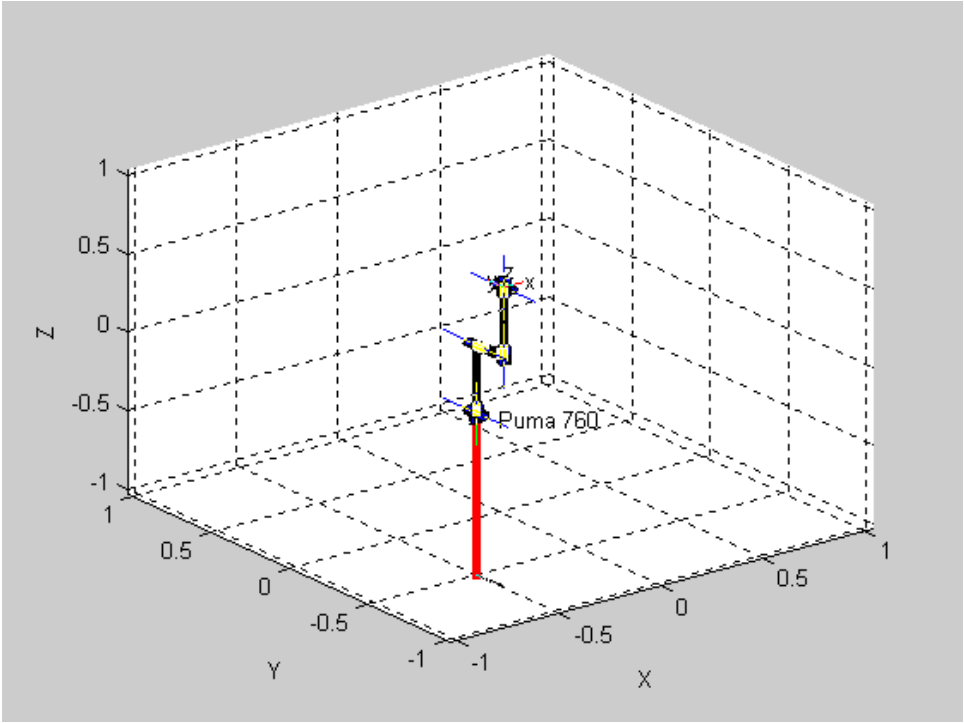


Fig.4.2 PUMA 760 with Translated Base

3D workspace enveloping, the view of the arm (in perspective or orthographical), wrist labels (xyz or noa) can be modified for the intended representation of the robot. Other than that, the user can draw also other robots as well, since the robot object is created once. Representations of the arm in subsequent instances can also be generated using the PUMA 760 robot object.

Instance is an individual object of a certain class. While a class is just the type definition, an actual usage of a class is called "instance". Each instance of a class can have different values for its instance variables, i.e. the base of the PUMA 760 object. While generating the representation for the next instance of the robot class object, user must not forget to translate the base of the robot, otherwise the bases of subsequent robot drawings will coincide.

```
>> p760_2=p760;

>> p760_2.name='2nd PUMA 760';

>> p760.base=transl([0.5 0.5 0]);

>> p760_2.base=transl([-0.5 -0.5 0]);
```

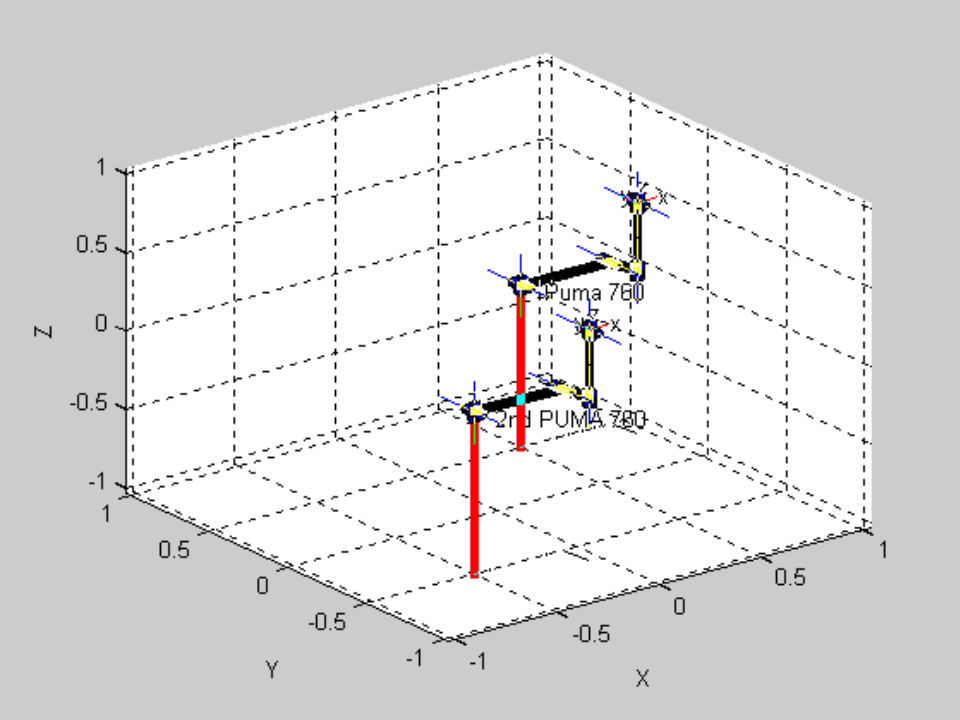


Fig.4.3 Two subsequent PUMA 760 Robot representations

This feature has been used here for testing purposes only, since we simulate and control only one robot in this study. DH parameters for each robot object has been modified and compared to the actual PUMA 760 manipulator, which fastened up the testing process of the control interface software. The changes in DH parameters effect both the simulation and the execution of robot motion on the real environment. We compared the outputs of each robot object defined within the software with the actual results obtained from the robot MARK II controller.

In simulating the robot motion, rotation matrices are needed to be implemented so that the following shows the result of those implementations for a specific example such as a rotation about x- axis for a particular rotation angle.

```
>> rotx(0.50265)  % generates  transformation matrix about the X-axis for 0.50265 (rad)

ans =
        1.0000    0         0         0
        0         0.8763   -0.4817    0
        0         0.4817    0.8763    0
        0         0         0         1.0000
```

These rotation matrices are needed to generate movements of robot in the simulation.

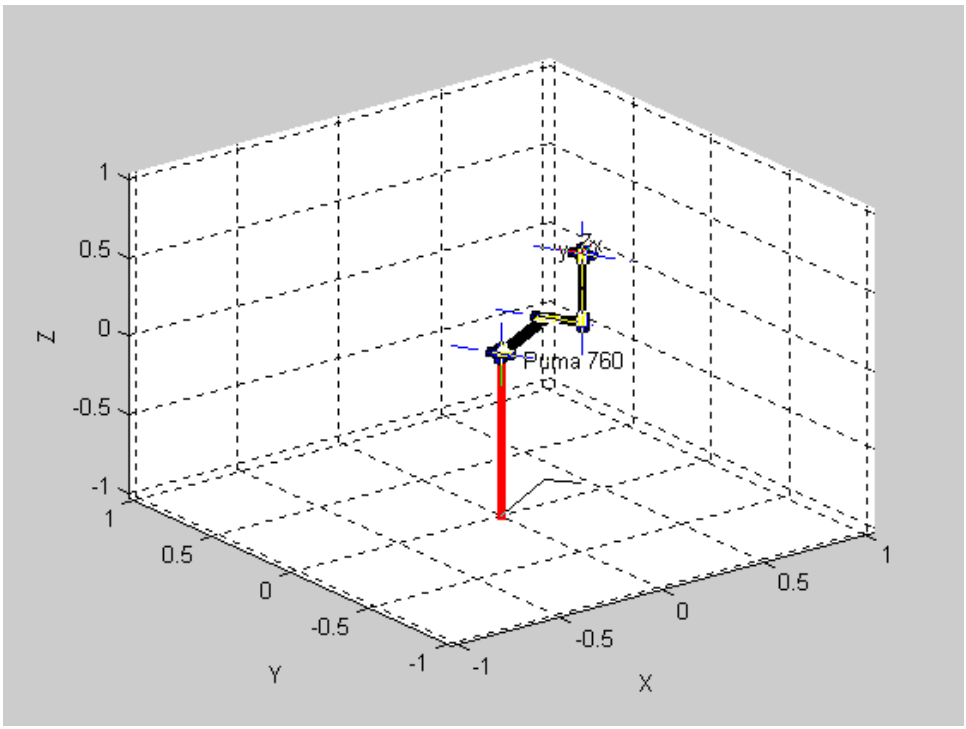


Fig.4.4 Joint 1 Rotated About its X-axis

Using all those transformation matrices a complete control interface has been designed in accordance with the features of the robotic toolbox. This interface enables the user both for calculating and simulating a robot pose. Since it is a complete interactive tool, it does not require the knowledge of VAL II robotic programming language.

The robot control interface can actually be used for any kind of manipulator for simulation purposes, but the only thing that has to be considered is that the robot object, which defines the kinematical characteristics of the robot to be represented, must be defined to the system.

Fig.4.5 displays the control panel for the robot control interface. This control panel lets the user plan the motion of the manipulator, by giving a means to modify the joint angles independently. The end-effector parameters (ax, ay, az, x, y, z) also are given within this control panel.

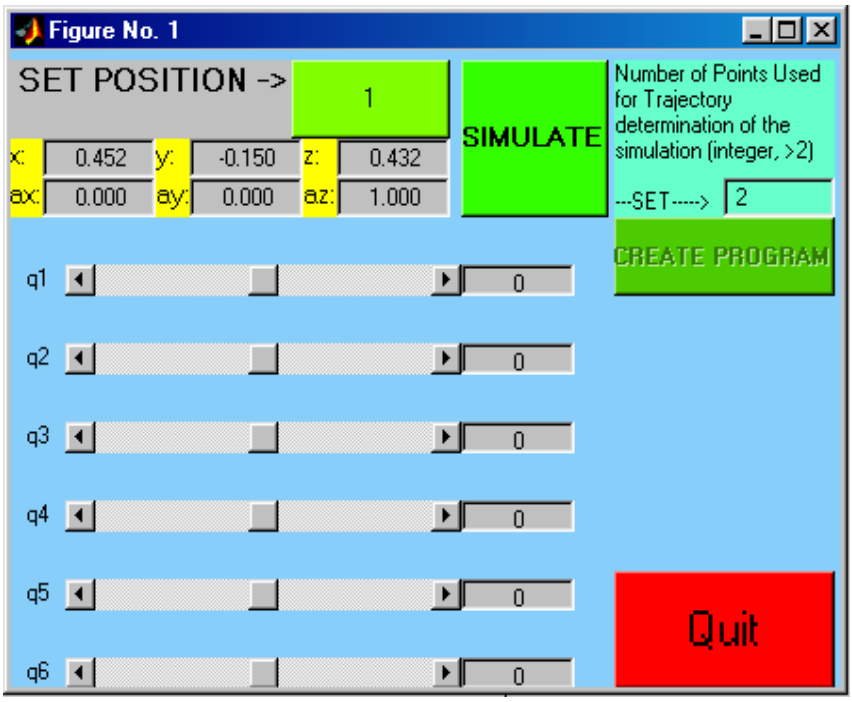


Fig.4.5 Control Interface Panel

The control panel initially starts the robot from the READY position that is the arm stretches up along the z-axis.

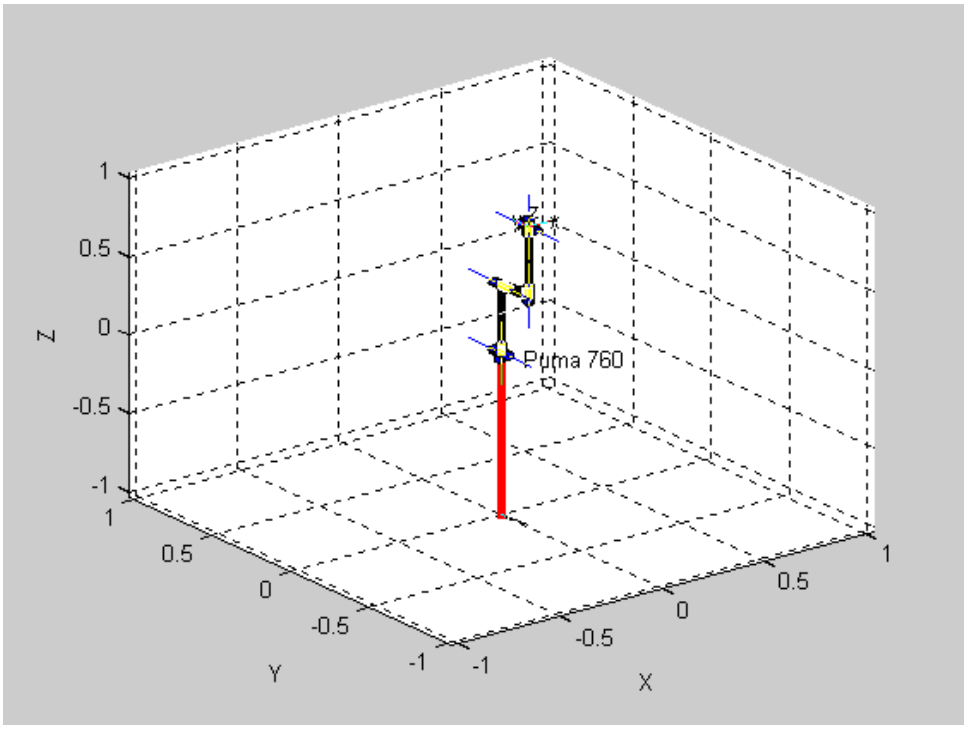


Fig.4.6 PUMA 760 at READY Position

The initial position of the robot that is set by the control software corresponds to the "DO READY" command positioning of the VAL system. The intended movements are executed via the scroll bars of the control panel. There exist six scroll bars, which actually correspond to each joint of the manipulator. Fig.4.7 shows how the scroll bars are used. The editable boxes at the right-hand side of each scroll box also display the angular value for each joint.

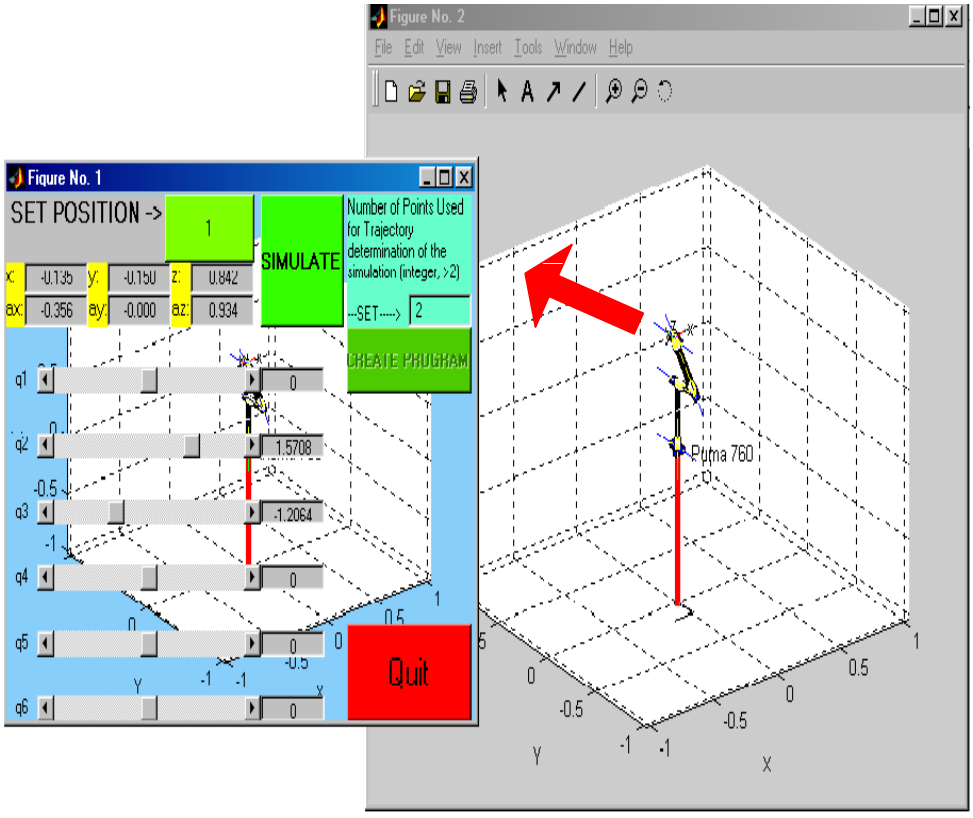


Fig.4.7 Joint Movements via the Control Panel

The robot motion is displayed in another figure window of MATLAB as the joint angular parameters are changed. The control panel also displays the motion of the robot at the bottom layer of the figure window, but in order to see the motion clearly it is recommended that the second window be used.

The user can also simulate the robot through different robot positions and poses using the 'simulate button' in the control panel. In order to do this user shall record each desired position of the robot by the set position button, it is the users choice to record as many positions. The number of points to be traced with the simulation is determined next, but the default is 2. That is the robot moves directly from 'position x' to 'position x+1', and no point is considered in between. The best choice in setting the number of points (Fig.4.8) is to set greater values for greater position distances considering end-effector positions after each set of robot position.

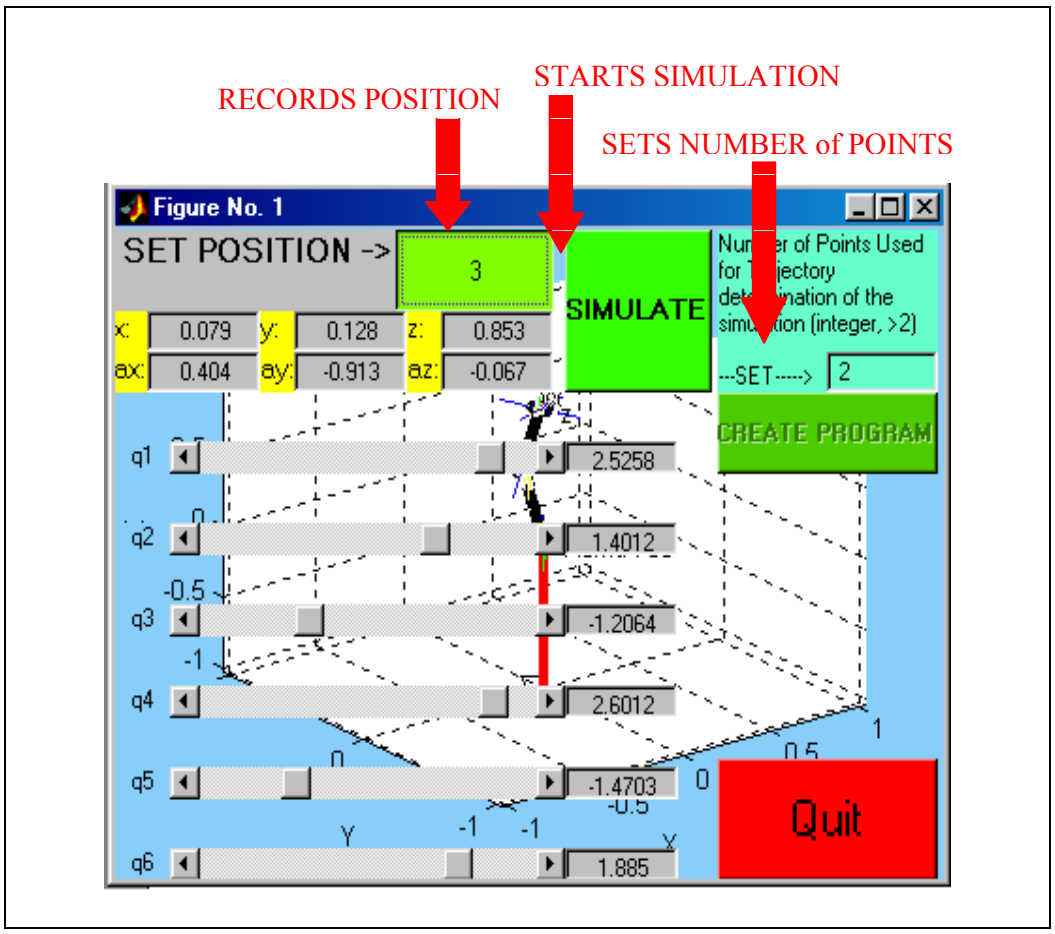


Fig.4.8 Simulation of Movements

If we move joint 1 from the angular ready position qr =[0 1.5708 -1.5708 0 0 0] to qd=[0.5655 1.5708 -1.5708 0 0 0] and set the number of points to 10, we end up with the following angular positions,

Initial Position (READY)	0.0000	1.5708	-1.5708	0	0	0
	0.0065	1.5708	1.5708	0	0	0
	0.0432	1.5708	1.5708	0	0	0
	0.1187	1.5708	1.5708	0	0	0
	0.2243	1.5708	1.5708	0	0	0
	0.3412	1.5708	1.5708	0	0	0
	0.4468	1.5708	1.5708	0	0	0
	0.5223	1.5708	1.5708	0	0	0
	0.5590	1.5708	1.5708	0	0	0

Destination Position	0.5655	1.5708	-1.5708	0	0	0

Thus, the joint space trajectory determination between initial and destination positions is possible. Each point between the initial position and the destination position can be calculated and described by the position of the robot in terms of joint positions/angles is Joint Space. The same calculations can also be carried out for cartesian trajectory determination between two positions as well, but this time the trajectory will be identified in terms of Cartesian coordinates.

The VAL commands to be transmitted to the PUMA controller via the supervisory communication port can be automatically generated after the simulation process; user is not involved with the program or command generation process. User creates the program, which is actually the set of VAL commands to be executed, through the control panel via the "CREATE PROGRAM" button (Fig.4.9 – Fig 4.10) so that the created VAL program is saved as a text file under any folder within the file structure of the computer. The file extension can also be changed but in order to run those VAL programs it is important that those file types are to be readable by MATLAB, these file type extensions can be .m (MATLAB file), .mat (MATLAB MAT workspace file), .doc (document file), or .txt (text file).

The repeat action value (Fig. 4.9) is used when a simulated robot motion is to be executed more then once.

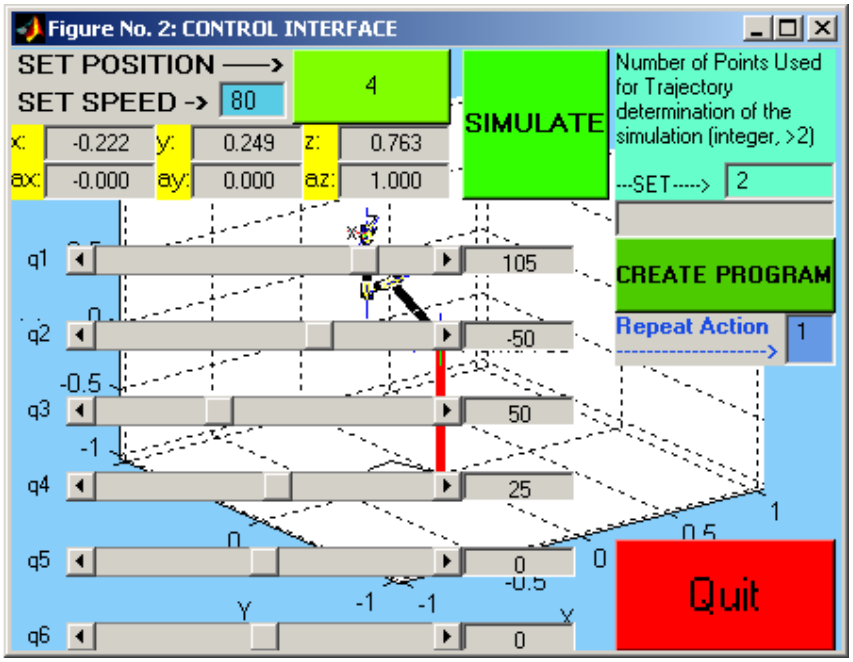


Fig.4.9 Creating VAL Programs

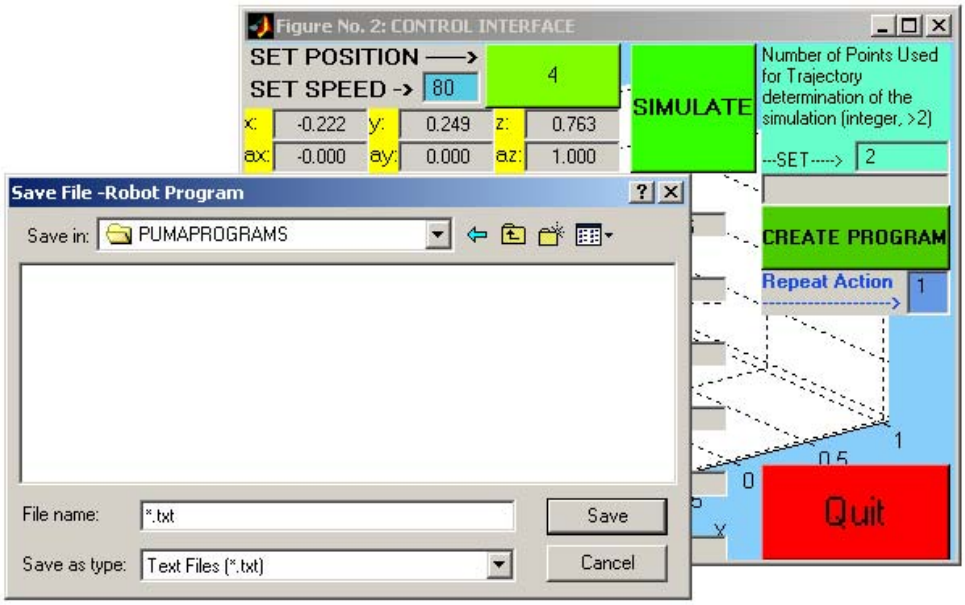


Fig.4.10 Creating VAL Programs

The content of the generated VAL program, which is actually the commands generated and prepared for transmission and execution via the supervisor, depends on the recorded position values and simulation results. A basic example for the generated VAL command set can be:

```
DO READY
SET SPEED 50
DO DRIVE 1,10,10
DO DRIVE 1,-90,80
DO MOVE #PPOINT(90,–90,0, 0,20,0)
DO DRIVE 2,-20,20
DO DRIVE 3,35,90
SPEED 80
DO DRIVE 6,266,90
DO DRIVE 6,-266,10
DO DRIVE 4,100,50
DO DRIVE 4,-35,90
```

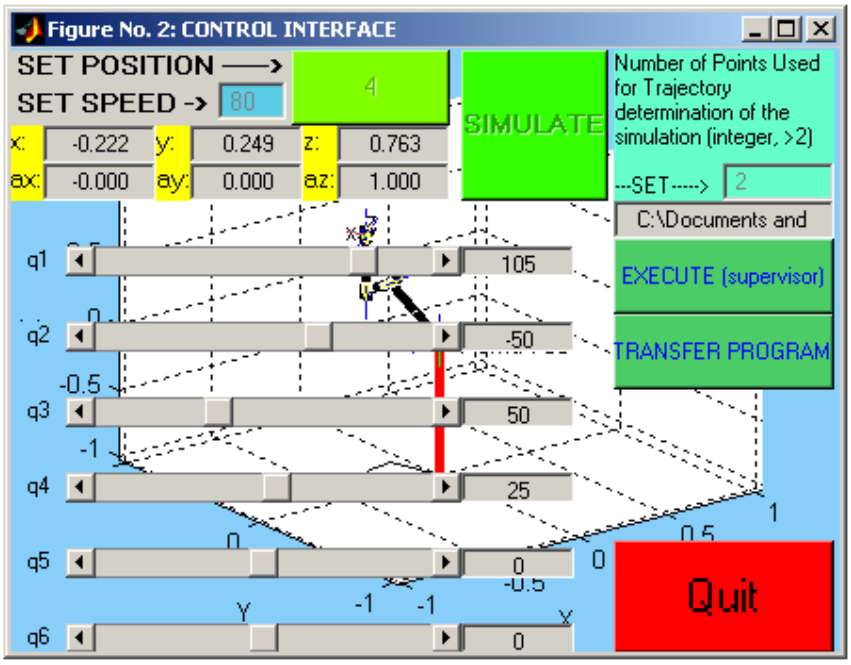


Fig. 4.11 Execution of The Automatically Generated VAL Program

The VAL program generated can be executed anytime via the "EXECUTE" button on the control panel. When this action is started, VAL Terminal I/O commands are sent to PUMA over the supervisory port. The "TRANSFER PROGRAM" executes a function that sends the overall program to the PUMA side of the communicating parties. The VAL program is then created within the memory of the controller. In order to execute the VAL programs the supervisory software, which is described within Chapter 3, shall be started before the execution.

4.4 The PUMA System Control Panel

To guide the user through the processes and to fully integrate the software modules a program module, which arranges the execution of programs, has been developed to run on top of all. It brings other features; it provides ease with operation, provides information on software processes, and guides the user on operation above the supervisory software. It is the PUMA System Control Panel and shown in Fig. 4.12.

Integrated with the software modules, which are listed within the control selection menu within the screen shown in Fig. 3.13, the supervisory system has become a complete control system for the PUMA 700 series robots. All disadvantages of supervisory control stated in Chapter 3 has also been overcome.

Fig.4.12 PUMA System Control Panel

The System Control Panel guides the user and executes required programs and functions that are required for other software modules. It also gives information about each software module. User can select which module to be executed from the control selection menu (Fig.4.13) within the system control panel, which handles all required programs underneath each module.

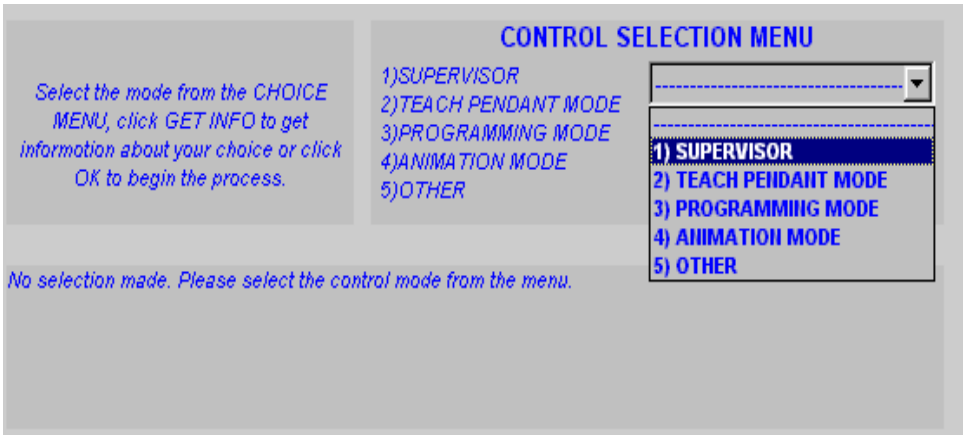


Fig.4.13 Control Selection Menu

The control selection menu consists of five items, which are:

- Supervisor
- Teach Pendant Mode
- Programming Mode
- Animation Mode
- Other

The supervisor and the teach pendant mode (control interface) were described earlier. Other than those described we have the Programming module and the animation module where as the fifth module named “Other” is left empty for future implementations. The module selection list can be enlarged with other third party modules and applications and even with the built-in toolboxes of MATLAB.

The programming module has been developed in order to let the user write and execute VAL programs using personal computers. The VAL programs can be written in text (.txt), document (.doc) or MATLAB (.m) file formats. This feature also enables the user to include external information to be included in VAL programs, which means if a trajectory has been defined by another program (e.g. AutoCAD) it can be carried and merged into VAL programs.

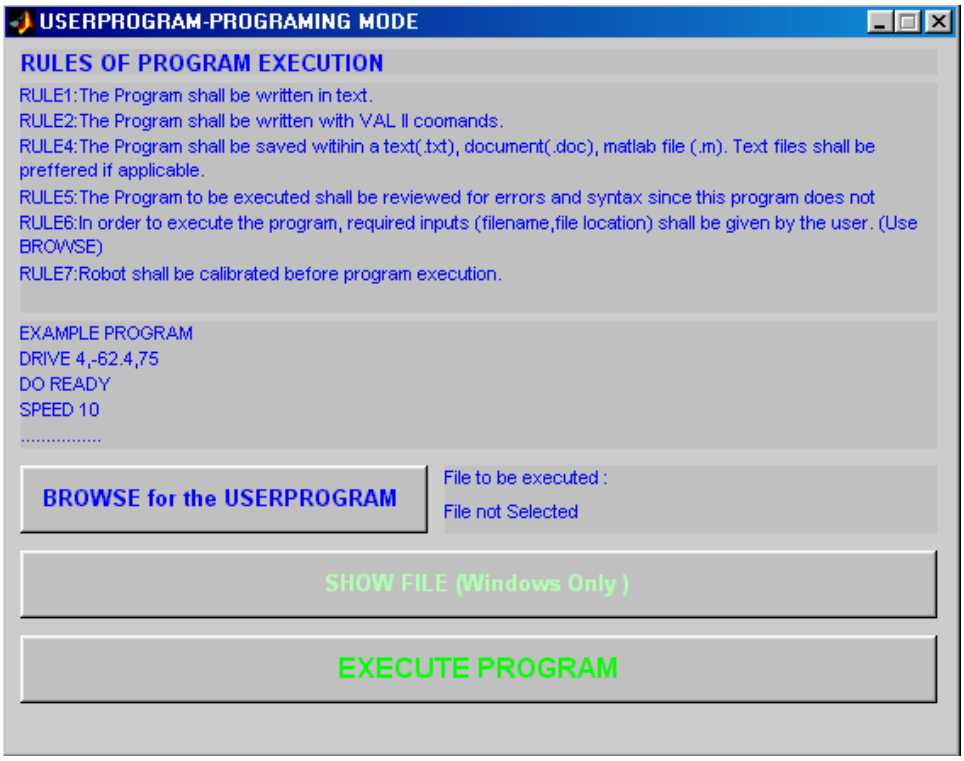


Fig.4.14 Programming Module Interface

The programming module also uses the supervisory control software (Chapter 3) in order to execute the VAL commands.

The programming module also enables VAL programming at home, which was not available before these studies since the PUMA controller is not suitable with any other operating system and it also does not have standard disk within its original configuration, thus the transportation of programs is not possible without the controller itself.

The fourth module is the animation module. It is actually a demonstration module, which uses the outcomes of another software developed for human face tracking. The demonstration is to show the capability of the new control system enabling the users to use new sensory data information like the information from a web cam. The animation module is described in Chapter 5, but since it is only for demonstration the video processing techniques will not be discussed in detail.

CHAPTER 5

5 AN INTERACTIVE CONTROL APPLICATION OF PUMA ROBOT USING THE SUPERVISORY COMMUNICATION PORT

5.1 General

In order to demonstrate the capability of the supervisory system that we have developed a different application was integrated to the overall system. With this new software integration we managed to use a video camera information in modifying the manipulator pose. The application is related to the human head/face motion, which tries to find the direction and the speed of the human motion out of pre-recorded video images.

As discussed before, the PUMA controller does not use sensory information other than those within the original manipulator such as its encoders. Before the development of the supervisory system we were not able to use camera (or any other new sensor) information for controlling the robot motion but with the ability of our supervisory software we can control our robot utilizing the data from other sources and applications. The studies held under the topic of Chapter 5, demonstrate the new ability that PUMA has gained with the capabilities of the supervisory communication.

We have transferred the output of the video system, as the speed of the action and displacement at the end of the action recorded in successive frames grabbed by the camera, to the supervisory software in order to modify the pose of the robot. The output data from the video system is obtained by processing the image frames extracted out of the video file. In order to obtain the direction and speed of motion within and between the image frames, we have used block-matching method, which can be considered as the most popular method for practical motion estimation [18].

5.2 Block – Matching Method

The basic idea of block matching is depicted in Fig. 5.1, where displacement for a pixel (n_1, n_2) in frame k, which is the present frame, is determined by considering an

N_1xN_2 block centered about (n_1, n_2) and searching frame k+1, which is the search frame for the location of the best matching block. In this basic idea the search is to find the motion vector, which is actually the displacement of the closest matching block in reference frame for a block in current frame.

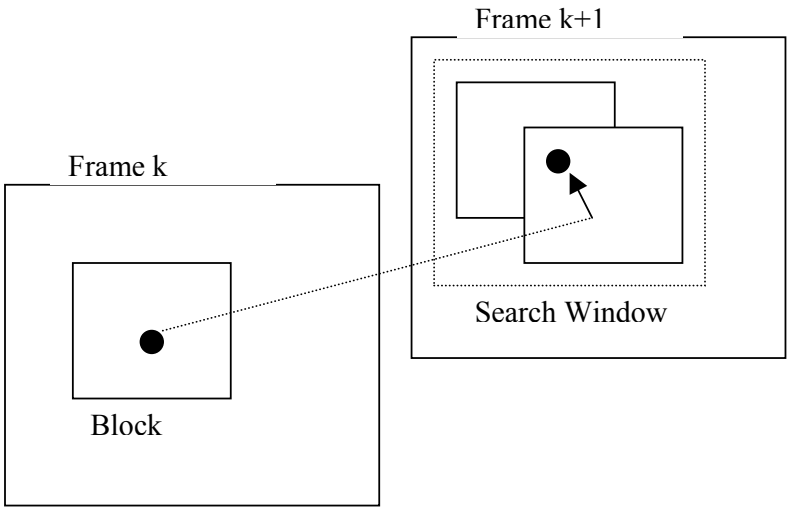


Fig. 5.1 Block - Matching

In our studies for the block-matching approach we have assumed the following while generating the algorithm and software code:

- Objects are rigid bodies; hence, object deformation can be neglected for at least a few nearby frames.
- Objects move only in translational movement for, at least, a few frames.
- Illumination is spatially and temporally uniform; hence, the observed object intensities are unchanged under movement.
- Background affects are neglected.
- The camera stability is sustained.
- Each block is viewed as an independent object.
- The motion of pixels within the same block is uniform.

The assumptions are not totally reachable but the resultant defects are reasonable.

The advantages and disadvantages of the block-matching method is listed below.

Advantages:

- It is straightforward,
- It is a regular parallel procedure good for software implementations.
- Robust (immunity to noise)

Disadvantages:

- A block may contain several moving objects.
- Criteria of search (displacement size) may not give us the true movement.

5.3 Block-Matching Software

A software module has been implemented for the block matching method. Fig. 5.2 shows the flow of the processes that are implemented.

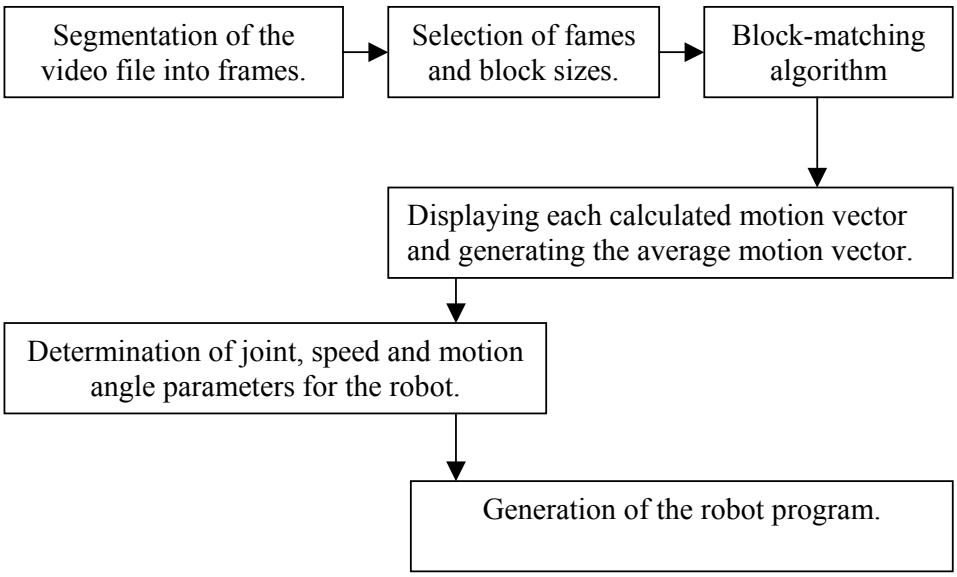


Fig. 5.2 Block-Matching Software Flow

In our studies we have used a video consisting of 10 frames. Frames extracted from the video are shown below (Fig. 5.3).

Fig. 5.3 Image Frames

After the extraction of image frames, user shall select the "Target" (reference frame) and the "Estimate" (search frame) frames using the GUI displayed below within Fig. 5.4

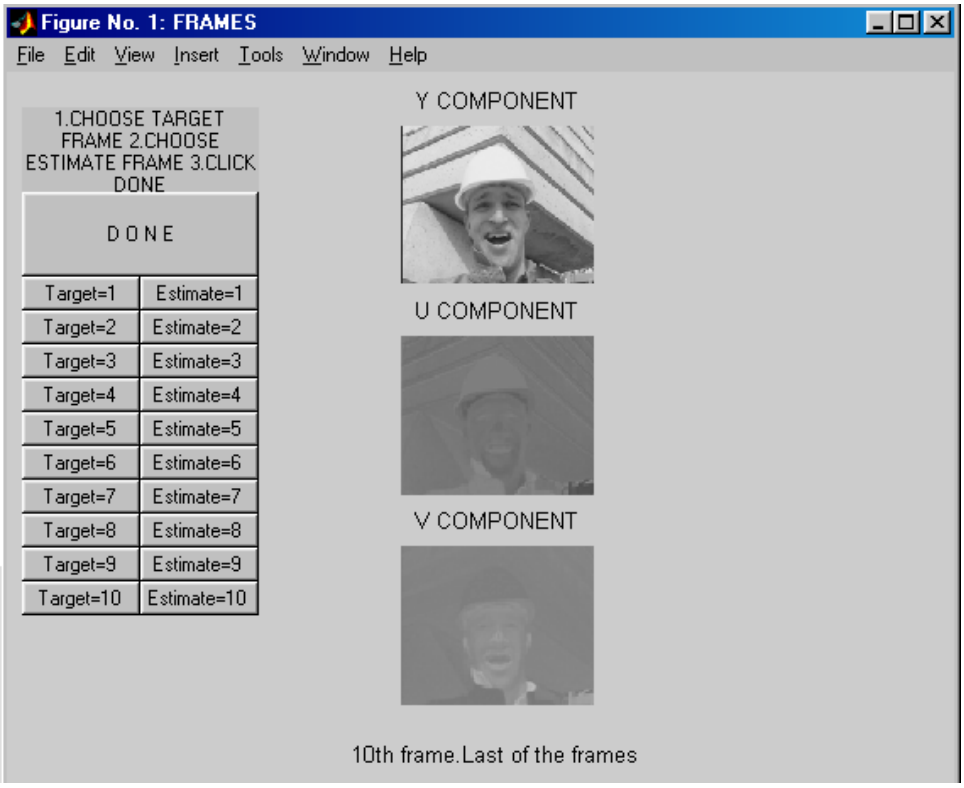


Fig. 5.4 Selection of Frames

The block size and the search window size are both selected via the GUI shown below (Fig. 5.5), after the selection of the frames that will be used for the motion estimation for the generation of robot program. The user can choose block size (8x8, 16x16, 32x32), the search window or area (8x8, 16x16, 32x32) and can also set a threshold value for the motion vectors. This threshold is used to discard motion vectors that are small in size. Thus we are trying to minimize the affects, which are caused by the instability of camera, background and illumination. It shall be noted that the threshold does not completely solve our problems. Threshold parameter shall be set by experiment, according to the video file and extracted frames otherwise it will not be able obtain correct motion vectors.

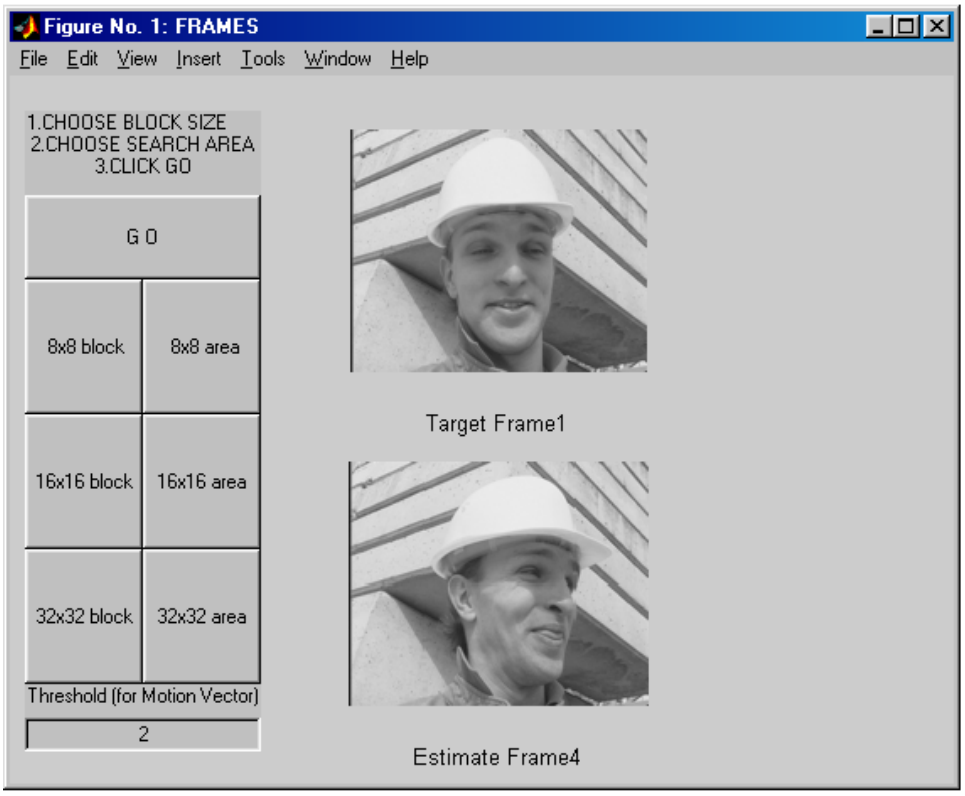


Fig. 5.5 Selection of Block and Search Area

According to the selected parameters the block-matching algorithm will produce motion vectors for each block within the selected frame. The motion vectors will then be filtered out by the use of the threshold parameter. The frames that are selected are displayed (Frames 1 and 4 for the case of Fig. 5.5) within the GUI as well.

The GUI shown below in Fig. 5.6 displays the motion vectors that carry the direction and size information.

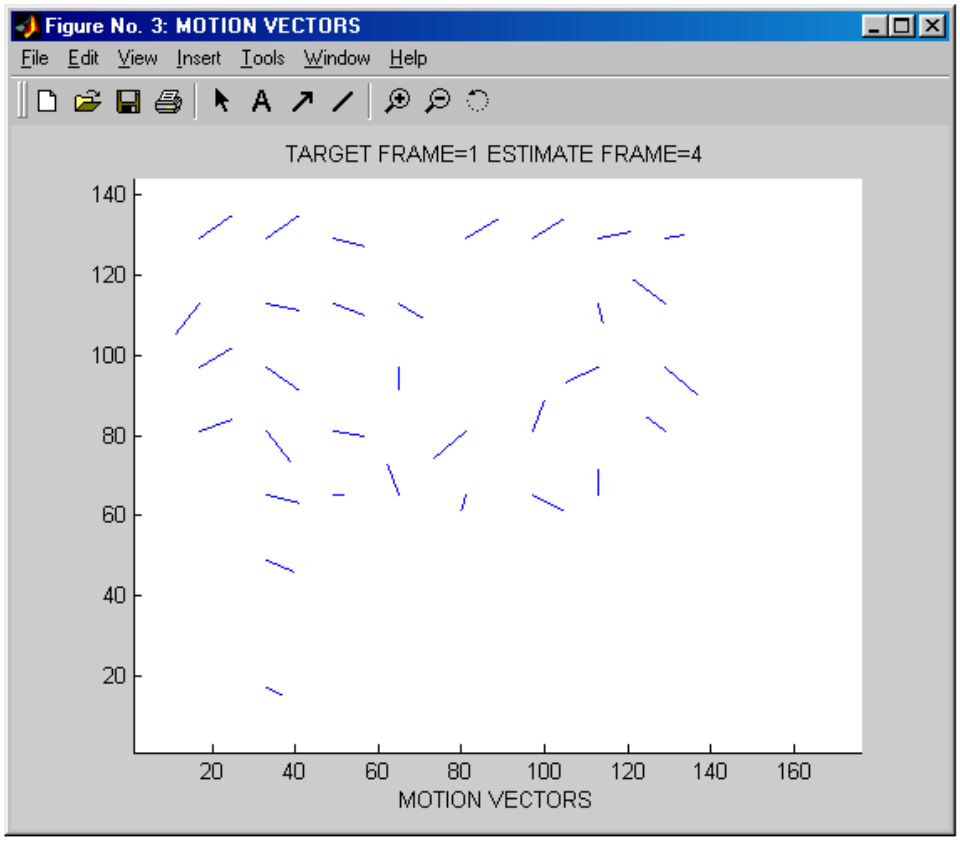


Fig. 5.6 Motion Vectors

Those motion vectors are used to generate an average motion vector, which will be used later to determine the robot movement parameters. The average motion vector is displayed within a GUI shown in Fig. 5.7. The size, which is called "delta" in the software, and the direction of motion is also displayed.

The direction is determined using the displacement observed on x-axis and y-axis. This information is also used for the calculation of the vector size, which is actually used for the speed determination of the motion since the greater displacement represent faster motion of the moving object between the selected frames. Thus displacement of pixels, size of the motion vector and speed has linear relationship between themselves.

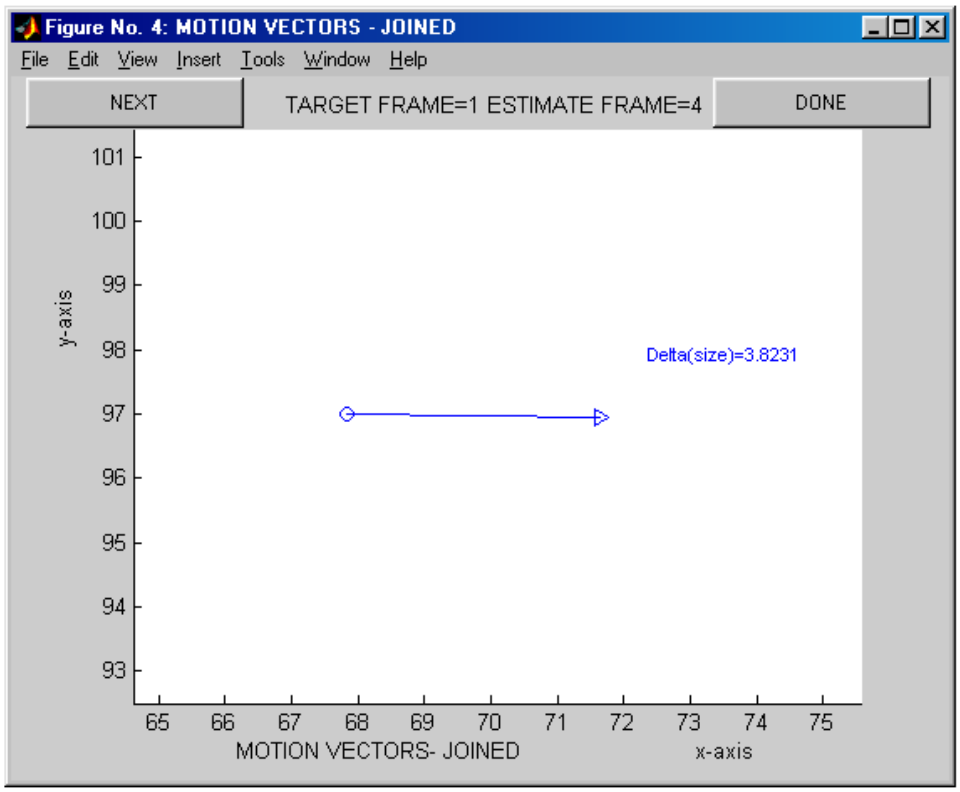


Fig. 5.7 Overall Motion Vector

Once the motion vectors are joined into one final motion vector, user can select another set of frames using the "NEXT" button, which will guide the user to the initial screen (Fig. 5.4). In order to determine the parameters required for the VAL program, user shall choose "DONE".

When we consider the case where we follow the processes for the estimation of two motions and choose the parameters as follows:

Motion Estimation 1
Target Frame=1 Estimate Frame=4, Block Size=16x16,
Search Window (area)=32x32, Threshold=2,

Motion Estimation 2
Target Frame=4 Estimate Frame=1, Block Size=16x16,
Search Window (area)=32x32, Threshold=2,

we would end up with the motion vectors shown in Fig. 5.8 and Fig. 5.9.

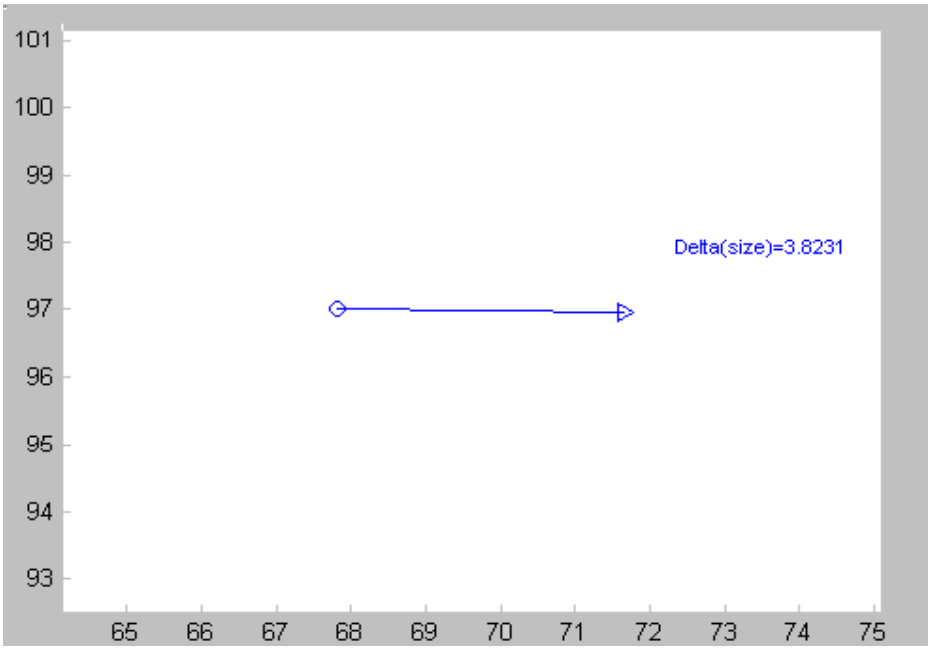


Fig. 5.8 Motion Vector 1

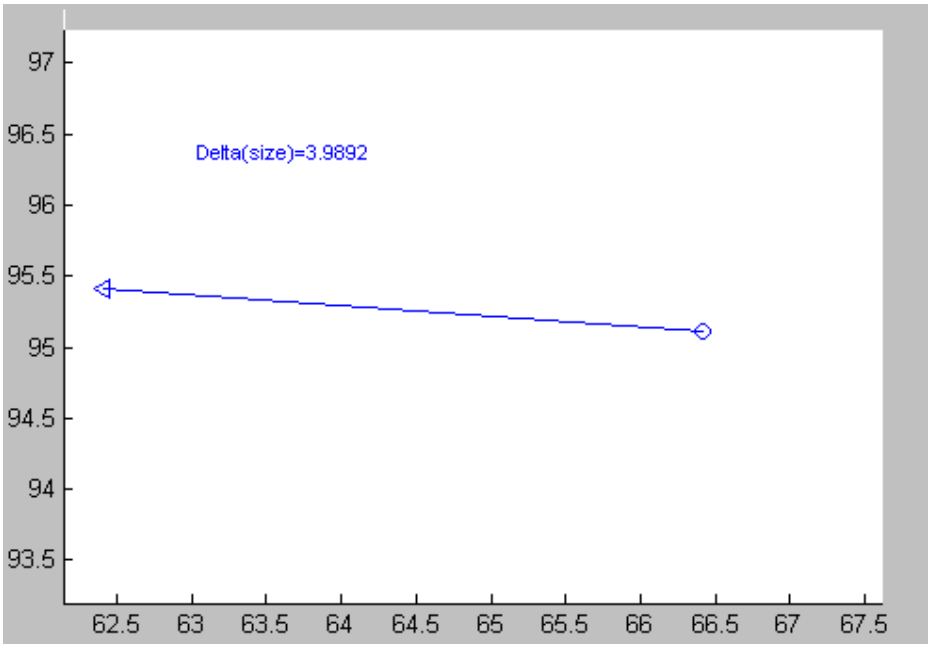


Fig. 5.9 Motion Vector 2

Using the motion vectors and the data they carry we display the Joint assignments GUI, where the user can see and modify the calculated parameters for the robot motion. Fig. 5.10 shows the joint assignment GUI.

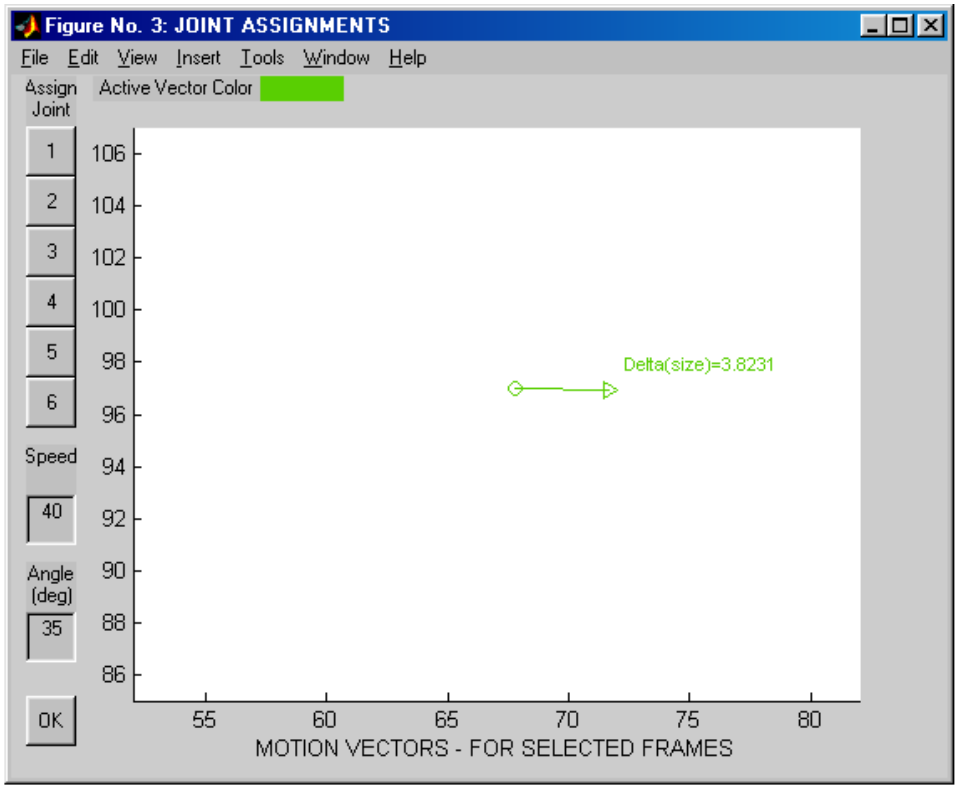


Fig. 5.10 Joint Assignment GUI

The default joint for the execution of the movement is set to joint 1, but user can change the joint itself and movement parameters using the Joint assignment GUI, which holds the required modification interface tools.

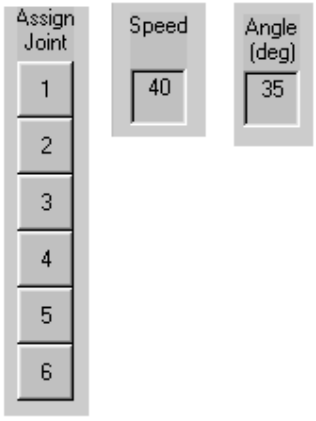


Fig. 5.11 Joint Assignment, Modification of Speed and Angle

Each of the motion vectors is displayed in different color in order to prevent confusion. Fig. 5.12 shows both of the calculated motion vectors plotted with different color and the active vector is also indicated.

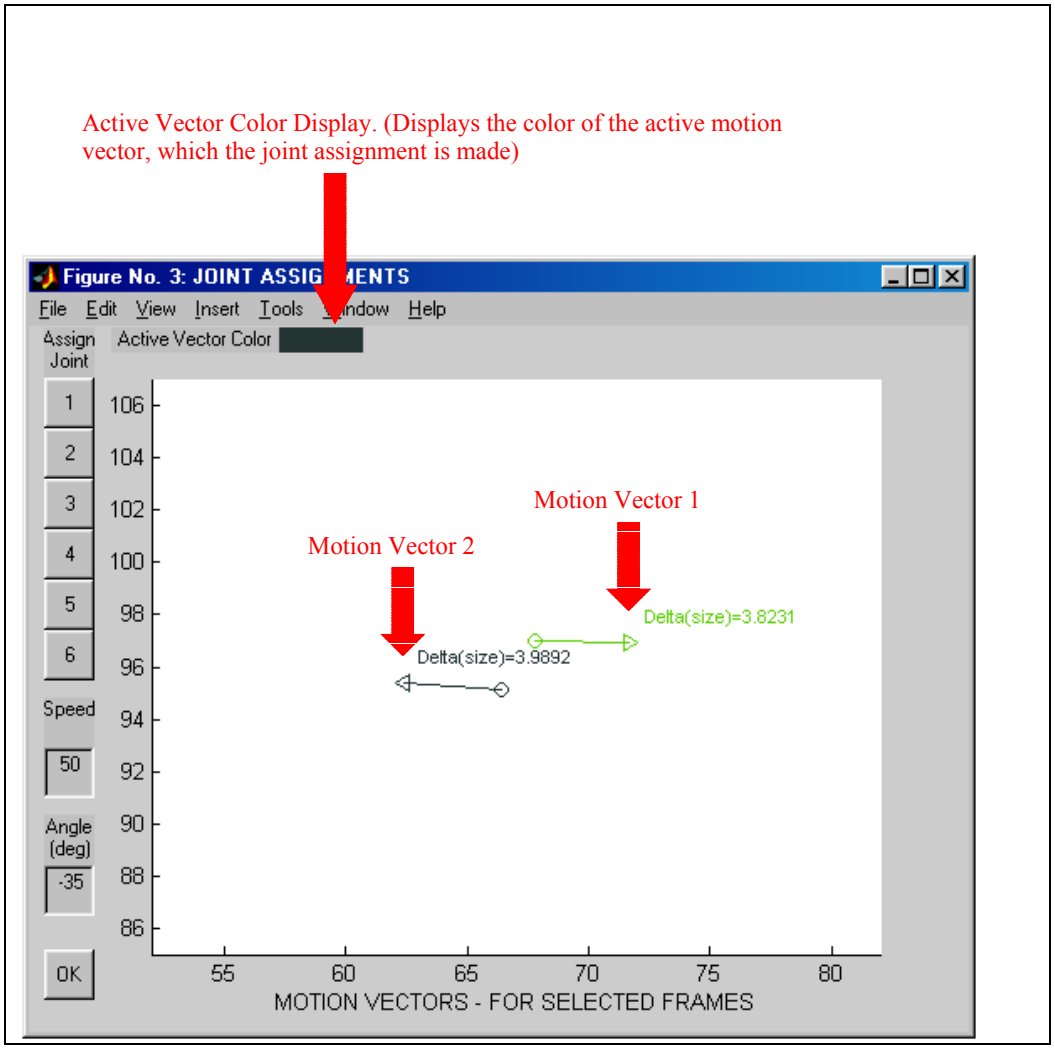


Fig. 5.12 Joint Assignment for Motion Vectors 1 and 2

Completing the parameter settings and joint assignment, the next phase is to generate the VAL programming code to execute on the PUMA system. The program generation user interface is shown in Fig. 5.13. Through this interface, user can select the file, which the VAL program is generated and written. The file generated holds basic VAL commands, which are actually PUMA system terminal input and output commands (Chapter 2). These commands can be executed via the supervisory software.

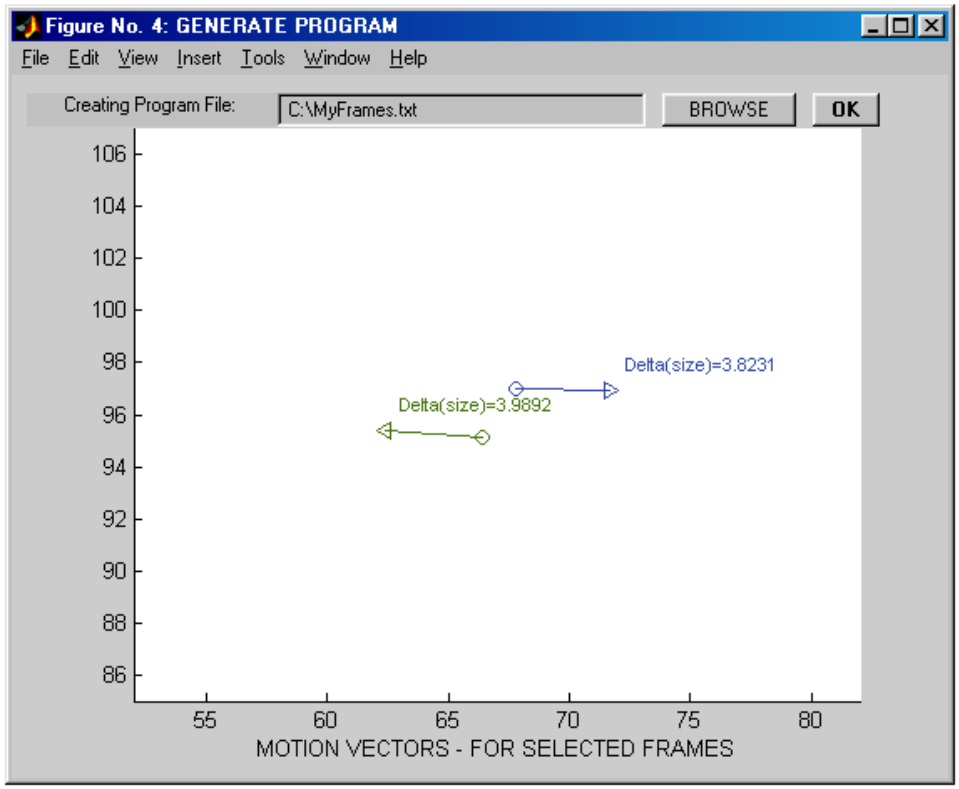


Fig. 5.13 Generating VAL Program (Terminal I/O Commands)

The VAL commands are generated after clicking the "OK" button within the "Generate Program" screen (Fig. 5.23). The commands generated would be similar to the commands shown below:

DO READY
SET SPEED 40
DO MOVE JOINT 1 45
SET SPEED 50
DO MOVE JOINT 1 -35

The commands are generated according to the information contained within the motion vectors thus the estimated motion. The selection of frames, joints and modification of speed, angle parameters are critical for the generation of each command.

Finally the commands are executed via the supervisory communication port since this software is also integrated to the supervisory control software. In order to prevent any errors within the program execution user shall start the supervisory control software. A warning message is also generated, which is to guide the user within the processes (Fig. 5.13).

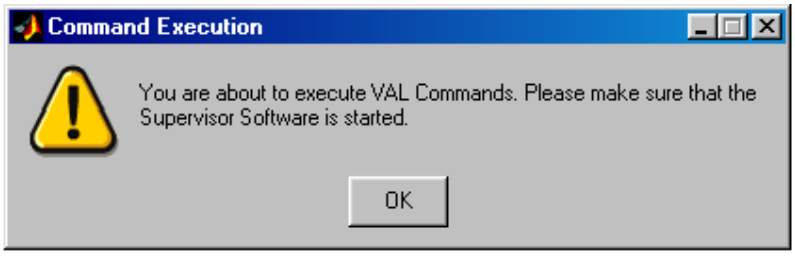


Fig. 5.14 Warning Message

VAL command execution is handled by the supervisory interface afterwards. There is variety of applications that can be integrated to the supervisory control software. The studies carried on for the motion estimation for robot motion planning only demonstrates the ability that we have gained by the use of supervisory interface. Without the supervisory software, third party applications, such as the one that we have developed, cannot be executed for the motion and path planning of the robot.

CHAPTER 6

6 CONCLUSION

6.1 Conclusion

An integrated interactive control software has been developed in the thesis for the PUMA 700 series robot communicating over its supervisory protocol. The control software running in a remote end computer has successfully established a communication line with the robot's own controller, so that remote and interactive control of the arm has been possible. The following points require to be mentioned for better evaluation of the implementation presented in this thesis study.

a. Communications port (COM1 or COM2) buffer on the external computer rarely generate errors due to a bug within MATLAB for Windows operating system. When this error occurs, the system may require start-up on the external computer side.

b. Resulting joint angular displacements in complying with the motion commands issued by the interactive control software suffer from approximately 1.5% error. The amount of these errors decreases if the arm is calibrated, but this amount increases as the number of executions of these commands increase. These errors are due to the inaccuracies in D-H parameters of the robot arm. The D-H parameters were calculated using the technical specifications [4] of PUMA 700 series robots supplied by the manufacturer, but since the robot under consideration is a rebuilt one, these parameters might have been changed during this mending process. The error magnitude may be reduced to some extent further by correcting D-H parameters based on the results obtained from a series of experiments carried out on the arm.

Establishing a communication channel between a remote end computer and the controller of the arm has improved our control ability over the PUMA manipulator. Furthermore, we have also gained a means of incorporating external sensory data providing sources such as camera, ultrasonic transmitters and receivers, etc. into this robot arm control system.

In order to have complete control ability, external feedback gear included, of PUMA without the use of VAL, the controller system needs to be removed thoroughly and replaced with a new controller system. Considering the labor and funding burden for such renewal process operating the existing system in this supervisory control scheme seems to be a feasible alternative.

6.2 Proposed Future Work

Further studies are suggested to be conducted to improve and effectiveness of this alternative control of the arm.

- Improvement on the response time of the software can be achieved by cleaning the warning indicator codes within the existing software code.

- The supervisory system (as a whole) has been tested under Windows, although MATLAB supports other operating systems, the software has not been tested for other operating systems. Thus the supervisory system can be carried into Linux or Unix environments for tests and execution.

Future work with supervisory control system: There are many future works that can be conducted over the supervisory system, since it enables the use of an external computer. Those future works may require the modification of the present supervisory system software. The proposed future works with the supervisory system are listed as follows:

- Integrating new sensory data processing software to the supervisory control system for robot control.

    - Speech processors may be used in order to transmit voice messages and commands to the robot over the supervisory communication port.
    - Image processing software may be used for obstacle determination, object pick and place applications and object tracking.

- Controlling the robot over a network or even the Internet is possible by the use of supervisory control system, but it requires modification or re-implementation of the present supervisory software.

In order to form a PUMA control network the most practical way is to implement Java servlets or applets that run in the users (clients) browsers enabling the supervisory system to obtain required parameters for controlling the PUMA inside METU ROLAB.

REFERENCES

[1] Unimation, “Programming Manual User’s Guide to VAL II ”, May 1985.

[2] Unimation, “User’s Guide to VAL II Communication with Supervisory System”, May 1985.

[3] Unimation, “User’s Guide to VAL II Real-Time Path Control”, May 1985.

[4] Unimation, “PUMA MARK II Robot 700 Series Equipment and Programming Manual”, April 1984.

[5] RP Automation Inc., PUMA Robots, www.rpautomation.com.

[6] Adept Technology Inc., “6-Axis Puma Robot Device Module (Standard and Enhanced)”, http://www.adept.com, 2003.

[7] Chia-Yu E. Wang, Wojciech K. Timoszyk, and James E. Bobrow, "Payload maximization for open chained manipulators: finding weightlifting motions for a Puma 762 robot," IEEE Transactions on Robotics and Automation, vol. 17, no. 2, pp. 218-224, 2001.

[8] G. Gini, “Caratteristiche Dei Robot”, www.elet.polimi.it, 2003.

[9] J. E. Bobrow, C-Y. E. Wang and W. K. Timoszyk, "Weightlifting Motions for a Puma 762 Robot," *IEEE International Conference on Robotics and Automation,* V23 (Video), Michigan, May 1999.

[10] Denny Oetomo, Marcelo Ang Jr., Ser Yong Lim, “Singularity Handling on Puma in Operational Space Formulation”, International Symposium on Experimental Robotics (ISER ‘2000), Honolulu, USA, December 2000

[11] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, “Robotics. Control, Sensing, Vision and Intelligence”, McGraw-Hill, 1987.

[12] R. S. Hartenberg and J. Denavit, “Kinematics Notations for Lower Pair Mechanisms Based on Matrices”, Journal of Applied Mechanics, vol. 77, pp. 215–221, June 1955.

[13] W. Bihr, “Interfacing of a force/torque sensor on a PUMA 500 Robot”, Msc. Thesis, Georgia Institute of Technology, March 1999.

[14] G. Taylor, L. Kleeman, “Flexible Self-Calibrated Visual Servoing for a Humanoid Robot”, Proceedings of the Australian Conference on Robotics and Automation 2001, pp 79-84, November 2001.

[15] DECnet, “DDCMP Functional Specification”, August 1984.

[16] P. Corke, "A Robotics Toolbox for MATLAB", IEEE Robotics and Automation Magazine, Volume 3(1), March 1996.

[17] O. Gebizlioglu, A. Ersak, “Supervisory Control of PUMA MARK II 700 Series Robot”, IJCI Proceedings of International Conference on Signal Processing, ISSN 1304-2386, Volume:1, Number:2, pp 217-222, September 2003.

[18] Murat Tekalp, “Digital Video Processing”, ISBN:0 –13-190075-7, February 1995.

[19] Mathworks Inc., “MATLAB, The Ultimate Computing Environment for Technical Education”, ISBN 0-13-184879-4, 1994.

[20] G.Tevvesz, I. Bezi, I. Olah, “A Low-cost Robot Controller and its Software Problems”, Periodica Polytechnica Electrical Engineering, pp.239-249, March 1997

[21] Trident Robotics and Research, Inc, http://pages.prodigy.net/tridentrobotics/

[22] QRTS, “Simulink Based Robotic Tool Kit for the Puma 560 Robot Arm Manual”, www.qrts.com

[23] David B. Stewart, Richard A. Volpe, Pradeep K. Khosla, “Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects”, IEEE Transactions On Software Engineering, Vol. 23, No. 12, December 1997

[24] Cannon, D. J.,Thomas, G., “Virtual Tools for Supervisory Control and Collaborative Control of Robots” Teleoperators and Virtual Environments Vol 6(1), pp 1-28, June 1997

[25] Users Manual, Foreman Hand Series, Monforte Robotics.

[26] S. Shenoy, L.Viswanadha, A. Agah, “Graphical User Interfaces For Mobile Robots”, Information and Telecommunication Technology Center

Publications, The University of Kansas, ITTC-FY2003-TR-27640-01, November 2002.

[27] Armstrong, O. Khatib, J. Burdick. “The explicit dynamic model and inertial parameters of the puma 560 arm”. IEEE Int. Conf. Robot and Automation San Francisco, California, USA, pp 510-518, April 1986.

[28] P.I. Corke and B.S. Armstrong-H'elouvry, “A search for consensus among model parameters reported for the PUMA 560 robot”. In Proceedings IEEE Int. Conf. Robotics and Automation, San Diego, pp. 1608-1614, May 1994.

[29] Dixon, W. E., E. Zergeroglu, Y. Fang, and D. M. Dawson, "Object Tracking by a Robot Manipulator: A Robust Cooperative Visual Servoing Approach," Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington, D.C., pp. 211–216., May 11–15, 2002.

[30] Dixon, W. E., D. Moses, I. D. Walker, and D. M. Dawson, "A Simulink-Based Robotic Toolkit for Simulation and Control of the PUMA 560 Robot Manipulator," Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2202-2207, Hawaii, November 2001.

[31] J. Anderson, “A Simulink Based Robotic Toolkit (SRTK) for Simulation and Control of the Barrett WAM and the Puma 560 Robot”, Master Thesis, Clemson University Control and Robotics Lab, May 2002.

[32] N. Costescu, M. Loffler, E. Zergeroglu, D. Dawson, "QRobot - A Multitasking PC Based Robot Control System", Microcomputer Applications Journal, May 1999.

[33] N. Costescu, M. Loffler, E. Zergeroglu, D. Dawson, "QRobot - A Multitasking PC Based Robot Control System", Proc. of the IEEE Conference on Control Applications, Trieste, Italy, pp 887-891, September 1998.

[34] N. Costescu and D. M. Dawson, "QMotor 2.0 - A PC Based Real-Time Multitasking Graphical Control Environment", Proc. of the American Control Conference, Philadelphia, PA, pp. 1266-1270, June 1998.

[35] S. Dindaroglu "Control of PUMA MARK II Robot with an External Computer", MSc. Thesis, Middle East Technical University, December 2002.

A. APPENDIX A

A Troubleshooting Charts

Troubleshooting charts are to be followed in order to diagnose the faults. The available troubleshooting charts are joined into a one chart and below in Table A.1, the whole chart is given.

Table A.1 Troubleshooting Chart

ADC dead Jt (n)	Defective boards	Replace boards in the following sequence, checking if problem priests after each replacement 1 Digital servo board for indicated joint 2 "B" interface board 3 "A" interface board Replace faulty board.
*ADC malfunction	Malfunction in analog input module	Attempt input again. If error repeats, hardware module should be replaced.
*ALTER aborted	a. Negative control byte received; computer cannot enter ALTER mode b. Error occurs while ALTER mode is active (more than three communication errors have occurred in succession)	a. Check external computer to ensure it is following proper communications protocol b. Check communications line to ensure it is not excessively noisy
ARM POWER off	a. ARM POWER is not turned on b. System malfunction	a. Turn on ARM POWER and reenter last command b .ARM POWER does not come on.
BAD POT Jt n	a. Faulty calibration b. Dirty or dry potentiometer c. Faulty potentiometer in	a. Run DIA POTCAL. Create overlay as required b. Clean and lubricate potentiometer c. Replace entire motor assembly on indicated joint

	indicated joint d. Faulty PC boards	 d. Check seating. Replace boards in the following sequence and check problem after each step: 1 "B" Interface board 3 "A" interface board 3 Digital servo board Exchange boards to isolate fault 4 LSI-11 processor board Replace faulty board
*Brake release enabled	a. Top panel switch enabled b. Faulty toggle switch c. Faulty PC boards	a. Move switch to OFF position b. Check switch for continuity; replace switch tray is faulty c. Check seating. Replace boards in the following sequence and recheck problem after each step: 1 Power amplifier control board 2 High power function assembly 3 "B" interface board 4 "A" interface board Replace faulty board
*Calibration will move our of range Jt n	Arm to close to end of range	Reposition arm using teach pendant in JONT mode and repeat CALIBRATE command
CHECKSUM Error	a. Diskette is use is defective b. Defective or loose floppy drive connector c. Board(s) not securely seated d. Defective board(s)	a. Attempt same procedure using known good diskette b. Check connection floppy drive if connection good check continuity of cable. Replace cable id necessary c. Check boards in card cage for proper seating d. Replace boards in the following sequence, checking if problem persists after each replacement 1 Quad serial interface (DLV11-J) 2 LSI_11 microprocessor 3 "B" interface board 4 CMOS board
Clock overrun	a. Defective "B" interface board b. Defective LSI-11 board	a. Replace defective board b. Replace defective board

	c. LSI-11 affected by electrical noise.	c. Ensure all covers correctly installed
Communication line already in use	Attempt is made to use a hardware communication line already in use for another purpose	Use a different communication line or discontinue its other use
CPU ERROR Trap to (n1) from kernel: (n2) (log, phy) (n3), (n4) *CPU error* Trap to (n1)from user: (n2) (log,phy) (n2), (n4)	a. Bad read from memory b. Electrical noise on lines c. Defective board(s)	a. Reset VAL II by typing "1100G" CAUTION Responding wit a Y will wipe out memory b. Isolate all high power or high frequency lines as far from robot as practical In some rate instances such lines may require additional shielding c. Replace boards in following sequence, checking if problem persists after each replacement 1 LSI-11microporvessor board 2 CMOS board Replace faulty board
Data check error	Transmission error detected while transferring information to or from external device	Attempt transfer again
Device not ready	Requested File storage not prepared to communicate with VAL II	a. On Unimation floppy disk drive, make sure drive is plugged in, diskette is inserted in drive, and it is properly formatted b. Replace disk
Directory Error	a. Incorrect disk, not properly installed, or not formatted b. Damaged disk	a. Make sure correct diskette is used, it is properly installed in drive, and it is properly formatted b. Replace disk
Envelope error Jt (n)	a. Minor joint degeneracy occurs when Jt5 is near o degrees, therefore aligning axes of Jt4 and Jt6. Error occurs as VAL a temps to reposition Jt4 and 6 simultaneously for tool orientation b. Major joint usable to maintain position along path of motion	a. Reprogram arm; include offset in Joint 5 WARNING Before proceeding to item b ensure that arm is properly supported prier to releasing joint brakes in FREE mode. Failure to mechanically support the arm will result in arm collapsing b. Ensure That payload (including tooling) is within specified limits. Place affected joint in FREE mode; ensure that brake is not dragging, and that there

		is no binding of gears or bearings. Replace power amplifier board and retry move
	c. Major joint runaway	c. Check electrical connections of affected joint. Check cables for connections of affected hone Check cables for continuity replace if faulty Replace components in the following sequence and recheck problem after each step: 1 Digital servo board Exchange boards to isolate fault 2 Servomotor 3 Power amplifier 4 Low power interconnect cable 5 Arm harness 6 Arm cable board Replace faulty component
	d. Minor joint runaway	d. Check electrical connections of affected joint Check cables for continuity; replace if faulty Replace components in the following sequence and recheck problem after each step: Digital servo board 2 Servomotor 3 Power amplifier 4 Low power interconnect cable 5 arm harness 6 Arm cable board Replace faulty component
	e. Joint stalled	e. Free joint of restraint
	f . Major joint drop	f. Check electrical connection of affected joint Check cables for continuity; replace if faulty Replace components in the following sequence and recheck problem after each step 1 Power amplifier 2 Low power interconnect cable 3 Arm harness 4 Digital servo board exchange boards to isolate fault Replace faulty component
[FATAL]		NOTE When [FATAL] appears on screen it indicates that a sufficiently serious problem had developed to cause the operation system to shut down. For

		possible system look up message alphabetically following word [FATAL]
File already open	Last file not completed successfully disk door opened to soon	Unplug disk drive from controller monetarily so that it undergoes a power up-reset.
File format error	a. Request floppy disk file not created by software b. File is corrupted	a. Use another diskette b. Reference anther file
Force processor not active	a. Force processing hardware missing b. Force processing hardware malfunctioning	a. Check that hardware is in system If Missing install hardware b. Replace with new hardware (Force processing board etc)
Function timeout Jt n	a. Robot is blocked and cannot reach its destination b. Arm has run into a hardware stop c. Current overload circuit tripped d. Motor potentiometer not properly zeroed (possible loosed potentiometer cap). e. Joint stalled; arm unable to move to programmed location. f. Faulty electrical interconnection. g. Faulty PC board(s).	a. Turn on ARMPOWER and try motion again b. Change program steps or locations as necessary to avoid hardware stops c. Shut system down and allow a few moments for cooling. If problem persists, look for shorts using standard short isolation routines. d. Check potentiometer voltage. If necessary, zero potentiometer. e. Inspect operating envelope for physical barriers. Inspect drive train for mechanical problems. Inspect servomotor brake systems. f. Check seating of ribbon cable interconnecting "A" interface board and servo interface board. Replace ribbon cable if faulty. g. Check seating. Replace boards in the following sequence and recheck problem after each step. (1) Digital servo board. Exchange boards to isolate fault. (2) "B" interface board. (3) "A" interface board. Replace faulty board.
*Hardware not in	Hardware has not been installed in	Proceed as follows:

system.	system, or hardware malfunctioning.	(1) Save necessary programs and data. (2) Turn off system. (3) Install or repair hardware module. (4) If necessary, modify programs so as not to attempt access to this hardware.
Hardware problem in joint 1, 2, or 3 *Hardware problem in joint 4, 5, 6*	a. Temperature fault. b. Faulty power amplifier. c. Faulty power amplifier control board. d. Shorted servomotor.	a. Turn system off and allow time for cooling; then reinitialize. b. Inspect controller top panel and power amplifier. If power amplifier assembly is faulty, replace. c. Check seating; replace board if faulty. d. Replace servomotor.
Initialization error	a. Current overload circuit tripped. b. Board(s) not securely seated. c. Defective board(s). d. Blown fuse in low voltage power supply	a. Shut system down and allow a few moments for cooling. If problem persists, look for shorts using standard short isolation routines. b. Check boards in card cage for proper seating. c. Replace boards in the following sequence, checking if problem persists after each replacement. (1) "B" interface board. (2) CMOS board. (3) Quad serial interface board. d. Check connector P23 at CRT/TTY for following voltage levels: (1) Pins 3 & 5 = +12 vdc. (2) Pins 1 & 5 = -12 vdc. (3) Pins 4 & 5 = +5 vdc. NOTE Lack of voltage readings is indicative of a blown low voltage power supply fuse. Replace fuse if necessary.
Logical unit busy	Internal error in software.	Field Service Required
Lost encoder sync	a. Defective encoder light source.	a. Replace motor on affected joint.

Jt (n)	b. Faulty encoder disk. c. Faulty digital servo board. d. Faulty arm cable assembly. e. Electrical noise.	b. Replace motor on affected joint. c. Check seating. Exchange boards to isolate fault. Replace faulty board. d. Check seating; replace assembly e. Install shield from interconnecting cable.
Memory error Address (nnnnnn)	a. Operator error. b. Weak CMOS batteries. c. CMOS board defective or not seated securely.	a. Check that no attempt has been made to store data beyond the capacity of the CMOS memory. b. Check CMOS battery charge. Replace batteries if necessary. c. Check to ensure secure seating of CMOS board. Replace defective CMOS board.
Motor hot Jt (n)	Drive motor for indicated joint overheats.	Proceed as follows: (1) Turn off ARM POWER to allow motor to cool for a few minutes. (2) Turn on ARM POWER and restart program. (3) Reduce robot motion speed or reduce carried load.
Motor stalled Jt (n)	Robot arm restrained from moving to programmed position.	Free arm of restraint.
Network closed	a. Network has not opened. b. Network closed by supervisory computer.	a. Open network or direct device not to use network. b. Reissue network opening message packet from host.
No blob (Univision)	No blob found corresponding to blob number given in a TRAIN, VINFO, or CLEARGRIP command.	If no blob is found, use PICTURE command to save new camera image and use VINFO to display list of all blobs seen. If no blobs are listed check for proper hardware operation (e.g., use video monitor to view image directly from camera). If message resulted from CLEARGRIP, make sure blob number parameter given is returned by a recent LOCATE or FINDHEAP.
No zero index Jt (n)	a. Defective digital servo board. b. Defective encoder.	a. Replace defective digital servo board of affected joint. b. Replace motor of affected joint.
Open failure	a. Remote computer is not responding.	a. Check that remote computer is working properly.

	b. Hardware problem in communications line.	b. Check that communications hardware is working properly. Retry command.
Out of range Jt (n)	a. Arm outside of or adjacent to robot arm operating envelope. b. Faulty potentiometer. c. Faulty electrical interconnection in potentiometer signals. d. Faulty PC board(s).	a. Reposition arm. b. Exchange harness connections to isolate servomotor. Clean and lubricate potentiometer. Replace servomotor if faulty. c. Check cables for continuity; replace if faulty. d. Check seating. Replace boards in the following sequence and recheck problem after each step: (1) "B" interface board. (2) "A" interface board. (3) Digital servo board Exchange boards to isolate fault. (4) LSI-11 processor board.
*Page fault from kernels: (n1) (log,phy) (n2), (n3) *Page fault from user: (n1) (log,phy) (n2),(n3)	Bug in software.	Restart system software by typing "1100G". CAUTION Responding with a Y will wipe out memory.
Problem with Amplifier C	Hardware interface to proportional hand is malfunctioning.	Requires field service
Problem with 40 volt power	a. Blown fuse on high-power function assembly. b. Faulty PC board(s).	a. Replace fuse. b. Check seating. Replace faulty boards in the following sequence and recheck problem after each step: (1) Power amplifier assembly. (2) Power amplifier control board. (3) High-power function Replace faulty board.
*Panic button pressed. *	OFF button on teach pendant pressed.	Reselect COMP mode on teach pendant before resuming program execution.
Servo dead Jt (n)	a. Current overload circuit tripped. b. Overheated amplifier.	a. Shut system down and allow a few moments for cooling. If problem persists, look for shorts using standard short isolation routines. b. Shut system down and allow a few

		moments for cooling. If problem returns after a short running period, the ambient temperature may be at fault and it may be necessary to consider nonstandard cooling procedures for the robot.
	c. Defective motor.	c. Replace motor on affected joint.
	d. Defective encoder.	d. Replace motor on affected joint.
	e. Faulty electrical connection.	e. Check seating of ribbon cable connecting "A" interface board and "B" interface board. Replace ribbon cable if faulty.
	f. Faulty PC board(s).	f. Check seating. Replace boards in the following sequence and recheck problem after each step. (1) Digital servo board. (2) "B" interface board. (3) "A" interface board.
Servo RAM error Jt (n)	Faulty digital servo board.	Check seating. Exchange boards to isolate fault. Replace faulty board.
Stopped due to servoing error	One or more servo errors.	Troubleshoot first error described.
Storage area format error	Momentary hardware failure in user data in RAM.	Attempt to save as much data as possible onto floppy disk.
Supervisory mode disabled	Supervisory mode is automatically disabled and is ready to accept commands from the local terminal.	Verify supervisory computer is working properly. Use ENABLE SUPERVISOR to reenter supervisory mode.
System clock dead	a. "B" interface board not seated properly or defective.	a. Check "B" interface board. Replace board.
	b. Ribbon cable faulty.	b. Replace cable.
	c. Ribbon cable connections faulty.	c. Replace cable connections.
Terminal interrupts dead	Empty back plane slots between CPU module and serial I/O modules.	Check board installation to ensure proper board placement in card cage.
Too many blobs	a. Most recent picture contains more than 11 blobs.	a. Use VINFO command to list all blobs in current camera image.
	b. Both BLACK and WHITE switches enabled.	b. Disable one switch.
	c. Picture noisy.	c. Adjust camera aperture and focus MINBLOB variable in system, or threshold setting.
	d. Scene too complex.	d. Remove extraneous material from

		camera range.
Unknown FPS error	CPU computer module malfunctioning.	Shut system down and reseat CPU in card cage. If problem persists, replace CPU board.
Unknown network error	Remote computer not sending valid data.	Refer to Supervisory Communications Manual, User's Guide to VAL II, Part 2.
Unknown prototype	Unknown prototype name used.	Use VINFO command to display list of all prototypes defined. If necessary, VDEFINE and TRAIN a new prototype.
Vision not enabled (Univision)	Vision switch not enabled.	Issue "ENABLE VISION" command or include it in the user program before first occurrence of vision instruction.
Vision time-out (Univision)	Communication problem between software and MIC vision system.	Retry preceding vision command or instruction. If error repeats, refer to following *Version system error*.
Vision system error (Univision)	Error within MIC vision system operating program.	Reload MIC vision system and try again
Wait with logical unit idle	Internal error in software.	Field service required.

B. APPENDIX B

B HARDWARE PROBLEMS ENCOUNTERED and SOLUTIONS FOUND

B.1 General

In this appendix, the initial hardware problems that came across with their solutions are stated before the results that we have gathered for the supervisory computer control. Initially, that is when we have first started our studies; the PUMA system did not function at all. As a result, we have started analyzing the system to get it back working again and we have found that the cause of our problems were hardware based.

B.2 Hardware Problems Encountered And Solutions Found

In order to get complete understanding of the robot system, its capabilities and operation, the robot was to be practiced but it was not operational. Thus four months of work has been spent to get it back working.

The controller got powered on, but had no response at all, thus we did not receive any error messages, which disabled the quick diagnoses of the failure. The troubleshooting charts [4,5] (see Appendix A) were traced, but no solution was found for the specific existing situation.

All hardware connections were checked using the circuit and connection layouts of the robot system [5], the connections were correct and connectors were functional (existing affects of corrosion were cleaned within this work). The failure diagnoses studies carried on with the control of short circuit and open circuit checks on the main processor and peripheral units interface cards. The connections and most of the components on each PCB were checked, which led to failure localization on interface card. It was found that, the problem occurred due to damage on the connections within the peripheral units interface card; jumper connections were not appropriate which ended up with different incorrect baud rates within peripheral units causing communication failures thus the malfunctioning of the robot system. The jumper connections were corrected (all set to 9600 BAUD) and tested after a hard work of diagnose.

Note: With the diagnoses studies, it was discovered that the servo controller driver board (PCB) for joint 2 had slight cracks and scratches on connection canals. It was examined that, this deficiency did not affect the functioning of the driver. But as to minimize the risk, considering the mass parameters of the PUMA 760 arm [7], the controller board of joint 2 has been replaced with the one for the joint 3.

After the corrections made, system is powered on and the VAL is loaded, from the floppy to the controller memory. The procedure for loading VAL is given below:

1. Turn control cabinet power on and allow to warm up,
2. Turn three-way MODE switch on front of controller to RUN,
3. Turn controller power on with POWER ON switch,
4. CRT terminal will print
   *** VAL-II Boot B760.2.0 AUG81 ***
   Load VAL-II from floppy (Y/N)?
5. Insert operating disk into floppy drive and press —Y“ <enter>,
6. The Robot serial number will be asked. Enter the last four digits of the robot serial number and press enter (The robot serial number is located on the label of operation disk and on the base of the arm),
7. At this point numbers should appear across the screen as the program is loading, if this is not the case, turn power off and check all connections and be sure you followed the above steps carefully, if this still fails use the troubleshooting charts (Appendix A),
8. After the program has been installed, you should initialize and calibrate the robot.

The procedure given here shall be applied if controller memory has been off for approximately 30 days [4]. Otherwise the batteries of the CMOS memory card will be used so that the VAL is not required to be loaded each time the robot arm powers on. But this was not the case for our system; VAL needed to be loaded at each start-up, which took a lot of time. When examined, it was found that the batteries (2x1.2V NiCad) were dead and had leakage, thus required replacement. The batteries were replaced with new ones but due to the performance of the battery charger unit on the CMOS memory card and the capacity of the new batteries, we have only reached a period of approximately three hours, which means that the user shall load VAL after a three hours of system power off. Replacing the whole battery charger unit mounted to the PCB can improve the performance of the memory.

As the studies went on with VAL programming, two different errors, which did not occur at all executions, were received from the system. The first one was the “SERVO DEAD, Joint x” error that is the joint controller for joint x did not respond to VAL commands. Using the troubleshooting charts available, possible causes, servicing instructions and actions taken with their results for those servicing instructions are listed in Table B.1.

Table B.1 Possible causes, Servicing instructions and Actions taken with results for the "SERVO DEAD Joint x" Error.

Possible Cause	Servicing Instruction	Actions Taken and Results
Faulty digital servo board.	-Check seating. -Exchange boards to isolate fault. -Replace faulty board.	-Seating(s) are checked. They are seated properly. -Boards have been exchanged but error remained as it was. -Faulty boards cannot be replaced, since no spare is available.
Faulty electrical interconnection.	Check seating of ribbon cable interconnecting parallel interface board and servo interface board.	Ribbon connection and all other connections available to those joint controllers have been made, they are all found to be appropriate. Error remained as it was.
Faulty PC board.	-Check seating. -Replace boards.	-Seating(s) have been checked and found to be proper. -Board replacement cannot be made since no spare board is available.

Completing the necessary activities for the "SERVO DEAD" error, it is now believed but not proven that this problem is due to the unstable state of the input power. Thus there is no recommendation of any solution is available.

The second error received, which did not occur at all executions, just like the "SERVO DEAD" error was the "ENVELOPE ERROR, Joint x". This occurs if the actual position of a joint is beyond a preset position from the position command by VAL and that joint forces VAL to declare a fatal error condition. After some research, it was found that minor joint degeneracy causing "ENVELOPE ERROR" is a common failure above PUMA and PUMA-like robots. There exist three singularities [8,10]:

- Alignment singularity (wrist is as close to the axis of joint 1 as it can get),
- Elbow singularity (elbow is fully extended or folded up; the latter is not possible because of joint limits),
- Wrist singularity (the axes of joints 4 and 6 are aligned).

These singularities can happen individually, or as a combination of two of even three at the same time [10]. The most common occurrence of those singularities is actually the wrist singularity which occurs when joint 5 is at 0 degrees, therefore aligning axes of joints 4 and 6. Error occurs as VAL attempts to reposition joints 4 and 6 simultaneously for tool orientation. The only solution that is available is; to

reprogram the robot arm, including offset in the robot arm. Thus users obtaining this error shall reprogram the robot arm giving an offset value for the joint 5.

Completing the available solutions for the problems encountered, which are mentioned above, the work mostly concentrated on VAL programming and supervisory control implementation.

C. APPENDIX C

C Image Frames Extracted From the Video File

In order to apply the block-matching method, we have extracted the image frames from the video file. The image frames are given below for clear view and understanding.

Fig. C.1 Frame 1

Fig. C.2 Frame 2

Fig. C.3 Frame 3

Fig. C.4 Frame 4

Fig. C.5 Frame 5

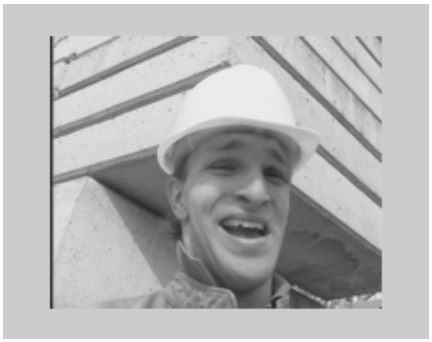


Fig. C.6 Frame 6

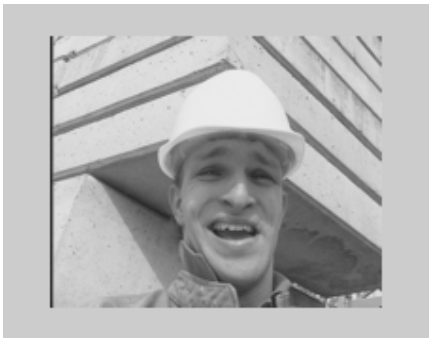


Fig. C.7 Frame 7

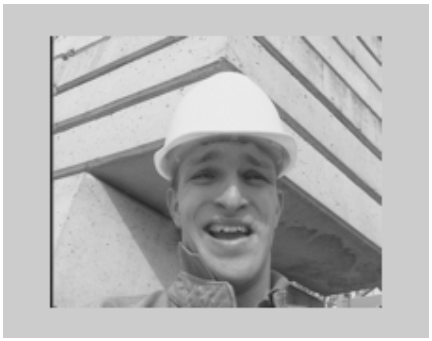


Fig. C.8 Frame 8

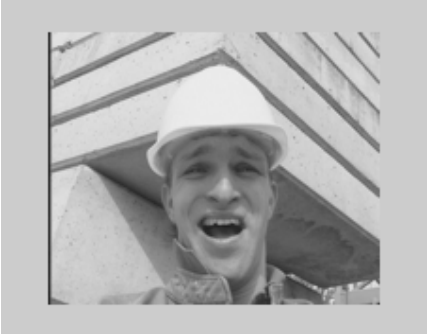


Fig. C.9 Frame 9

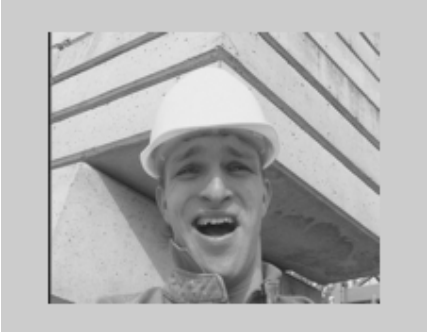


Fig. C.10 Frame 10

D. APPENDIX D

D Thesis Studies Planning

In order to achieve the goals and objectives of the thesis (Chapter 1), planning shall be made. This planning only covers tasks not the duration and effort parameters of those tasks.

The tasks to conclude are determined to follow all through the studies conducted. Those tasks are determined according to our purpose, the problems encountered and the work or activity done. This section describes those tasks and associated sub tasks to be followed.

At first the problem that was closing our way to modification was the not operational state of the PUMA arm. This operational status of robot had to be changed which happened to be the first task. Under this task there are sub tasks that are stated below:

- Understanding the PUMA 760 robot arm.
- Investigations on the problem or problems causing PUMA not to operate.
- Determining and implementing the solutions to the problem.
- Testing the implemented solutions.

Secondly the PUMA robot was taken under research in order to understand its structure, capabilities, communication modes and its software, which is VAL II. This task has the following sub tasks:

- Research on PUMA robots, their types.
- Determining the physical specifications.
- Understanding of electrical connections, the control cabinet in which the peripherals are located.
- Understanding the Unimation's software package VAL II, its capabilities.
- Real-time practicing of VAL II programming.

After those tasks stated above, we had to understand how the PUMA robot arm communicated with the external world so the third main task has come across, which was understanding the communications available and determining the right communication mode for control. The sub tasks for this task are:

- Research on PUMA robots external communications.
- Understanding each communication modes available and their costs.
- Implementing the chosen communication mode.
- Testing the software implementation.

The fourth main task after completing the communication externally to the PUMA was to develop user-friendly software enabling the user to control the robot. The sub tasks for this are stated below:

- Review of kinematics and dynamics knowledge for robots.
- Research on robot simulators available.
- Choosing the development environment.
- Implementation and modification of software.
- Testing the software.
- Integrating simulator software to the communication software.
- Testing the integrated modules of software.

As to reach the goals of this thesis a third part application for the supervisory system is implemented and integrated to the overall system as well. This application, which detects and outputs difference vectors out of human motion shows the new ability of robot. In order to complete this task following sub tasks are concluded:

- Review of code for the application (The motion estimation implementation was generated within the studies held for the Video Processing course held in METU Electrical Electronics Engineering Department).
- Integration of software.
- Testing the whole system.

Completion of the thesis is the last task, which includes thesis documentation and presentation.

Table D.1, which shows the task planning with main and sub tasks, is followed throughout the thesis studies.

Table D.1 Task List

Main task number:	Sub task number:	Definition of Task:
1	**-**	**Fixing the un-operational robot to operate.**
-	1	Understanding the PUMA 760 robot arm.
-	2	Investigations of the problem or problems causing PUMA not to operate.
-	3	Determining and implementing the solutions to the problem.
-	4	Testing the implemented solutions.
2	**-**	**Understanding PUMA 760 robot and its control software package, which is VAL II.**
-	1	Research on PUMA robots, their types.
-	2	Determining the physical specifications.
-	3	Understanding of electrical connections, the control cabinet in which the peripherals are located.
-	4	Understanding the Unimation's software package VAL II, its capabilities.
-	5	Real-time practicing of VAL II programming.
3	**-**	**Understanding the communications available and determining the right communication mode for the control of PUMA robot arm.**
-	1	Research on PUMA robots external communications.
-	2	Understanding each communication modes available and their costs.
-	3	Implementing the chosen communication mode.
-	4	Testing the software implementation.
4	**-**	**Developing user-friendly software enabling the user to control the PUMA robot.**
-	1	Review of kinematics and dynamics knowledge for robots.
-	2	Research on robot simulators available.
-	3	Choosing the development environment.
-	4	Testing the software.
-	5	Integrating simulator software to the communication software.
-	6	Testing the integrated modules of software.
5	**-**	**Integration of Interactive Control Application for PUMA motion control.**
-	1	Review of motion-estimation code for the application.

-	2	Integration of software.
-	3	Testing the whole system.
6	**-**	**Completion of thesis work, documentation and presentation.**

The tasks are listed following each other but actually some tasks are carried out in parallel, the task sequence and processing of these tasks for the studies of this thesis are not given.

The outcomes of each task and their related studies has been recorded and given as a whole at this thesis according to their relation with the topic of each chapter.